Universität Passau

Lehrstuhl für Mathematik
mit Schwerpunkt Digitale Bildverarbeitung

Bachelorarbeit

Gabor-Transformation für Musikdaten

Autorin	Christina Praml
Matrikelnummer	85612
Betreuer	Prof. Dr. Tomas Sauer

27. September 2023

Inhaltsverzeichnis

Abkürzungen **2**

Begriffe **2**

1 Einleitung **3**

2 Transformationen **3**
2.1 Signal . . . 3
2.2 Fouriertransformation . . . 4
2.3 Gabor-Transformation . . . 6
2.3.1 Gauß-Fenster . . . 8
2.3.2 Kurzzeit-Fouriertransformation . . . 8
2.3.3 Gabor-Transformation . . . 9
2.3.4 Spektrogramm . . . 9
2.3.5 Inverse Gabor-Transformation . . . 9
2.3.6 Heisenbergsche Unschärferelation . . . 10

3 Implementierung der Gabor-Transformation in Python **11**
3.1 Hardware . . . 11
3.2 Implementiertes Programm . . . 13
3.3 Vergleich der eigenen und vordefinierten Methoden . . . 16

4 Gabor-Transformation für Musikdaten **19**
4.1 Töne und Frequenzen in der westlichen Musik . . . 19
4.2 Synthesizer-Sounds . . . 21
4.3 Klavier-Sounds . . . 23
4.4 Gitarren . . . 26
4.4.1 Vergleich Westerngitarren . . . 27
4.4.2 Vergleich E-Gitarren . . . 28
4.4.3 Melodie: Happy Birthday . . . 30

5 Zusammenfassung **31**

Abkürzungen

FFT *Fast Fourier Transform*. 2, 13

Hz *Hertz*. 6

STFT *short-time Fourier transform*. 3, 8

WAV *Waveform Audio*. 3

Begriffe

Cooley-Tukey-Algorithmus Verfahren zur Berechnung der Fast Fourier Transform nach Cooley und Tukey [CT65].

Fast Fourier Transform Schnelle Fouriertransformation.

Frequenz Schwingungszahl von Wellen (pro Sekunde).

Hertz Maßeinheit für periodisch wiederkehrende Vorgänge pro Sekunde, Einheit der *Frequenz* und der *Sampling Rate*[1].

monochromatisch zu nur einer Spektrallinie gehörend.

Sample Zeitlicher Moment eines digitalen Audiosignals, für den ein Audiopegel existiert und der die zeitliche Auflösung des Audiosignals definiert[2].

Sampling Rate Beim Digitalisieren von analogen Audiodaten gibt die Sampling Rate an, wie oft pro Sekunde das analoge Signal abgetastet wird[1].

short-time Fourier transform Kurzzeit-Fouriertransformation.

Waveform Audio Format für digitale Audiodaten[1].

[1] `https://home.uni-leipzig.de/horst-rothe/vidlexik.htm`
[2] `https://www.bet.de/lexikon/audiosample`

1 Einleitung

Stellen Sie sich folgende alltägliche Situation vor: Sie sind unterwegs - in einem Café, einem Geschäft, im Auto - und hören ein Lied, das Ihnen so gefällt, dass Sie unbedingt wissen möchten, wie es heißt. In einem solchen Augenblick erweist sich eine kleine Anwendung auf dem Smartphone als ziemlich nützlich: die App „Shazam“. Richtet man das Mikrofon des Geräts nach dem Öffnen der App passend aus, erhält man innerhalb von Sekunden den gesuchten Songtitel und den zugehörigen Interpreten. Anhand einer kurzen Audioaufnahme kann die App Musikstücke erkennen. Vielleicht haben Sie eine ähnliche Situation schon erlebt und mit einer solchen App den Titel des gesuchten Lieds erhalten. Aber wie kann es diese App schaffen, in Sekunden den gehörten Klang zu analysieren?
Eine Möglichkeit, das zu erreichen, ist mithilfe der Gabor-Transformation.
Unter Nutzung der Fouriertransformation können mit der Gabor-Transformation - im Gegensatz zur Fouriertransformation - sowohl die zeitlichen als auch die spektralen Informationen von Signalen extrahiert werden, zum Beispiel zur Analyse von Audiodateien. Sie wird eingesetzt beim Komprimieren von Audio- (z.B. mp3) oder Bilddateien (z.B. jpg) oder für drahtlose Kommunikation [Coh89].
Die vorliegende Arbeit konzentriert sich auf die Untersuchung von Audiodateien mittels der Gabor-Transformation.
Dazu werden zunächst die Grundlagen der Fouriertransformation und der Gabor-Transformation erläutert. Im Kontext der Gabor-Transformation werden spezifische Konzepte wie das Gauß-Fenster, die Kurzzeit-Fouriertransformation (*STFT*), das Spektrogramm sowie die Inverse Gabor-Transformation und die Heisenbergsche Unschärferelation behandelt. Im zweiten Teil wird die Entwicklung eines Python-Projekts zur Anwendung der Gabor-Transformation auf *WAV*-Dateien einschließlich eines Vergleichs mit bereits vorhandenen Methoden beschrieben. Abschließend werden Musikdaten mithilfe eines implementierten Programms analysiert, darunter Synthesizer- und Keyboard-Sounds sowie Töne verschiedener Western- und E-Gitarren.
Diese Arbeit trägt dazu bei, das Verständnis für die Gabor-Transformation und ihre Anwendung in der Audiodatenanalyse zu vertiefen und bietet einen praktischen Einblick in die Umsetzung dieser Technik auf konkrete musikalische Signale.

2 Transformationen

2.1 Signal

Signale werden in der Mathematik mit Funktionen von einer oder mehreren unabhängigen Variablen der Zeit, des Raums usw. modelliert. Es gibt verschiedenste Signale, zum Beispiel Tonsignale [Gas99]. Bei Tonsignalen $f(t)$ wird t als Zeit interpretiert [Wer06]. Betrachtet man für ein Signal $f : X \to \mathbb{C}$, wobei X Banach- oder Hilbertraum ist, den Definitionsbereich X genauer, so erhält man verschiedene Arten von Signalen:

- zeitkontinuierlich: $X := \mathbb{R}$
- zeitdiskret: $X := \mathbb{Z}$ oder $X := \mathbb{N}$

- zeitbeschränkt: $X := [a, b], \quad -\infty \leq a \leq b \leq +\infty$

Zur Signalanalyse werden verschiedene Funktionenfamilien verwendet, zum Beispiel periodische oder nicht periodische Funktionen. Welchen Funktionstyp man für die Analyse eines Signals verwendet, hängt vom jeweiligen zugrundeliegenden Modell ab, das das Signal generiert. Für zeitbeschränkte Signale verwendet man etwa andere Funktionen als für zeitlich unbegrenzte Signale sowie für zeitkontinuierliche oder zeitdiskrete Signale [For10]. Unter diesen sind trigonometrische Signale die einfachsten periodischen Signale und gehen in die Struktur aller periodischen Signale ein [Gas99].
Gabor bezeichnet diese harmonischen Schwingungen, die genau eine Information bzw. eine bestimmte Frequenz enthalten, als elementare Signale. Reelle Signale $f(t) = a \cdot \cos(\omega t) + b \cdot \sin(\omega t)$ werden in komplexe Schreibweise $\psi(t) = f(t) + j\sigma(t) = (a - jb)\mathrm{e}^{j\omega t}$ umgewandelt [Gab46].
Eine Funktion f gilt als periodisch mit Periode a, $a > 0$, wenn für alle $t \in \mathbb{R}$ gilt $f(t + a) = f(t)$. Dabei ist a nicht unbedingt die kleinste Periode.
Weil $e_n(t) = \mathrm{e}^{2i\pi n\frac{t}{a}}$ die Periode a für jede Zahl n hat, gilt dies auch für Funktionen p der Form $p(t) = \sum_{n\in I} c_n \mathrm{e}^{2i\pi n\frac{t}{a}}$, mit I feste, endliche Menge von Zahlen und c_n beliebige komplexe Zahlen. Mit Hinzufügen von Nulltermen, falls nötig, kann man annehmen $p(t) = \sum_{n=-N}^{n=+N} c_n \mathrm{e}^{2i\pi n\frac{t}{a}}$. Diese Funktion nennt man ein trigonometrisches Polynom vom Grad kleiner oder gleich N. Diese Funktionen modellieren die Überlagerung einer endlichen Anzahl *monochromatischer* Signale, also solcher Signale, die nur eine Frequenz enthalten [Gas99].

Die Zeit-Frequenz-Analyse beschäftigt sich mit der Charakterisierung und Manipulation von Signalen, deren Frequenzkomponenten sich über die Zeit verändern.
Unter diesen Signalen unterscheidet man folgende Arten:

- T-periodische Signale: $f(t) = f(t + kT), \quad T \in \mathbb{R}^+$ und $k \in \mathbb{Z}$
- Signale endlicher Energie: $f \in L^2(\mathbb{R})$ oder $f \in l^2(\mathbb{Z})$
- begrenzte Signale: $f \in L^\infty(\mathbb{R})$ oder $f \in l^\infty(\mathbb{Z})$
- integrier- oder summierbare Signale: $f \in L^1(\mathbb{R})$ oder $f \in l^1(\mathbb{Z})$

Im Folgenden handelt es sich um zeitbeschränkte und diskrete Signale, die integrierbar sein müssen, um die Fouriertransformation definieren zu können [For10].

2.2 Fouriertransformation

Signale f werden zur besseren und schnelleren Interpretation in eine andere Form $\hat{f}$ gebracht bzw. transformiert. Mit der transformierten Funktion $\hat{f}$ können dann Störsignale oder ungewollte Geräusche entfernt oder herausgefiltert werden [For10].
Mithilfe der Fouriertransformation kann ein Signal, zum Beispiel ein Ton- oder Bildsignal in seine Frequenzanteile zerlegt werden [Coh89].

Definition 1 *Die Fouriertransformation ist definiert als*

$$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} f(t) \mathrm{e}^{-i\omega t} \mathrm{d}t. \tag{1}$$

Durch Anwendung der Fouriertransformation erhält man lediglich die Frequenzanteile von Signalen und keine lokale Information in Abhängigkeit der Zeit t. Ändert sich die Frequenz des Signals über die Zeit, so kann dies nicht festgestellt werden [Goe18], [Coh89], [Fei98].

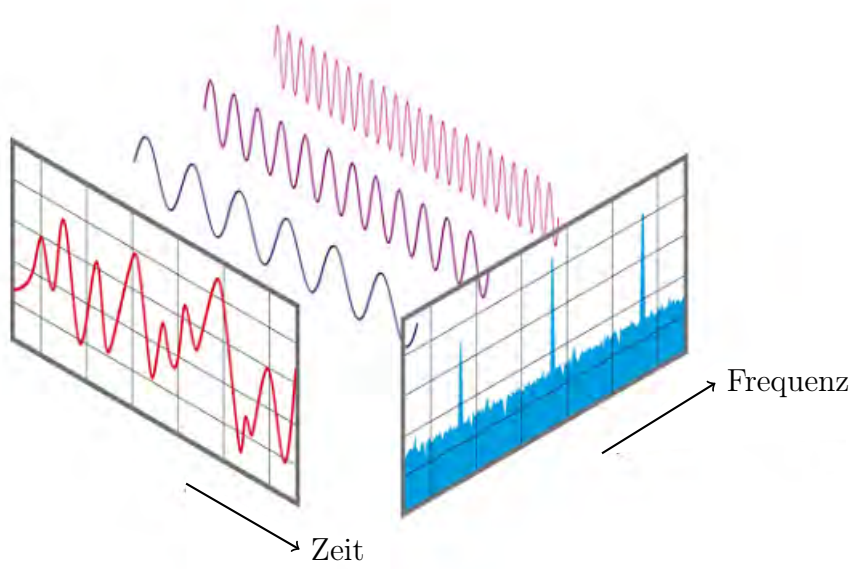


Abbildung 1: Zerlegung eines Signals in seine Frequenzanteile mit der Fouriertransformation[3]

Die Fouriertransformation ist gut geeignet, stationäre Signale und Prozesse zu untersuchen, die sich über die Zeit nicht ändern, etwa ein Dreiklang aus verschiedenen Tönen bzw. Frequenzen, bei dem die Töne nicht abklingen, siehe zum Beispiel Fouriertransformation eines monochromatischen Sinussignals in Abbildung 15a.

Man hat mit f und $\hat{f}$ zwei äquivalente Repräsentationen für dasselbe Objekt f, die dieselbe Information enthalten. Jede Darstellung macht bestimmte Eigenschaften von f sichtbar und zugänglich:

- f beschreibt das zeitliche Verhalten des Signals, zum Beispiel sind rhythmische Muster der Musik erkennbar. Die Melodie oder die Tonart sind schwer zu bestimmen.
- Mit $\hat{f}$ sind die enthaltenen Töne ersichtlich. Die Tonart ist wahrscheinlich erkennbar, aber weder der Rhythmus noch die Melodie.

Obwohl sowohl f als auch $\hat{f}$ alle möglichen Informationen enthalten, zeigt keine der beiden Darstellungen diejenigen Eigenschaften des Signals, die vom menschlichen Gehör wahrgenommen werden: Denn das Ohr erhält das Signal f und das Gehirn verarbeitet es in eine

[3] `https://commons.wikimedia.org/wiki/File:FFT-Time-Frequency-View.png`

Form, die gleichzeitig Information über Zeit und Frequenz bietet - sogenannte Musik.

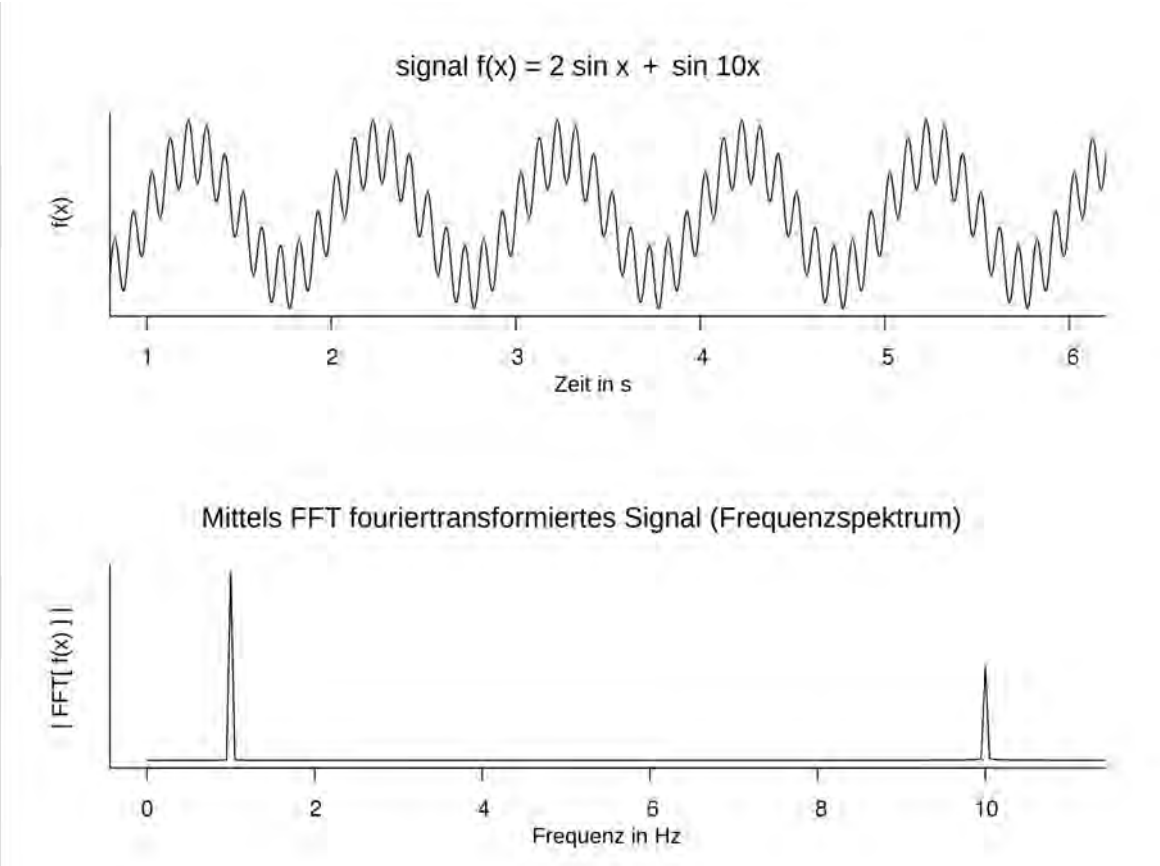


Abbildung 2: Signal f und zugehörige Fouriertransformation[4]: Bei der Fouriertransformation sieht man die enthaltenen Frequenzen, hier 1 *Hz* und 10 Hz. Beim Signal sieht man den zeitlichen Verlauf.

Die Fouriertransformation zeigt die dominierenden Frequenzen, allerdings geht das Verhalten von f in Abhängigkeit der Zeit verloren. Für kompliziertere Signale können weder f noch $\hat{f}$ die relevante Zeit-Frequenz-Information veranschaulichen [Grö01].

2.3 Gabor-Transformation

Um das Gehör zu imitieren und Darstellungen für das Signal zu finden, die sowohl Zeit als auch Frequenz repräsentieren, werden f und $\hat{f}$ kombiniert: Man benötigt alle Werte des Signals $f(x)$, um die Fouriertransformierte $\hat{f}(\omega)$ berechnen zu können. Musikalisch gesprochen muss man das gesamte Musikstück kennen, um die Häufigkeit des Auftretens einer bestimmten Frequenz zu bestimmen.
Außerdem hilft es wenig, nur die enthaltenen Frequenzen zu kennen, während der Zeitpunkt außer Acht gelassen wird, zu dem eine bestimmte Frequenz auftritt. Musikalisch gesagt bedeutet das, dass der Rhythmus, enthaltene Pausen, ruhigere Stellen oder schnelle

[4] `https://commons.wikimedia.org/w/index.php?lang=de&title=File%3ASimple_time_domain_vs_frequency_domain.svg&uselang=de`

Teile des Musikstücks unbekannt bleiben. Man kennt nur die vorkommenden Töne.
Denn sowohl f als auch $\hat{f}$ sind Funktionen mit einer Variable, während eine Zeit-Frequenz-Darstellung von $f(x, \omega)$ das Auftreten einer Frequenz ω zu einer bestimmten Zeit x angibt. So kann man die Änderung von Signalen über die Zeit sehen, zum Beispiel das Abklingen eines Gitarrentons oder eine Melodie mit mehreren Tönen. Eine ideale Darstellung würde somit direkte Informationen über die Frequenzen ω bereitstellen, die zu einer beliebigen Zeit x auftreten, siehe Kapitel 2.3.6.
In der Praxis löst man das Problem mithilfe einer symbolischen Zeit-Frequenz-Darstellung: der Partitur. Diese liefert für jedes Instrument eine solche Darstellung, mit der horizontal die Zeit und vertikal die zu spielenden Töne abgebildet sind.

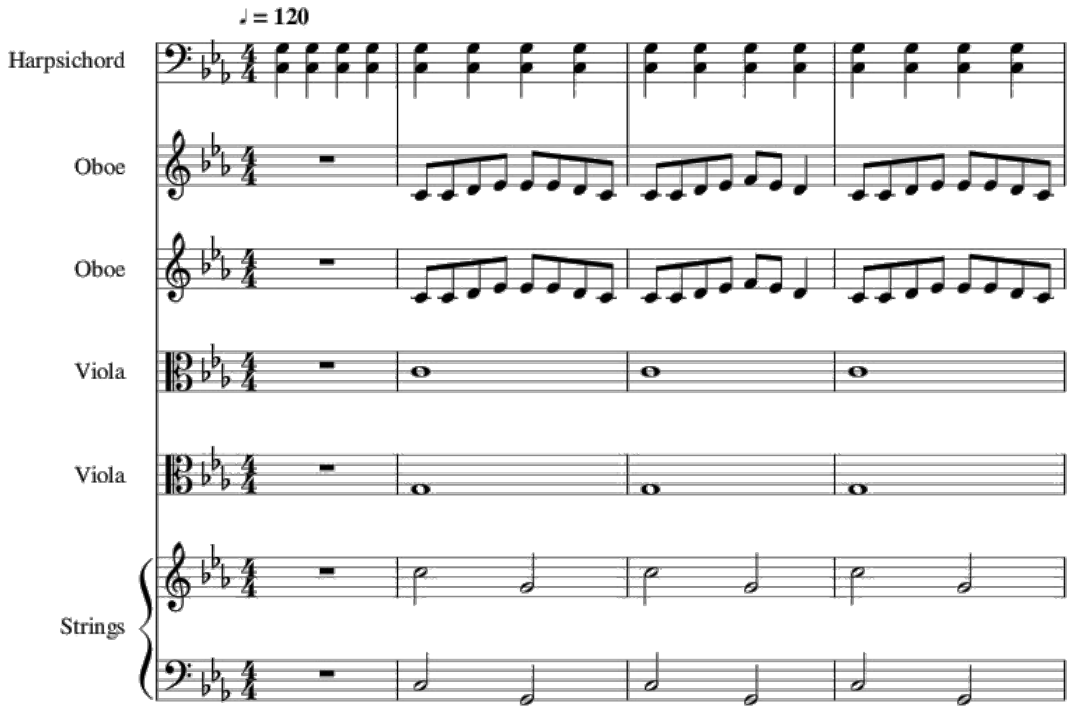


Abbildung 3: Ausschnitt einer Partitur[5]: Pro Notenzeile sieht man im Verlauf der Zeit (horizontal) die zu spielenden Töne (vertikal). Eine Partitur ist insofern komplex als dass sie in jeder Zeile, d.h. für jedes Instrument eine Art Zeit-Frequenz-Darstellung bietet.

Die globale Fouriertransformation eines Langzeitsignals ist von geringem praktischen Wert für die Analyse des Frequenzspektrums des Signals. Aus $\hat{f}$ kann man beispielsweise schlecht hochfrequente Ausschläge ablesen. Signale, die sich über die Zeit auf unvorhergesehene Art ändern, zum Beispiel ein Sprachsignal oder ein EEG Signal, benötigen Frequenzanalysen, die das Signal in Abhängigkeit der Zeit betrachten. Ein Versuch für eine solche Zeit-Frequenz-Analyse war, das Signal erst in Stücke zu schneiden, dann eine Fourier-Analyse auf diesen Stücken durchzuführen. Die erhaltenen Ergebnisse können allerdings an den Rändern dieser Stücke verfälscht sein, weil die Fouriertransformation

[5] `https://commons.wikimedia.org/wiki/File:MuseScore_3.jpg`

die Sprünge an den Grenzen als Diskontinuität oder eine abrupte Änderung des Signals interpretiert.
Um das zu verhindern, versuchte man das Untersuchen mittels einer Fensterfunktion, sogenanntes „Windowing“. Um keine harten Kanten zwischen den aufgeteilten Stücken zu erhalten, kann man eine Fenster-Funktion verwenden. Diese Art der Zeit-Frequenz-Analyse nennt man (kontinuierliche) Kurzzeit-Fouriertransformation(STFT) oder gefensterte Fouriertransformation [Grö01], [Fei98].
Mathematisch schreibt man für eine beliebige Funktion $f \in L^2(\mathbb{R})$ mit einem Fenster g [Fei98]

$$\mathcal{V}_g f(t,\omega) = \int_{\mathbb{R}} f(s)\overline{g(s-t)}\mathrm{e}^{-2\pi i\omega s}\mathrm{d}s. \tag{2}$$

2.3.1 Gauß-Fenster

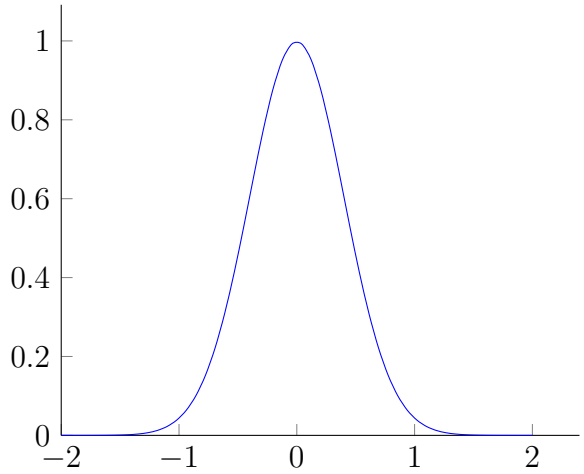


Abbildung 4: Beispiel für ein Gauß-Fenster

Die von Gabor als Fenster verwendete Funktion ist von der Form $g(t) := A\mathrm{e}^{-\alpha t^2}$. In der Literatur werden verschiedene Gauß-Fenster beschrieben, darunter $g(t) := \pi^{-\frac{1}{4}}\mathrm{e}^{-\frac{t^2}{2}}$ [Gas99], $g_s(t) := \frac{1}{2\sqrt{\pi s}}\mathrm{e}^{\frac{-t^2}{4s}}$ mit Parameter $s > 0$ [For10] oder $g(t) = \frac{1}{\sqrt{2\pi\sigma_t^2}}\mathrm{e}^{-\frac{1}{2}\frac{t^2}{\sigma_t^2}}$ [vdB05].
Beliebte Fenster sind neben Gauß-Fenster z.B. Hamming-, Hann- oder Kaiser-Fenster. Sie sind alle symmetrisch und nahe bei 1 um den Ursprung und gehen an den Grenzen gegen 0 [Fei98], [For10], [Gas99].

2.3.2 Kurzzeit-Fouriertransformation

Bei einer gefensterten Fouriertransformation, auch Kurzzeit-Fouriertransformation (engl. short-time Fourier transform, kurz *STFT*), wird das Signal nicht wie bei der diskreten Fouriertransformation in viele kleine Signale zerteilt, auf denen die Fouriertransformation durchgeführt wird, sondern mithilfe einer Fensterfunktion g über die Signalfunktion gegangen. So überlappen sich die betrachteten Teile und man erhält zu den jeweiligen Frequenzen den Zeitpunkt, zu dem diese im Signal auftreten [Gab46].
Mit einer Fensterfunktion φ ist $\Phi f(\tau,\omega)$ eine solche gefensterte Fouriertransformation.

Definition 2 *Seien $\varphi \in L^1(\mathbb{R}) \cap L^2(\mathbb{R})$ und $f \in L^2(\mathbb{R})$. Für $\omega \in \mathbb{R}$ und $b \in \mathbb{R}$ sei*

$$\Phi f(\tau, \omega) := \int_{\mathbb{R}} f(t)\varphi(t-\tau)\mathrm{e}^{-i\omega t}\mathrm{d}t \tag{3}$$

die STFT [For10]. Die Kurzzeit-Fouriertransformation ist ein Standardwerkzeug in der Signalanalyse, aber sie hat auch Nachteile: die begrenzte Kapazität der Zeit-Frequenz-Auflösung (vgl. Abschnitt 2.3.6) [Fei98].

2.3.3 Gabor-Transformation

Dennis Gabor führte 1946 in „Theory of Communication“ [Gab46] seine Methode ein, um ein eindimensionales Signal in zwei Dimensionen, Zeit und Frequenz, darzustellen. Gabor wollte die Informationen eines Signals mit unzähligen Funktionswerten $f(t)$ mit einer endlichen Menge an Daten lokal so gut wie möglich repräsentieren. Die Gabor-Transformation ist also eine gesamplete STFT [Fei98].
Gabor verwendete ein Gauß-Fenster (vgl. 2.3.1) in der STFT.

Definition 3 *Seien $f \in L^2(\mathbb{R})$ und seien $s > 0$ und $b \in \mathbb{R}$. Dann sei*

$$Gf(\tau, \omega) := \int_{\mathbb{R}} f(t)g(t-\tau)\mathrm{e}^{-i\omega t}\mathrm{d}t \tag{4}$$

die Gabor-Transformation mit Fensterfunktion g.

Die Gabor-Transformation ist eine sogenannte Zeit-Frequenz-Repräsentation, in der die Parameter τ und ω die Zeitvariable und die Frequenzvariable darstellen [Fei98], [vdB05].

2.3.4 Spektrogramm

Das Spektrogramm ist das Ergebnis einer Gabor-Transformation, ein Zeit-Frequenz-Diagramm, das angibt, wie viel Energie zu welchem Zeitpunkt sich in einer Frequenz „befindet“ [Coh89].

Definition 4 *Sei $g \in L^2(\mathbb{R}^d)$ eine Fensterfunktion, sodass $\|g\|_2 = 1$. Dann ist das Spektrogramm von f d definiert als $|\mathcal{V}_g f(t, \omega)|^2$.*

Das Spektrogramm kann man interpretieren als Energieintensität über der Zeit-Frequenz-Ebene [Grö01], [Fei98].
Ein Spektrogramm sieht man in Abbildung 5. Dieses Spektrogramm stellt ein 20-sekündiges Signal dar, bei dem die Frequenz, also die Höhe des Tons im Signal alle 5 s erhöht wird. Zuerst liegt diese bei 10 Hz, gefolgt von 30 Hz, 50 Hz und 100 Hz.

2.3.5 Inverse Gabor-Transformation

Die Gabor-Transformation ist invertierbar, d.h. man kann die Ausgangsfunktion f aus der Gabor-Transformation erhalten[For10], [Fei98], [Gas99], [Grö01].

[6] `https://commons.wikimedia.org/wiki/File:STFT_colored_spectrogram_375ms-de.png`

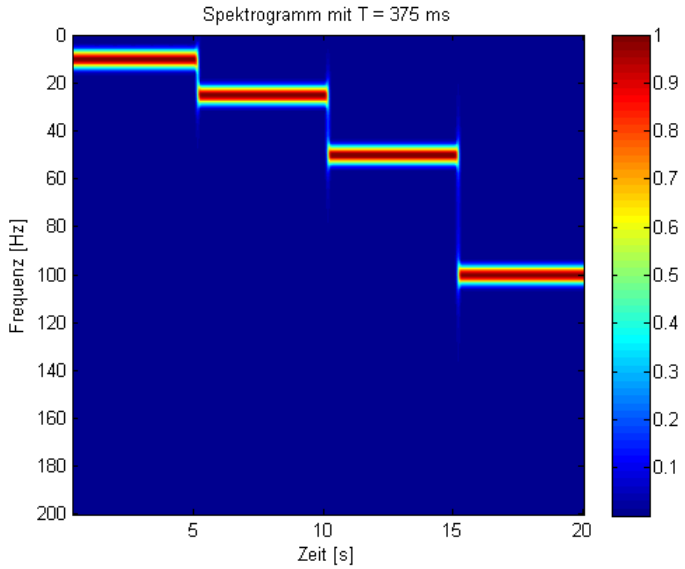


Abbildung 5: Beispiel für ein Spektrogramm[6] mit vier verschiedenen monochromatischen Tönen

Satz 1 *Inverse Gabor-Transformation. Sei* $f \in L^1(\mathbb{R}) \cap L^2(\mathbb{R})$. *Für alle* $t \in \mathbb{R}$, *für die* f *kontinuierlich ist, gilt:*

$$f(t) = \frac{1}{2\pi} \int_{\mathbb{R}} \int_{\mathbb{R}} Gf(\tau, \omega) g(t-\tau) \mathrm{e}^{i\omega t} \mathrm{d}\omega \, \mathrm{d}\tau. \tag{5}$$

2.3.6 Heisenbergsche Unschärferelation

Weder eine Partitur noch ein mathematisches Modell können eine ideale Zeit-Frequenz-Analyse darstellen. Es ist nicht möglich, zu jedem exakten Zeitpunkt genau die auftretenden Frequenzen zu bestimmen [For10]. Denn kleine Frequenzen können nur schwierig mit kurzen Fenstern abgebildet werden, während nur kurz auftretende Impulse mit langen Fenstern schlecht in der Zeit lokalisiert werden können.
Diese Tatsache beschreibt die Heisenbergsche Unschärferelation [Fei98].

Satz 2 *Seien* $(t_0, \omega_0) \in \mathbb{R} \times \hat{\mathbb{R}}$. *Dann gilt* $\forall f \in L^2(\mathbb{R})$,

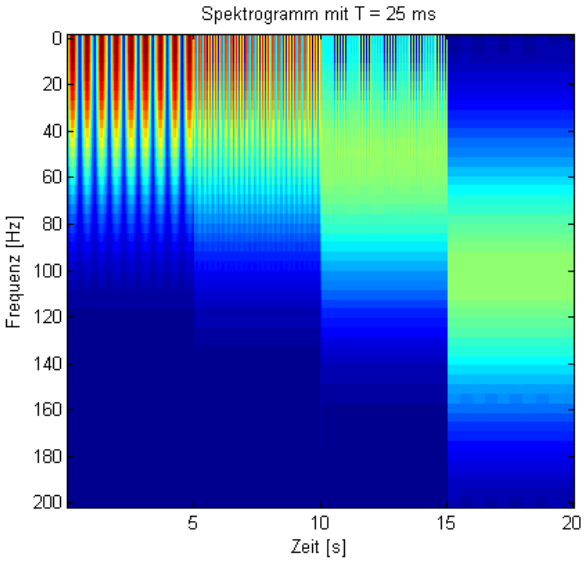
$$\|f\|_2^2 \leq 4\pi \|(t-t_0) f(t)\|_2 \|(\omega - \omega_0) \hat{f}(\omega)\|_2 \tag{6}$$

Gleichheit kann nur erreicht werden mit Funktionen der Form

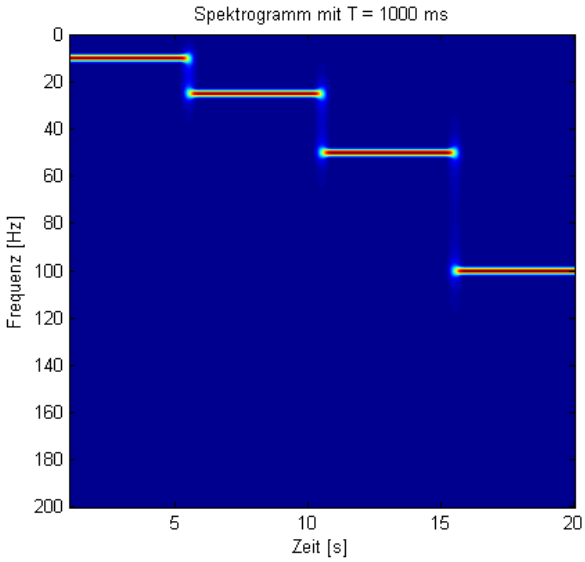
$$f(t) = C \mathrm{e}^{2\pi i t \omega_0} \mathrm{e}^{-s(t-t_0)^2} \quad , \qquad C \in \mathbb{C}, s > 0. \tag{7}$$

Trotz dieser Unschärferelation ist eine Zeit-Frequenz-Analyse nützlich, wenn man die Fenstergröße passend wählt [Grö01], [Fei98].

[6] `https://commons.wikimedia.org/wiki/File:STFT_colored_spectrogram_25ms-de.png`
`https://commons.wikimedia.org/wiki/File:STFT_colored_spectrogram_1000ms-de.png`



(a) Spektrogramm erstellt mit kurzem Fenster: Die kleinen Frequenzen am Anfang werden nur sehr schlecht abgebildet und selbst bei 15 s - 20 s ist die Frequenz mit 70 Hz bis 140 Hz sehr ungenau abzulesen. Dafür ist eine gute Zeitlokalisierung möglich mit klarer Abtrennung z.B. bei 15 s.



(b) Spektrogramm erstellt mit langem Fenster: Die Frequenzen können gut lokalisiert werden, z. B. Frequenz bei 15 s - 20 s liegt bei 100 Hz, während die Grenzen zwischen den Tönen hier nicht mehr so klar abgebildet sind. Zum Zeitpunkt 15 s ist ungenau, welche der beiden Frequenzen auftritt.

Abbildung 6: Vergleich der für das gleiche Signal wie in Abbildung 5 mit unterschiedlich großen Fenstern erstellten Spektrogramme[7]

3 Implementierung der Gabor-Transformation in Python

Das Python-Projekt inklusive aufgenommener Audiodateien sowie Anweisungen zur Installation ist auf GitHub[8] zu finden. Es befasst sich mit dem Einlesen von Audiodateien im WAV-Format, dem Transformieren dieser Dateien und dem anschließenden Generieren eines Plots der Datei, der Fouriertransformation und der Gabor-Transformation.

3.1 Hardware

Für die Ausführung der bereits vordefinierten Methoden mit den Klassen `NpFourier` und `NpGabor` ist keine besondere Rechenleistung erforderlich, , sondern ein handelsüblicher PC genügt. Im Speziellen war dies ein MacBook Air mit 1,8 GHz Dual-Core Intel Core i5 Prozessor, 8 GB 1600 MHz DDR3 RAM und Intel HD Graphics 6000 1536 MB Grafikkarte. Die eigenen Methoden zur Fouriertransformation (Klassen `OwnFourier` sowie `OwnGabor`) benötigen bei der Ausführung mit diesem Rechner allerdings zu lange und wurden mit einem PC mit Microsoft Windows 11, AMD Ryzen 7 3700X 8-Kern Prozessor, 16 GB RAM und NVIDIA GeForce RTX 3060 Ti Grafikkarte getestet.

[8] `https://github.com/Chris-tina-P/gabor`

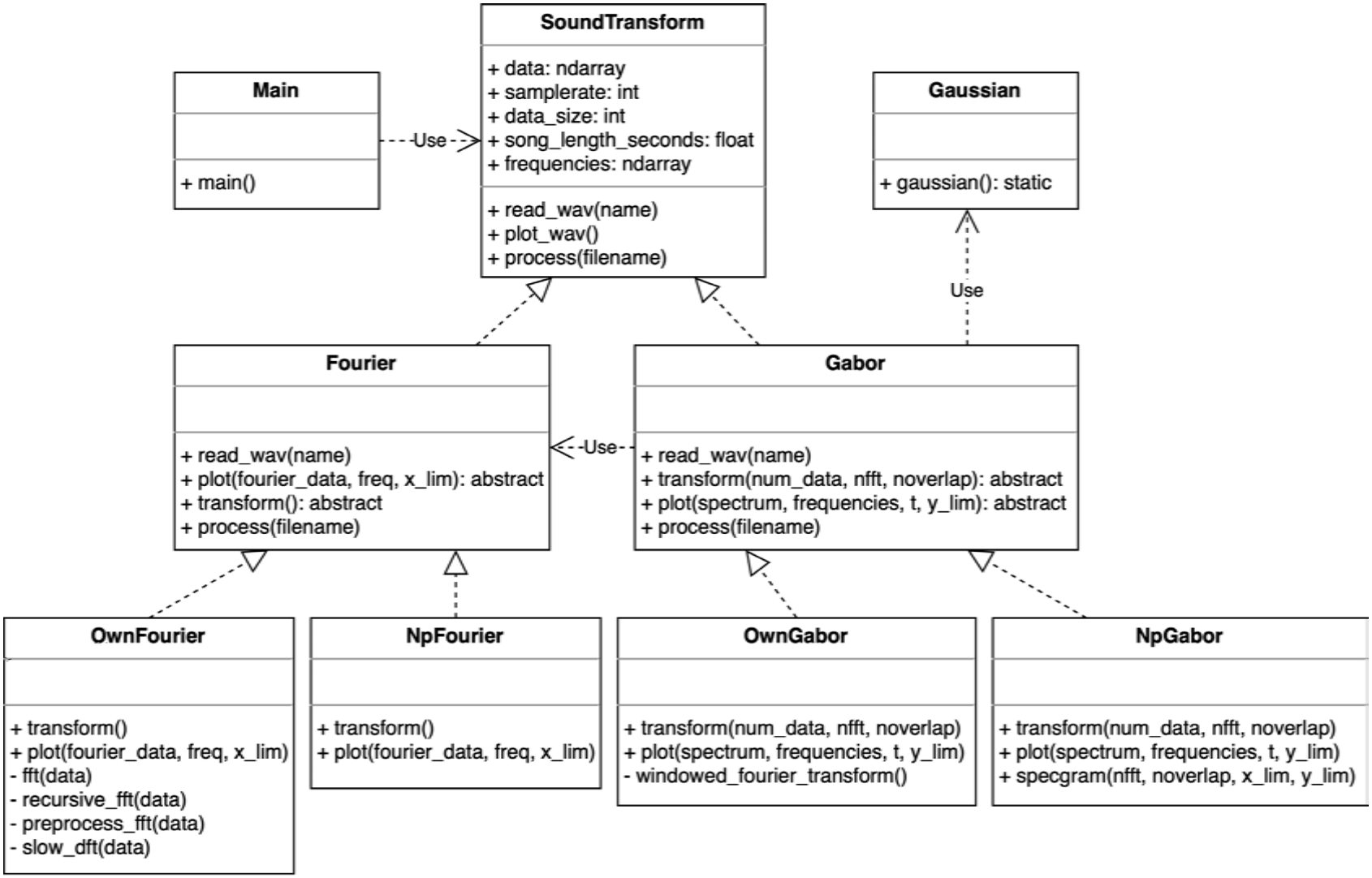


Abbildung 7: Klassendiagramm zum implementierten Python-Projekt

3.2 Implementiertes Programm

Das Klassendiagramm für das entstandene Programm ist in Abbildung 7.
Mithilfe der `main()`-Methode werden die Methoden der Klasse `SoundTransform` aufgerufen:

- Mit `read_wav(name)` wird eine WAV-Datei eingelesen.
- Die eingelesenen Daten können mit `plot_wav()` grafisch dargestellt werden.
- `process(filename)` erledigt bereits das Einlesen der Datei, transformiert die Daten und stellt die Ergebnisse in einem entsprechenden Plot dar.
  Dazu werden jeweils `read_wav()`, `transform()` und `plot()` der Unterklassen (mit entsprechenden Parametern) aufgerufen.
  Je nachdem, ob man `process(filename)` mit einem Fourier- oder Gabor-Objekt aufruft, wird dann eine Fourier- oder Gabor-Transformation durchgeführt. Außerdem kann man jeweils eigens geschriebene Methoden der Klassen `OwnFourier` und `OwnGabor` verwenden lassen oder die bereits vordefinierten Methoden für die FFT[9] in `NpFourier` und für das Spektrogramm[10] in `NpGabor` aus dem Package `numpy` bzw. `matplotlib`.
  Zur Berechnung der Gabor-Transformation mit vordefinierten Methoden gibt es mehrere Möglichkeiten: Mit `specgram()` wird direkt ein Spektrogramm erstellt, während mit `process(name)` die Methoden `transform()` und `plot()` aufgerufen werden, wobei auf beide Arten dasselbe Ergebnis erzielt wird.

Für die Berechnung der Fouriertransformation werden die vordefinierte oder die selbst implementierte schnelle Fouriertransformation (engl. Fast Fourier Transform, kurz *FFT*) verwendet, wobei die eigene FFT den *Cooley-Tukey-Algorithmus* nutzt [Gas99].
Um ein Spektrogramm zu erhalten, werden nach dem Einlesen der Daten mit `read_wav(name)` die Methoden `transform()` (Listing 1) und `plot(spectrum, frequencies, t, y_lim)` in `OwnGabor` (Listing 3) aufgerufen.

Listing 1: Teil der `transform`-Methode in der Klasse `OwnGabor` für die eigene Berechnung der Gabor-Transformation bestehend aus vielen FFT. Mit den Rückgabewerten kann ein Spektrogramm erstellt werden (siehe Listing 3)

```
def transform(self, num_data: int = 5001, nfft: int = 10000, noverlap: int
    = 500) -> (
        List[ndarray], ndarray, ndarray):
    """
    This method computes several fourier transforms of the loaded sound file
    :param num_data: The number of data values to be averaged
    :param nfft: The number of data points used in each block for the FFT
    :param noverlap: The number of points of overlap between blocks
    :return: spectrum, frequencies and time points for plotting a spectrogram
    """
```

[9] https://numpy.org/doc/stable/reference/generated/numpy.fft.rfft.html
[10] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.specgram.html

```
        fft_to_use = self.fourier

        spectrum = []
        Fs = self.samplerate
        div = len(self.frequencies) // num_data
        upper_limit = int((self.data_size - nfft / 2 + 1 + nfft - noverlap) //
            (nfft - noverlap))

        # Calculate the spectrum
        t = np.linspace(0, self.song_length_seconds, self.data_size)
        for i in range(0, upper_limit):
            window = Gaussian.gaussian(t, i * (nfft - noverlap) / Fs, 0.1)

            gaussian_filtered = self.data * window

            fourier_data = abs(fft_to_use(gaussian_filtered))
```

`OwnGabor.transform()` berechnet die gefensterten Fouriertransformationen unter Zuhilfenahme des in `OwnFourier` implementierten Algorithmus für die FFT (Listing 2). Dazu ruft `OwnGabor.transform()` die Methode `OwnFourier.fft(data)` auf, welche die Daten an `preprocess_fft(data)` und die dann erhaltenen präparierten Daten an `recursive_fft()` weitergibt. `preprocess_fft(data)` hängt lediglich fehlende Nullen an die Daten an. `recursive_fft(data)` teilt die Daten bis zu einer bestimmten Größe auf und ruft sich solange selbst rekursiv auf, bis eine kleine Größe erreicht ist, auf der dann mit `slow_dft(data)` eine diskrete Fouriertransformation berechnet wird.

Listing 2: Methoden in der Klasse `OwnFourier` für die eigene Implementierung der FFT

```
    def _slow_dft(self, data) -> ndarray:
        """
        Compute the discrete Fourier Transform of the 1D array data
        :param data: The data to transform
        :return: The transformed data
        """
        x = np.asarray(data, dtype=float)
        N = x.shape[0]
        n = np.arange(N)
        k = n.reshape((N, 1))
        M = np.exp(-2j * np.pi * k * n / N)
        return np.dot(M, x)

    def _preprocess_fft(self, data) -> ndarray:
        """
        This method preprocesses the data for the FFT by adding zeros to the end
            of the data array.
        :param data: The data to preprocess
        :return: The preprocessed data ready for the FFT
        """
```

```
        logarithm = int(math.log(len(data), 2))
        next_power_of_two = 2 ** (logarithm + 1)

        if len(data) < next_power_of_two:
            data = np.append(data, np.zeros(next_power_of_two - len(data)))

        return data

    def _recursive_fft(self, data) -> ndarray:
        """
        This method computes the FFT of the data recursively.
        :param data: The data to transform
        :return: The transformed data
        """
        x = np.asarray(data, dtype=float)
        n = x.shape[0]

        cutoff = 32 # cutoff can be optimized
        if n <= cutoff:
            return self._slow_dft(x)
        else:
            x_even = self._recursive_fft(x[::2])
            x_odd = self._recursive_fft(x[1::2])
            factor = np.exp(-2j * np.pi * np.arange(n) / n)
            return np.concatenate([x_even + factor[:n // 2] * x_odd, x_even +
                factor[n // 2:] * x_odd])

    def fft(self, data) -> ndarray:
        """
        A recursive implementation of the 1D Cooley-Tukey FFT
        :param data: The data to transform
        :return: The transformed data
        """
        x = self._preprocess_fft(data)
        processed = self._recursive_fft(x)
        return processed
```

Mit den Rückgabewerten aus `transform()` wird anschließend `plot(spectrum, frequencies, t, y_lim)` aufgerufen, um ein Spektrogramm zu erhalten.

Listing 3: `plot`-Methode in der Klasse `OwnGabor` für die eigene Erstellung eines Spektrogramms

```
    def plot(self, spectrum: ndarray, frequencies: ndarray, t: ndarray, y_lim:
        int = 1000) -> None:
        """
        This method plots the spectrogram of the sound file.
        :param spectrum: color spectrum
```

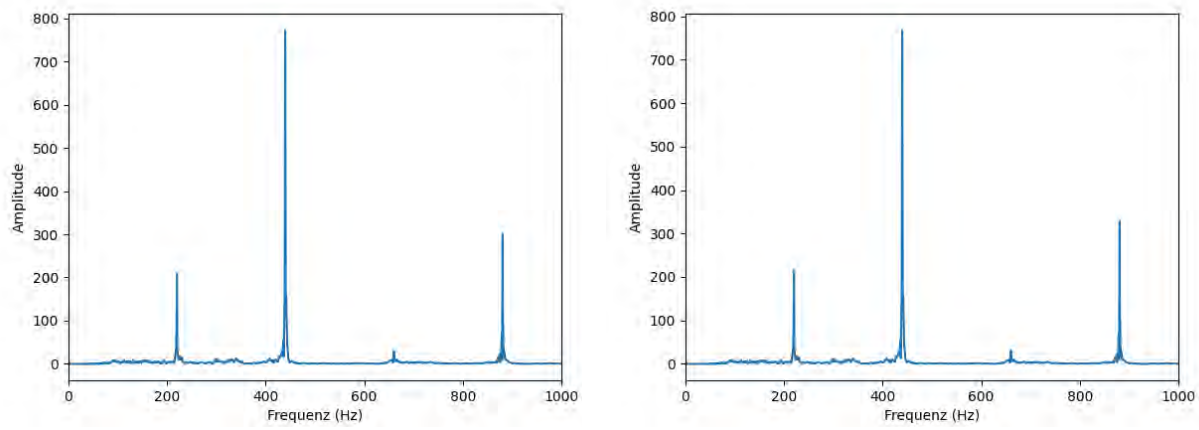
```
        :param frequencies: y-axis
        :param t: x-axis
        :param y_lim: limit for the y-axis
        :return: None
        """
        # shading='gouraud' makes diagram smoother
        plt.pcolormesh(t, frequencies, spectrum, shading='nearest',
            cmap='jet_r', norm='log')
        plt.ylim([0, y_lim])
        plt.ylabel('Frequenz (Hz)')
        plt.xlabel('Zeit (s)')
        plt.show()
```

3.3 Vergleich der eigenen und vordefinierten Methoden

Zum Vergleich der selbst implementierten und vordefinierten Methoden wurden mehrere WAV-Dateien auf verschiedene Arten getestet.

- Fouriertransformation
  - `OwnFourier.process()`
  - `NpFourier.process()`
- Gabor-Transformation
  - `OwnGabor.process()` mit eigener FFT aus `OwnFourier`
  - `OwnGabor.process()` mit vordefinierter `numpy` FFT aus `NpFourier`
  - `NpGabor.process()` mit `specgram()` aus `matplotlib`, aber eigenem Plot mit `pcolormesh()`
  - `NpGabor.specgram()` mit `specgram()` aus `matplotlib`

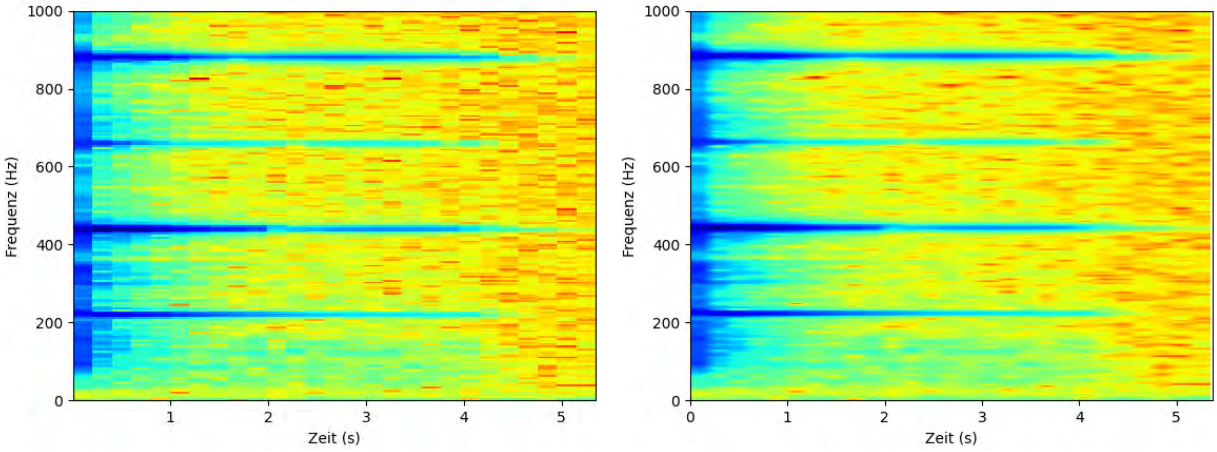
Die eigene Implementierung der FFT ist langsam und deshalb funktioniert die eigene Gabor-Transformation mit eigener FFT nicht mit jedem Computer (siehe Kapitel 3.1) Hier dauert die Berechnung mit den eigenen Methoden zwar länger als die vordefinierten Methoden, kommt aber nach ein paar Minuten zum Ergebnis.
Der in der Datei `Klavier_A_leicht.wav` aufgenommene Klavierton hat eine Dauer von ca. 5 s, *Sampling Rate* 48000 Hz und enthält 257879 *Samples*.
Für die FFT ergibt sich fast kein Unterschied, die erhaltenen Frequenzen sind dieselben. Bei der Höhe der Ausschläge sind kleine Unterschiede, zum Beispiel bei 880 Hz. Größter Unterschied ist die Berechnungsdauer: Bei `OwnFourier` erhält man den Plot nach ca. 1 s, während das Berechnen bei `NpFourier` nur 0,2 s dauert. Die eigene FFT dauert sogar bei dieser nur kleinen Datei um das Fünffache länger, könnte also sicher noch optimiert werden. Zum Beispiel werden sehr viele Nullen angehängt, um die passende Datengröße in Form einer Zweierpotenz zu erhalten.
Mit der vordefinierten Methode `specgram()` im Package `numpy` ergeben sich zwei Möglichkeiten: Mit der Übergabe der Rückgabewerte der Methode `specgram()` an `pcolormesh`



(a) FFT mit `OwnFourier.process()` **(b)** FFT mit `NpFourier.process()`

Abbildung 8: Vergleich von `OwnFourier` und `NpFourier`: kleinste Unterschiede in Höhe der Ausschläge, z.B. bei 880 Hz

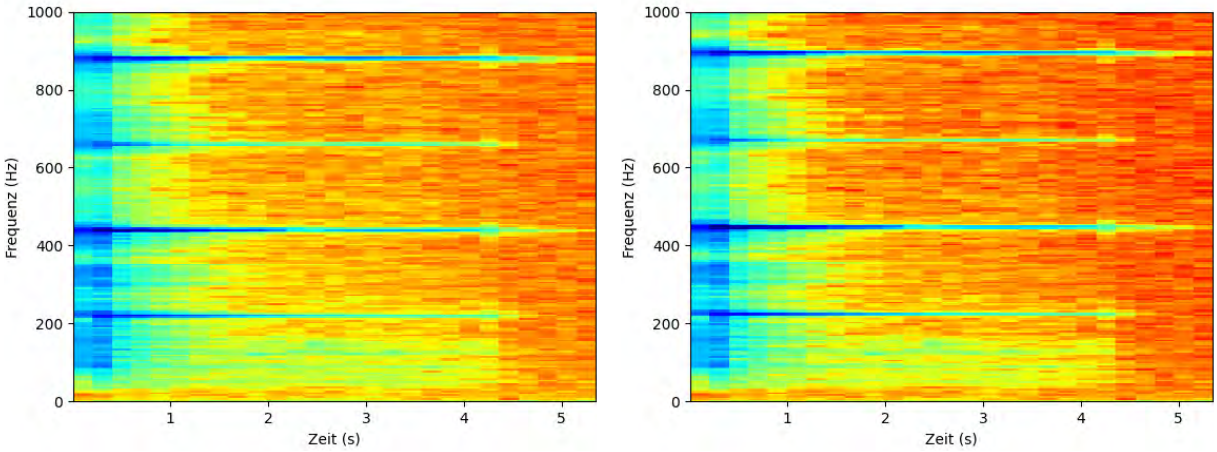
(Abbildung 9a) oder direkt mit `specgram()` kann ein Diagramm erstellt werden (Abbildung 9b). Der Unterschied in der Abbildung 9 besteht nur in der Darstellung der Rechtecke bzw. dem Verschwimmen der Grenzen dieser. Mit dem zusätzlichen Attribut `shading=gouraud` bei `pcolormesh()` stimmten die beiden Plots in Abbildung 9 exakt überein.



(a) `NpGabor` mit `pcolormesh()` **(b)** `NpGabor` mit der in `matplotlib` vordefinierten `specgram()`-Methode

Abbildung 9: Mit den vordefinierten `numpy`-Methoden mit `NpGabor` erhaltene Spektrogramme

Die Methoden der eigenen Gabor-Transformation benötigen sogar für die kurze Datei schon relativ lange. Während sich die Berechnung mit der vordefinierten FFT mit 2 s noch in Grenzen hält, dauert es 24 s, bis man mit der komplett eigenen Methode zur Gabor-Transformation mit der eigenen FFT den Plot in Abbildung 10b erhält. Mit dem in Kapitel 3.1 erwähnten MacBook konnte sogar nicht einmal ein Ergebnis erzielt werden, sondern es musste der ebenfalls in Kapitel 3.1 genannte Windows-PC verwendet werden.



(a) Eigene Gabor-Transformation mit `numpy` FFT

(b) Eigene Gabor-Transformation mit eigener FFT

Abbildung 10: Vergleich der eigenen Gabor-Transformationen mit `numpy` und eigener FFT: kleine Unterschiede im Plot, aber große Unterschiede in der Berechnungsdauer

Ein weiteres Problem bei der eigenen Gabor-Transformation ist das verwendete Gauß-Fenster, dessen Werte an den Grenzen nie wirklich 0 werden, sondern nur näher gegen 0 gehen. Um nicht unnötigerweise alle Werte zu berechnen, könnte man das Gauß-Fenster zuschneiden. Dann werden bei der Gabor-Transformation mit einem Aufruf der FFT nicht alle Werte der Datei berücksichtigt. So könnte die Berechnung verkleinert und damit schneller werden. Die verschiedenen Methoden wurden für mehrere WAV-Dateien verschiedener Längen getestet, exemplarisch sind hier die erhaltenen Grafiken nur für den Klavierton, da sich für den Gitarrenton und den Anfang der Melodie zu „Happy Birthday“ dieselben Unterschiede ergeben.

Mit den verschiedenen Methoden ergeben sich sehr unterschiedliche Berechnungszeiten: Am schnellsten sind die vordefinierten Methoden mit Werten unter einer Sekunde, während die eigens implementierten Methoden deutlich länger brauchen. Für die eigens implementierte Fouriertransformation erhält man zumindest innerhalb von Sekunden ein Ergebnis, wohingegen die eigene Gabor-Transformation durch die vielen Aufrufe und jeweils große Berechnung der FFT im Falle der 25 s langen WAV-Datei „Happy Birthday“ mehr als eine Viertelstunde für die Berechnung benötigt. Mit dieser Dauer ist die selbst implementierte Gabor-Transformation zu ineffizient und praktisch nicht sinnvoll nutzbar. Die eigene Implementierung erfüllt aber ihren Zweck: die Veranschaulichung einer prinzipiellen Implementierung der Gabor-Transformation - ohne Optimierung und nur mit einfacher Berechnung.

	Klavier	Gitarre	Happy Birthday
Länge Aufnahme	5,4 s	14,3 s	25,4 s
OwnFourier	1 s	3,8 s	7,6 s
NpFourier	0,2 s	0,3 s	0,3 s
NpGabor pcolormesh	0,3 s	0,3 s	0,4 s
NpGabor specgram	0,2 s	0,2 s	0,2 s
OwnGabor np FFT	2 s	12 s	37 s
OwnGabor own FFT	24 s	$>$ 4 min	$>$ 15 min

Tabelle 1: Dauer der Berechnung der verschiedenen Arten mit Windows-PC aus Kapitel 3.1

4 Gabor-Transformation für Musikdaten

4.1 Töne und Frequenzen in der westlichen Musik

In der westlichen, hier betrachteten Musik wird eine Oktave in 12 Töne unterteilt. Ein um eine Oktave höherer bzw. tieferer Ton bedeutet eine Verdopplung bzw. Halbierung der Frequenz. Ausgehend vom sogenannten Kammerton a' mit der Frequenz 440 Hz werden so alle anderen Töne be- und gestimmt wie in Tabelle 2.

'A	55	A	110	a	220	a'	440	a"	880
'Ais	58,3	Ais	116,5	ais	233,1	ais'	466,2	ais"	932,3
'B	61,7	B	123,5	b	246,9	b'	493,9	h"	987,8
C	65,4	c	130,8	c'	261,6	c"	523,3	c"'	1046,5
Cis	69,3	cis	138.6	cis'	277,2	cis"	554,4	cis"'	1108,7
D	73,4	d	146,8	d'	293,7	d"	587,3	d"'	1174,7
Dis	77,8	dis	155,6	dis'	311,1	dis"	622,3	dis"'	1244,5
E	82,4	e	164,8	e'	329,6	e"	659,3	e"'	1318,5
F	87,3	f	174,6	f'	349,2	f"	698,5	f"'	1396,9
Fis	92,5	fis	185,0	fis'	370,0	fis"	740,0	fis"'	1480,0
G	98,0	g	196,0	g'	392,0	g"	784,0	g"'	1568,0
Gis	103,8	gis	207,7	gis'	415,3	gis"	830,6	gis"'	1661,2

Tabelle 2: Töne mit zugehöriger Frequenz, gerundet

Je nach Modell der Gitarre hat diese unterschiedlich viele Bünde und damit mehr oder weniger Töne. Die untersuchten Gitarrensaiten oder im 12. Bund erzeugten Töne sind in der Praxis auf nahezu jeder Gitarre vorhanden.

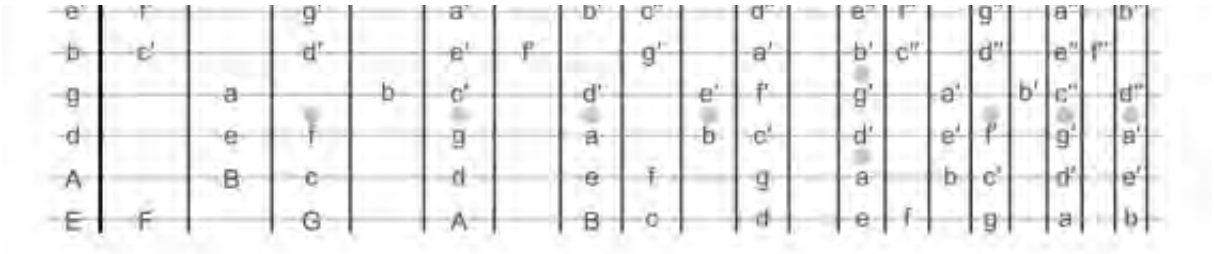


Abbildung 11: Gitarrengriffbrett[11] mit Noten ergänzend zur Tabelle 2 und zur Klaviatur in Abbildung 12

Mit 88 Tasten haben Klaviere 88 verschiedene Töne und umfassen damit einen großen Bereich. Im Gegensatz zur Gitarre (siehe Abbildung 11) kommen keine Töne mehrfach vor. Töne aller Instrumente setzen sich immer aus mehreren verschiedenen Sinusschwin-

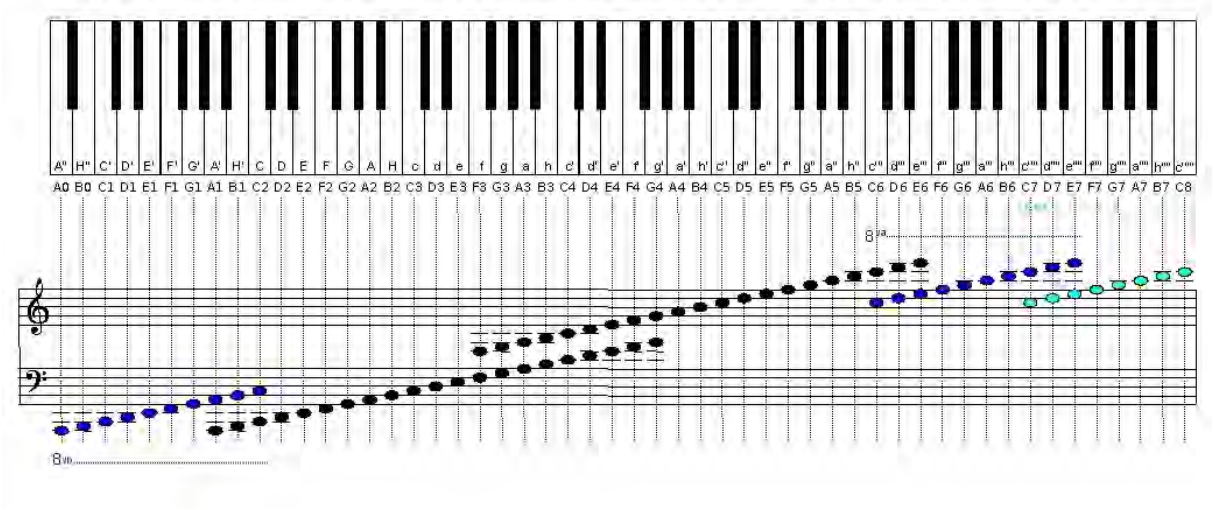
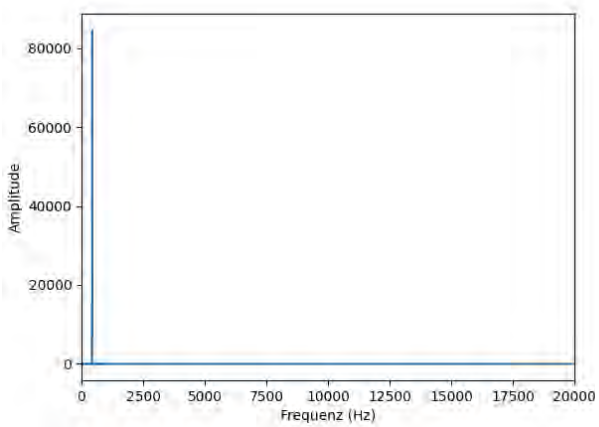


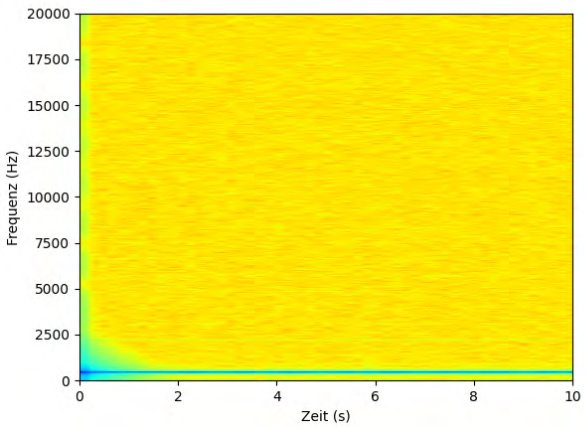
Abbildung 12: Klaviatur mit eingezeichneten Noten auf der Klaviatur und in der Notenzeile[12], ergänzend zur Tabelle 2 und zum Gitarrengriffbrett in Abbildung 11

gungen zusammen. Die Grundschwingung eines Tons bestimmt den Ton, alle weiteren sogenannten Obertöne bestimmen den Klang: Beim Spielen des Tons A auf einem Klavier und auf einer Gitarre ergeben sich zwar meist die gleichen Obertöne, diese unterscheiden sich aber in der Länge und Intensität. So entsteht der für die jeweiligen Instrumente charakteristische Klang [Tob18], [Gas99]. Ein Beispiel für den Vergleich eines Klavier- mit einem Gitarrenton ist in Abbildung 21.
Eine reine Sinusschwingung mit nur einer Frequenz kann lediglich ein Synthesizer erzeugen. Hier gibt es dann keine Obertöne. In der zugehörigen Fouriertransformation in Abbildung 13a bzw. 14a sieht man lediglich einen Ausschlag bei 440 Hz, im Spektrogramm in Abbildung 13b bzw. 14b keine zeitliche Veränderung des Tons.

[11] `https://egalo.com/2012/05/03/notes-on-the-guitar-fretboard-introduction/#chart`
[12] `https://commons.wikimedia.org/wiki/File:Pianoforte_Klaviatur-wIKI_4.jpg`



(a) FFT des Synthesizer-Sinustons. Durch die fehlende Achsenanpassung erkennt man, dass lediglich eine einzige Frequenz im untersuchten Ton enthalten ist, und dass die Frequenz kleiner als 1000 Hz ist.



(b) Spektrogramm des Synthesizer-Sinustons. Durch die fehlende Achsenanpassung erkennt man, dass sich der gespielte Ton über die Zeit nicht ändert und keine Obertöne enthalten sind.

Abbildung 13: Vergleich der FFT und der Gabor-Transformation des Synthesizer-Sinustons a' ohne Zuschneiden der Achsen, um deutlich zu sehen, dass lediglich eine Frequenz und keine Obertöne im Ton enthalten sind.

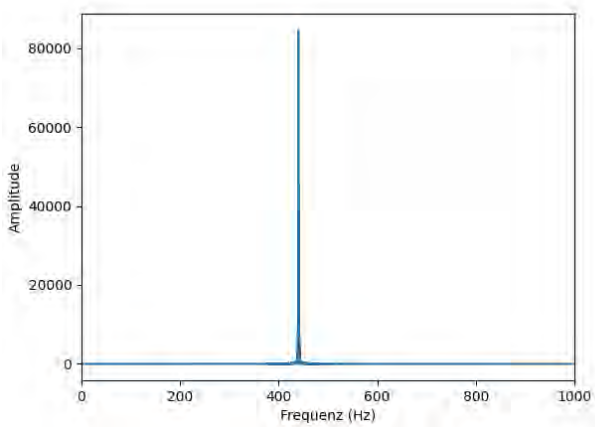
Damit man aus den folgenden Diagrammen genauere Werte erkennen kann, werden die Achsen angepasst und dann nur bestimmte Frequenzen bis zu einer Grenze betrachtet wie in Abbildung 14.

Nachfolgend werden verschiedene Audiodateien im WAV-Format mithilfe der eingangs beschriebenen Fourier- und Gabor-Transformation und im Kapitel 3 dargestellten Implementierung untersucht. Die selbst aufgenommenen Dateien wurden mit Logic Pro unter Verwendung des Focusrite Scarlett 2i2 mithilfe eines Klinkenkabels aufgenommen.
Betrachtete Instrumente:

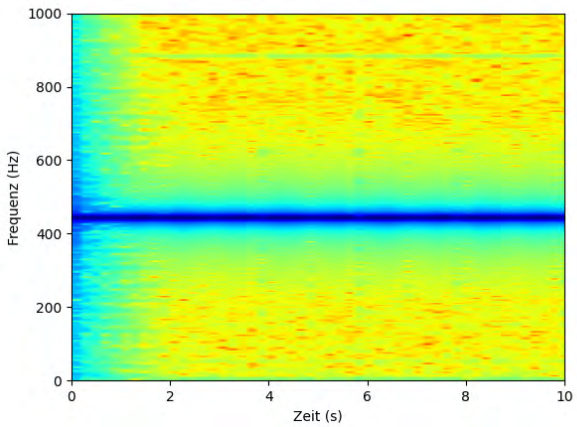
- Westerngitarre Furch Blue Gc-CM Element VTC
- Westerngitarre Takamine EG321C
- Stratocaster E-Gitarre marathon Replay Series
- Telecaster E-Gitarre Charvel Pro Mod So-Cal Style 2
- Keyboard Clavia Nord Stage 3 88 (Synthesizer- und Klavier-Sounds)

4.2 Synthesizer-Sounds

In der Synthesizer-Sektion des Keyboards gibt es verschiedene Einstellungen, u.a. das in den Abbildungen 13 und 14 betrachtete Sinussignal.



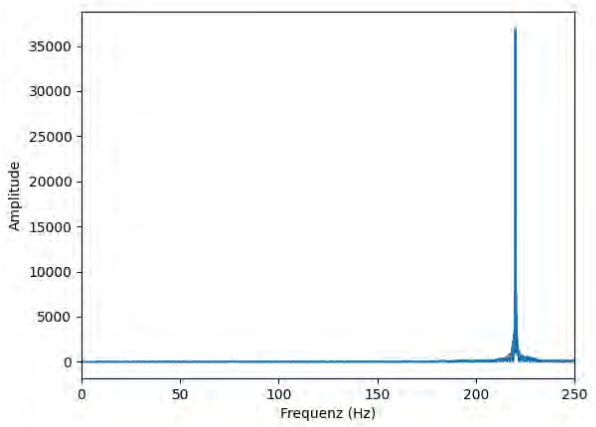
(a) FFT des Synthesizer-Sinustons. Mit angepasster Frequenz-Achse erkennt man, dass der betrachtete Ton zwischen 400 Hz und 500 Hz liegt. Passt man die Achsen noch weiter an, würde man die Frequenz bei 440 Hz einordnen können.



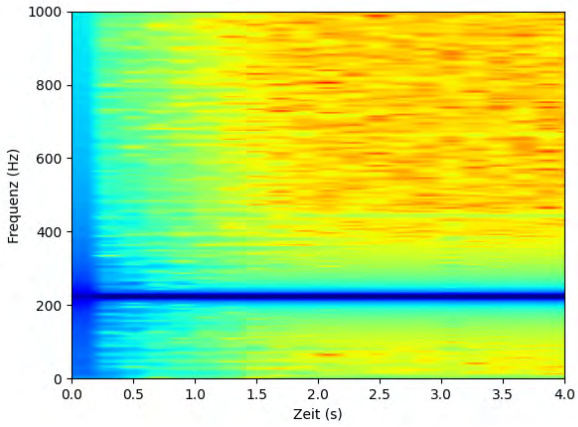
(b) Spektrogramm des Synthesizer-Sinustons. Mit angepasster Achse erkennt man wie in Abbildung 14a eine genauere Frequenz als in Abbildung 13 und ebenso keine weiteren Obertöne bis auf einen ganz leicht grün hervorstechenden Oberton bei etwa 900 Hz (vermutlich 880 Hz)

Abbildung 14: Vergleich der FFT und der Gabor-Transformation des Synthesizer-Sinustons a' mit angepassten Achsen. Der gespielte Ton a' mit 440 Hz ist viel besser zu erkennen als in Abbildung 13, klingt nicht nach bestimmter Zeit ab und enthält keine anderen Frequenzen.

Dasselbe Bild mit nur einer abgebildeten Frequenz ohne Obertöne ergibt sich für den um eine Oktave tieferen Ton a bei 220 Hz.



(a) FFT des Sinustons a mit einziger erkennbarer Frequenz bei 220 Hz

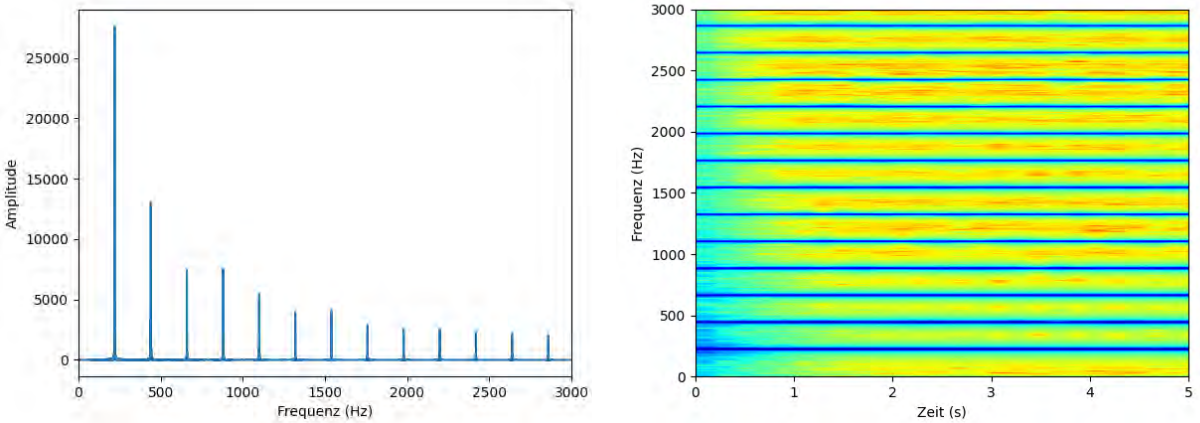


(b) Spektrogramm des Sinustons a, keine zeitliche Veränderung und praktisch keine Obertöne sichtbar

Abbildung 15: Fourier- und Gabor-Transformation des Synthesizer-Sinustons a mit 220 Hz

Im Unterschied dazu erhält man für den Synthesizer „Saw“ nicht nur einen Sinuston bei

einer Frequenz, sondern einen Ton aus mehreren Sinusschwingungen unterschiedlicher Frequenzen, die aber ebenfalls nicht ausklingen. In Abbildung 16 sieht man den hervortretenden Grundton bei 220 Hz sowie die Obertöne bei 440 Hz, 660 Hz und 880 Hz.



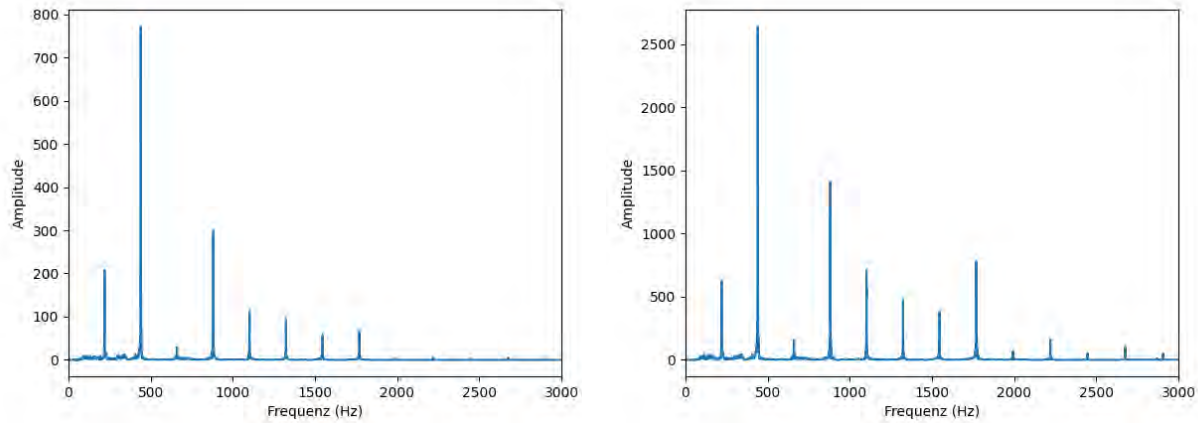
(a) FFT des Synthesizer-Saw-Tons a mit erkennbarem Grundton bei 220 Hz und Obertönen bei 440 Hz, 660 Hz, 880 Hz, usw.

(b) Spektrogramm des Synthesizer Saw-Tons a mit Obertönen in regelmäßigem Abstand und keiner zeitlichen Veränderung des Tons

Abbildung 16: Fourier- und Gabor-Transformation des Synthesizer-Saw-Tons a (220 Hz)

4.3 Klavier-Sounds

Beim Vergleich der Fouriertransformation eines Klaviertons in laut und leise sieht man die enthaltenen Frequenzen in Abbildung 17, wie stark der Grundton und die zugehörigen Obertöne nach welcher Zeit abklingen, sieht man im Spektrogramm in Abbildung 18.



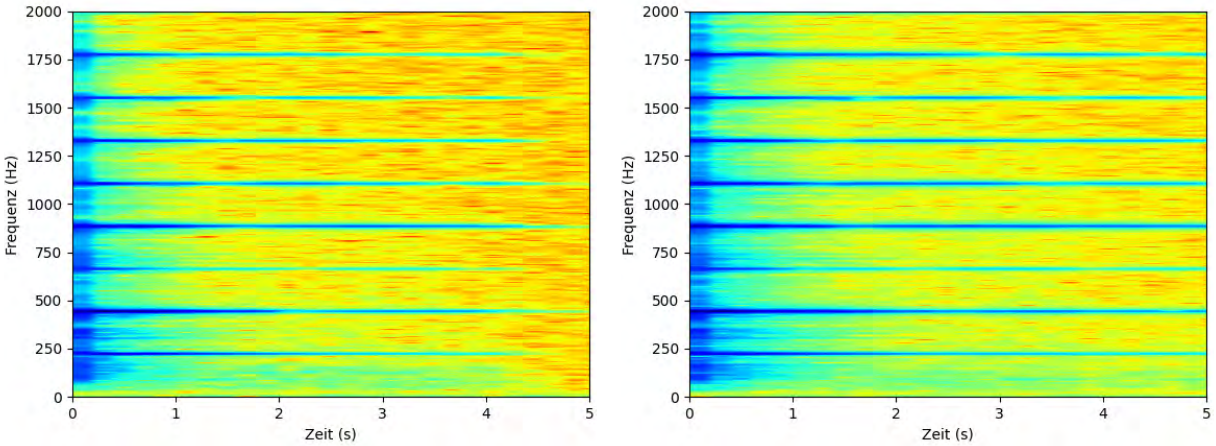
(a) FFT des leicht angespielten Klaviertons a

(b) FFT des stark angespielten Klaviertons a

Abbildung 17: Leicht und stark angespielter Klavierton im Vergleich: gleiche enthaltene Frequenzen in unterschiedlicher Intensität

Der Vergleich des Klaviertons in laut bzw. leise liefert beim Anhören Unterschiede in Dauer und Lautstärke, allerdings sind diese nicht auf ersten Blick sichtbar, weil das ausge-

gebene Spektrogramm automatisch angepasst wird auf Signallänge und Lautstärke. Nach Anpassung der abgebildeten Zeit im Spektrogramm auf eine Dauer von 5 s, ist erkennbar, dass der stärker angespielte Ton länger klingt.



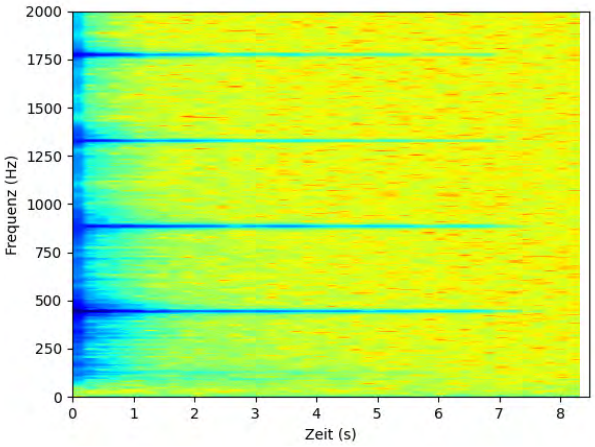
(a) Spektrogramm des leicht angespielten Klaviertons a

(b) Spektrogramm des stark angespielten Klaviertons a

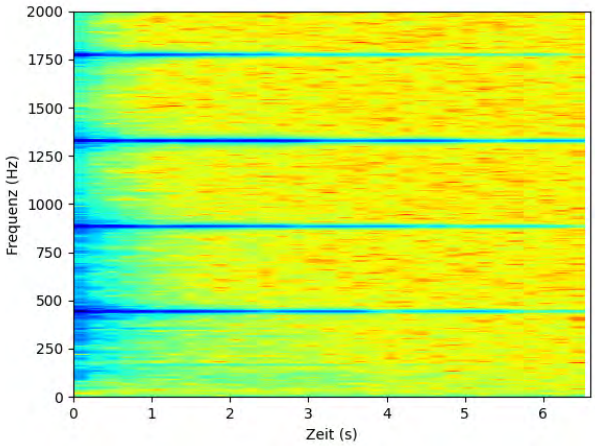
Abbildung 18: Spektrogramme für leicht bzw. stark angespielten Klavierton. Auffallend ist der nur leicht auftretende Ton bei 660 Hz

Deutlich zu sehen ist in beiden Bildern die Frequenz (220 Hz) des angespielten Tons a und die bis 2000 Hz abgebildeten Obertöne bei Vielfachen (440 Hz, 660 Hz 880 Hz, ...) wobei der Ton bei 660 Hz eher eine untergeordnete Rolle zu spielen scheint, während alle anderen Obertöne in etwa gleich lange und stark klingen.

Vergleicht man einen „normalen“ Klavier-Sound mit einem Honkytonk-Sound, so fällt auf, dass der Ton bei 1320 Hz beim Honkytonk genauso (oder sogar noch länger) hervortritt wie das gespielte a’ bei 440 Hz, während er beim Klavier deutlich kürzer und schwächer auftritt.
Am weißen Streifen rechts im Spektrogramm ist erkennbar, dass die Zeitachse nicht abgeschnitten wurde und wie lange die Aufnahme dauert: In Abbildung 19a sind es etwas mehr als 8 s, in Abbildung 19b ca. 6,5 s.

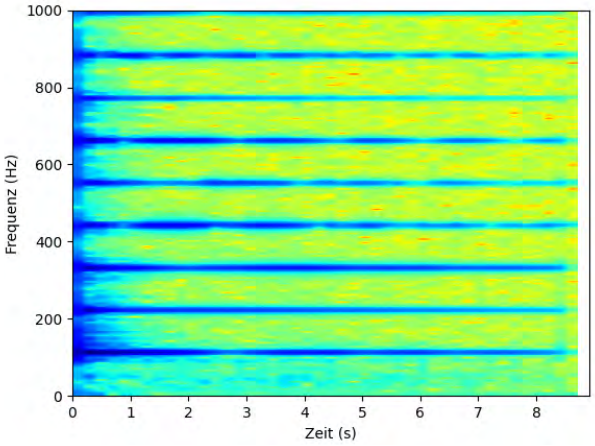


(a) Spektrogramm des Tons a' mit Klavier

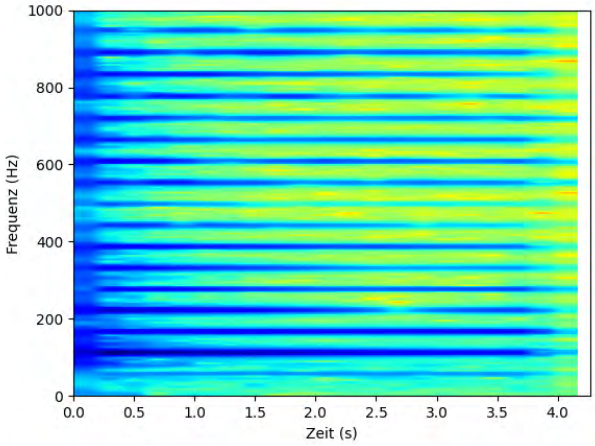


(b) Spektrogramm des Tons a' mit Honkytonk

Abbildung 19: Spektrogramme des Tons a', gespielt mit Klavier-Sound und Honkytonk-Sound. Größter Unterschied bei 1320 Hz



(a) Spektrogramm des Tons 'A mit Klavier



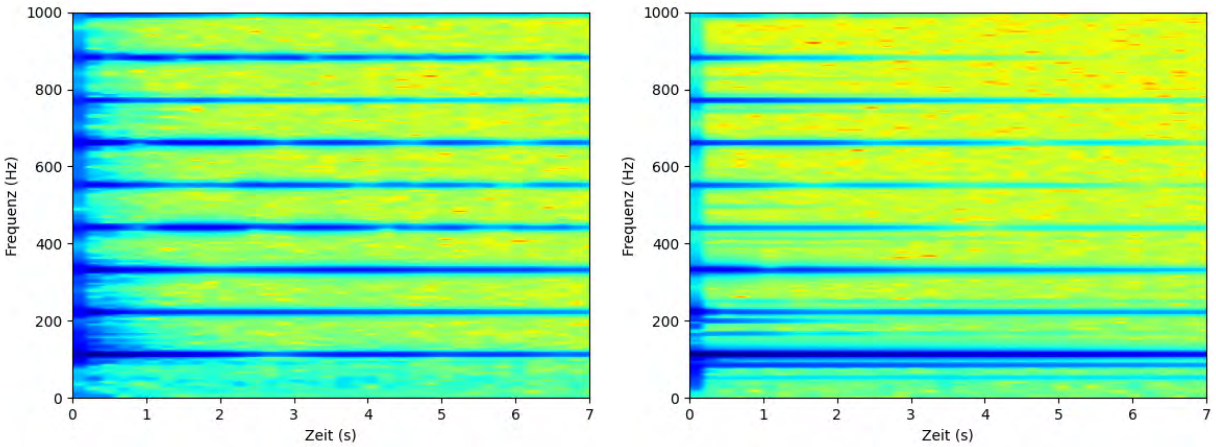
(b) Spektrogramm des Tons "A mit Klavier

Abbildung 20: Spektrogramme der Klaviertöne 'A und "A zeigen unterschiedlich viele Obertöne

Für den um eine Oktave tieferen Ton 'A (55 Hz) sind im Spektrogramm in Abbildung 20 doppelt so viele Obertöne im Gesamtton enthalten wie für den Ton A (110 Hz): Beim Ton A sind dies die Frequenzen bei Vielfachen von 110 Hz (220, 330, 440, 550, ...) und bei 'A Vielfache von 55 Hz (110, 165, 220, 275, ...).
Für den Ton a (220 Hz) und a' (440 Hz) ist dies in Abbildung 18 bzw. 19 zu sehen.

4.4 Gitarren

Vergleicht man den Klavierton A aus Abbildung 20a mit dem der Gitarre auf der A-Saite, so sieht man die gleichen Obertöne. Beim Gitarrenton klingen diese allerdings viel schneller ab, lediglich der Grundton bei 110 Hz und die beiden Töne bei 220 Hz und 330 Hz klingen annähernd so lange wie beim Klavierton. Die übrigen Obertöne klingen nach etwa 4 s oder sogar schon vorher ab, während beim Klavier praktisch alle Obertöne länger klingen, lediglich bei der Frequenz 770 Hz klingt der Ton schon vorher nicht mehr. Interessant ist hierbei, dass eben dieser Ton bei der Gitarre noch mit am längsten und am stärksten klingt, während er beim Klavier eher kürzer und nicht so laut klingt.



(a) Spektrogramm des Klaviertons wie in Abbildung 20a, mit verkürztem Plot auf 7 s

(b) Spektrogramm des Gitarrentons der Saite A (110 Hz), vermutlich mit Störsignal unter 100 Hz

Abbildung 21: Spektrogramme des Tons A gespielt auf Klavier bzw. Gitarre zeigen unterschiedliche Obertöne.

Betrachtet man nicht nur einzelne Saiten, sondern einen A-Dur-Akkord gespielt mit Gitarre, so ergibt sich das Spektrogramm in Abbildung 22. Gespielte und u.a. erkennbare Töne beim A-Dur-Akkord:

- A-Saite: Ton A (110 Hz)
- d-Saite 2. Bund: Ton e (ca. 330 Hz)
- g-Saite 2. Bund: Ton a (440 Hz)
- h-Saite 2. Bund: Ton c# (555 Hz)
- e'-Saite: Ton e' (660 Hz)

Weitere erkennbare Frequenzen bei etwa 220 Hz oder 880 Hz sind Obertöne der gespielten Töne. Besonders schön sieht man das unterschiedliche Abklingen der einzelnen Töne, während man bei einer Fouriertransformation nur die enthaltenen Frequenzen sehen würde. Einige Frequenzen des Signals sind nicht erklärbar, etwa die leicht hellblaue Linie bei 50 Hz. Bei der Gitarre können das verschiedenste Störgeräusche sein, die durch die Stromver-

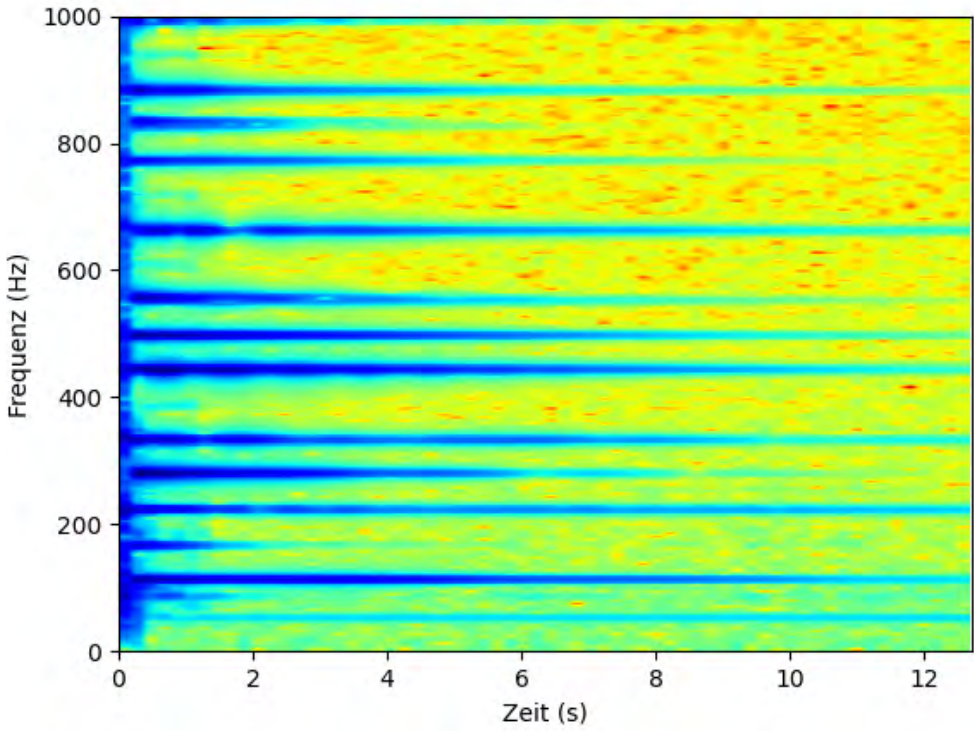
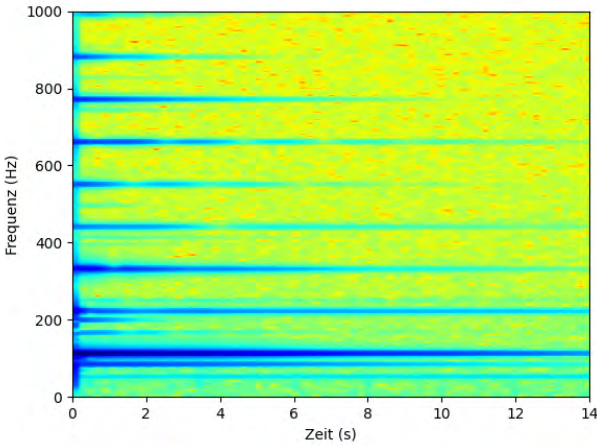


Abbildung 22: Spektrogramm des A-Dur-Akkords auf der Gitarre

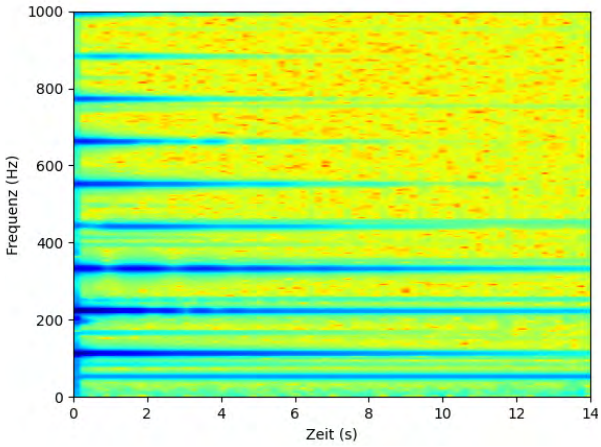
sorgung bei der Aufnahme, beim unsauberen Greifen der Akkorde und Töne oder durch zu starkes Anschlagen der Saiten auftreten können [Van17], [SS].

4.4.1 Vergleich Westerngitarren

In Abbildung 23 sind Spektrogramme für den Ton A zweier Gitarren. Bei der Furch-Gitarre (Abbildung 23a) sticht der angespielte Ton A am meisten hervor, im Vergleich dazu klingen bei der Takamine-Gitarre (Abbildung 23b) die Töne bei 110, 220 und 330 Hz etwa gleich stark und gleich lang. Vor allem beim a mit 220 Hz sieht man einen großen Unterschied im Vergleich. Bei der Takamine-Gitarre treten die Obertöne stärker und länger hervor als bei der Furch-Gitarre.
Bei beiden Gitarren gibt es außerdem ein Störsignal bei ca. 50 Hz.



(a) Spektrogramm des Gitarrentons A, gespielt mit der A-Saite auf der Furch-Gitarre

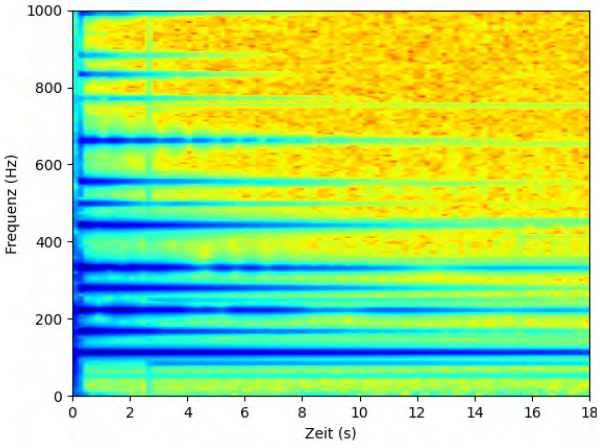


(b) Spektrogramm des Gitarrentons A, gespielt mit der A-Saite auf der Takamine-Gitarre

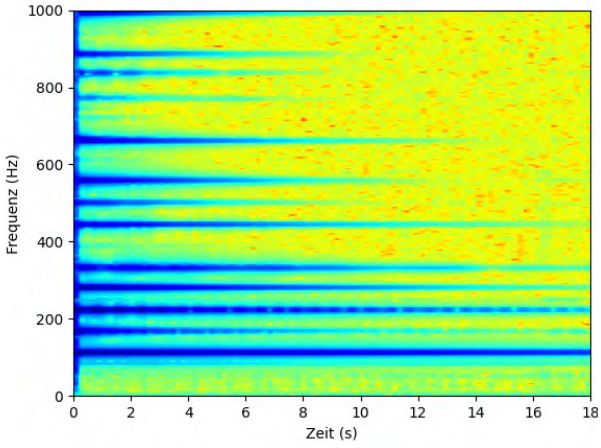
Abbildung 23: Spektrogramme des Tons A, gespielt mit den zwei Westerngitarren

4.4.2 Vergleich E-Gitarren

Im Vergleich zum A-Dur-Akkord auf der Westerngitarre 22 klingen die Akkorde auf den beiden E-Gitarren länger. Zwischen den beiden E-Gitarren ist der Unterschied beim Akkord nicht sehr auffällig: Die abgebildeten Frequenzen sind bei beiden vorhanden und auch die Dauer der jeweiligen Teiltöne unterscheidet sich nicht großartig. Nur höhere Frequenzen treten bei der Telecaster-E-Gitarre in Abbildung 24b besser hervor als bei der Stratocaster-E-Gitarre in Abbildung 24a.

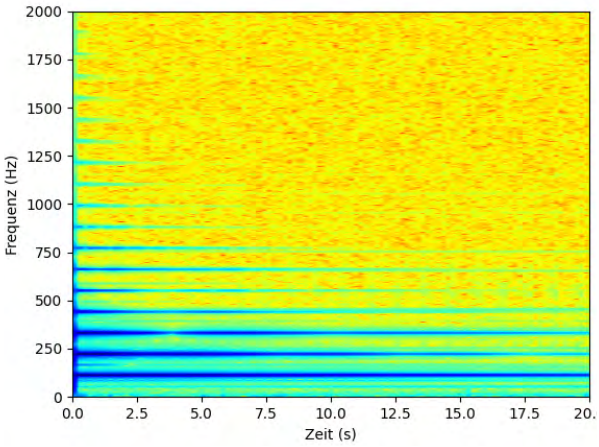


(a) Spektrogramm des Akkords A, gespielt auf der Stratocaster-E-Gitarre

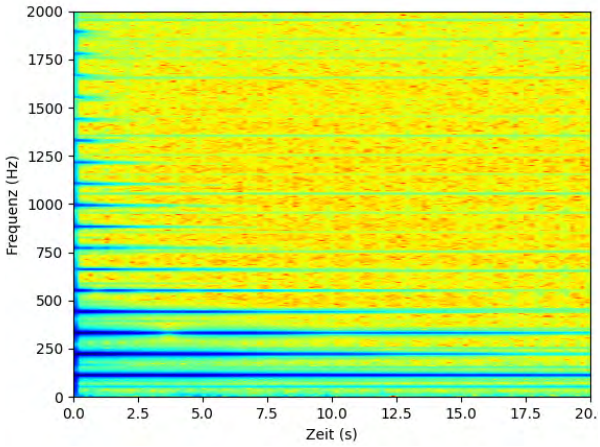


(b) Spektrogramm des Akkords A, gespielt auf der Telecaster-E-Gitarre

Abbildung 24: Spektrogramme des Akkords A, gespielt mit den E-Gitarren



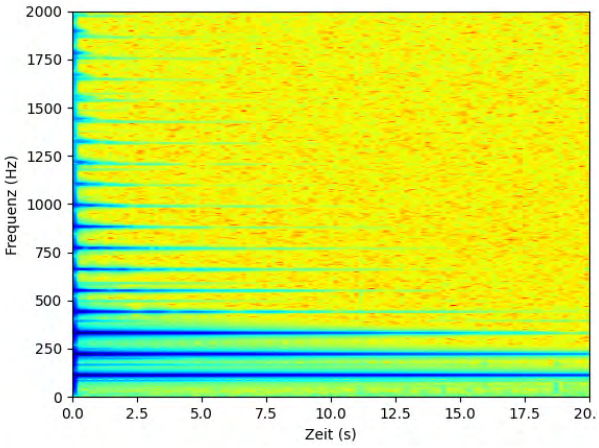
(a) Spektrogramm des Tons der Saite A, gespielt auf der Stratocaster-E-Gitarre mit Tonabnehmer oben



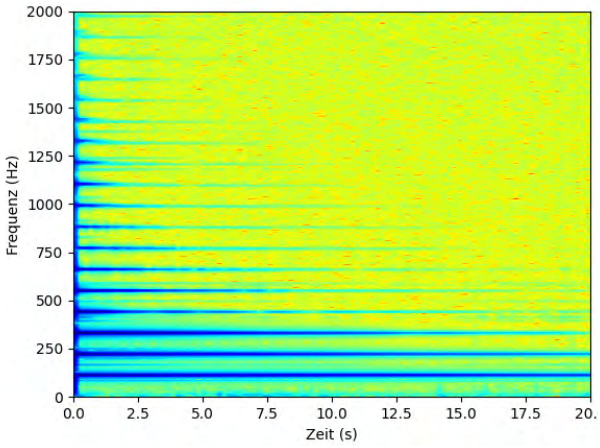
(b) Spektrogramm des Tons der Saite A, gespielt auf der Stratocaster-E-Gitarre mit Tonabnehmer Mitte

Abbildung 25: Spektrogramme des Tons A mit zwei verschiedenen Tonabnehmern der Stratocaster-E-Gitarre

In Abbildung 25 werden die verschiedenen Tonabnehmer einer E-Gitarre verglichen. Es ergibt sich kein herausragender Unterschied im Spektrogramm bis etwa 1000 Hz. Höhere Töne mit Frequenzen ab 1000 Hz sind beim oberen Tonabnehmer nicht so vertreten wie beim mittleren Tonabnehmer. Im Spektrogramm 25b sind höhere Töne besser erkennbar. Ebenfalls wenig Unterschied sieht man bei den beiden Tonabnehmern der Telecaster-E-Gitarre in Abbildung 26. Sogar beim Vergleich der Abbildungen 25 und 26 erkennt man wenig Unterschied bis auf die beschriebenen Obertöne in Abbildung 25b.



(a) Spektrogramm des Tons der Saite A, gespielt auf der Telecaster-E-Gitarre mit Tonabnehmer oben



(b) Spektrogramm des Tons der Saite A, gespielt auf der Telecaster-E-Gitarre mit Tonabnehmer unten

Abbildung 26: Spektrogramme des Tons A mit den zwei Tonabnehmern der Telecaster-E-Gitarre

4.4.3 Melodie: Happy Birthday

Mit der Gabor-Transformation kann man nicht nur das Abklingen der verschiedenen Töne untersuchen, sondern auch eine Melodie.
In der Notenzeile 27 steht der Anfang der Melodie des Lieds „Happy Birthday“ in Notenschrift, allerdings vereinfacht: ohne Pausen und nur als Viertelnoten.

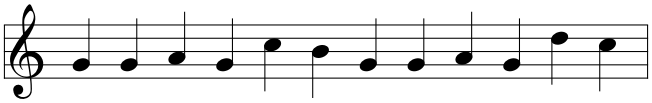


Abbildung 27: Anfang der Melodie zu Happy Birthday

Im Spektrogramm zu dieser mit Gitarre gespielten Melodie in Abbildung 28 sind die unterschiedlichen Tonhöhen gut zu erkennen. Wenn man die Obertöne ignoriert, sieht das Spektrogramm der Notenzeile in Abbildung 27 sehr ähnlich.

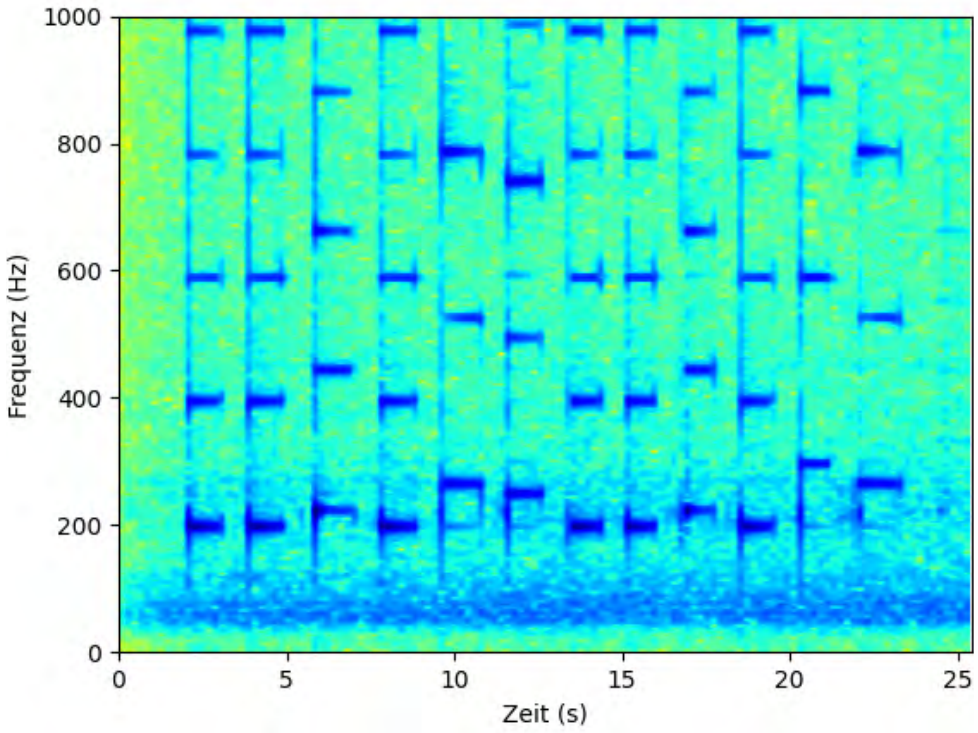


Abbildung 28: Der Anfang von Happy Birthday gespielt auf der Gitarre, Noten dazu in Abbildung 27

5 Zusammenfassung

Mit der Gabor-Transformation wurden im Gegensatz zur Fouriertransformation sowohl der zeitliche Verlauf als auch spektrale Informationen von Musikdateien berücksichtigt. Dies ist von besonderer Bedeutung, da fast alle betrachteten Signale nicht stationär sind, sondern sich mit der Zeit verändern.

Die eigene Implementierung der Gabor-Transformation in Python hat einen Einblick in die Berechnung der FFT und der Gabor-Transformation geboten, ist aber aufgrund zu langer Laufzeiten nicht praxistauglich. In der Praxis effizient und vielseitig in unterschiedlichen Anwendungen einsetzbar sind dafür die im Package `numpy` und `matplotlib` implementierten Methoden, wie die Analyse der Musikdaten verschiedener Instrumente zeigt.
Hier könnte man zum Beispiel noch andere Instrumente analysieren und mit den bereits betrachteten vergleichen oder die Laufzeiten der implementierten Methoden verbessern.
Neben der Zusammensetzung einzelner Töne verschiedener Instrumente ist besonders das Spektrogramm für eine Melodie hervorzuheben, das der in der Musik verwendeten Notenschrift erstaunlich ähnlich sieht.

Zusammenfassend sieht man in der Bachelorarbeit den Einsatz der Gabor-Transformation zur Untersuchung von Musikdateien, die mit verschiedenen Instrumenten aufgenommene Töne enthalten.

Literatur

[Coh89] Leon Cohen. Time-frequency distributions-a review. *Proceedings of the IEEE*, 77(7):941–981, 1989.

[CT65] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[Fei98] Thomas Feichtinger, Hans G. und Strohmer. *Gabor analysis and algorithms : theory and applications*. Birkhäuser, 1998.

[For10] Peter Forster, Brigitte und Massopust. *Four Short Courses on Harmonic Analysis*. Birkhäuser, 2010.

[Gab46] Dennis Gabor. Theory of communication. part 1: The analysis of information. *Journal of the Institution of Electrical Engineers-part III: radio and communication engineering*, 93(26):429–441, 1946.

[Gas99] Patrick Gasquet, Claude und Witomski. *Fourier Analysis and Applications*. Springer, 1999.

[Goe18] Steffen Goebbels. *Mathematik verstehen und anwenden - von den Grundlagen bis zu Fourier-Reihen und Laplace-Transformation*. Springer Spektrum, 2018.

[Grö01] Karlheinz Gröchenig. *Foundations of time-frequency analysis*. Springer Science & Business Media, 2001.

[SS] Carlos San Segundo. Gitarre aufnehmen am PC – Tutorial. `https://www.delamar.de/recording/tutorial-gitarre-am-computer-aufnehmen-943/`, zuletzt besucht am: 2023-09-21.

[Tob18] Andreas Tobola. Tonfrequenzen in der Westlichen Musik, 2018. `https://tnotes.de/NotenFrequenzen`, zuletzt besucht am: 2023-09-05.

[Van17] Andreas Vandenhoff. Geräusche und Unsauberkeiten beim Greifen, Loslassen und Rutschen über die Saiten, 2017. `https://gitarrenblog.vandenhoff.de/geraeusche-und-unsauberkeiten-beim-greifen-loslassen-rutschen-ueber-die-saiten/`, zuletzt besucht am: 2023-09-21.

[vdB05] R. v. d. Boogaart, C. G und Lienhart. Fast gabor transformation for processing high quality audio. University of Augsburg, 2005.

[Wer06] Martin Werner. *Digitale Signalverarbeitung mit MATLAB*. Vieweg, 2006.

Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig und ohne unzulässige Hilfe verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die wörtlich und sinngemäß übernommenen Passagen aus anderen Werken kenntlich gemacht habe.
Die Arbeit ist weder von mir noch von einer anderen Person an der Universität Passau oder an einer anderen Hochschule zur Erlangung eines akademischen Grades bereits eingereicht worden.

Ort, Datum	Christina Praml