Gewerbliche und Hauswirtschaftlich- Sozialpflegerische Schulen
Emmendingen

Technisches Gymnasium Emmendingen

Jahrgangsstufe 1

2022/2023

Seminararbeit zum Thema:

Musikvisualisierung & Darstellung

Eingereicht von:

Felix Zimmermann TGI12
Julian Dold TGI12

Emmendingen den, 08.06.2023

Inhaltsverzeichnis

1. Einleitung

Musik ist seit Beginn der Menschheitsgeschichte ein entscheidender Bestandteil von dieser. Von Blockflöten in der Altsteinzeit, über Wiener Klassik mit Wolfgang Amadeus Mozart oder van Beethoven, bis hin zu heutigen Popikonen wie Katy Perry, Taylor Swift oder Avril Lavigne[1], von natürlicher Klassik zu ausgefallenem Jazz, über hartem Rock, melodischem Pop oder schnellem Hip-Hop, Musik hat sich über die Jahrzehnte immer weiterentwickelt und neue außergewöhnliche Genres hervorgeholt.

Darstellungen und Bilder nehmen zudem eine immer größere Bedeutung in der Musik ein. Wir Menschen verbinden seit jeher Gefühle und Erinnerungen mit Musik und erstellen uns ein eigenes Bild dazu. Seit der Erfindung der ersten Lichtorgel oder dem ersten Visualisierungsgerät von Atari[2] waren Menschen erstaunt von der Visualisierung der Musik. Die Möglichkeit Rhythmik und Melodie visuell darzustellen ist faszinierend. So wird dies auch heute noch gemacht. Von großen Lichtshows auf Konzerten bis hin zum kleinen Ambient-Light hinter dem Fernseher.

Die Audiovisualisierung ist ein großer Teilbereich der digitalen Signalverarbeitung und enthält mehrere Teilbereiche. Die Visualisierung des Frequenzspektrums gehört hier schon zu den Standardtechniken, welche heutzutage sehr beliebt ist, genauso wie Effekte zur Darstellung der Lautstärke, Melodie oder Rhythmik in der Musik.

Die vorliegende Arbeit befasst sich mit Konzepten und Algorithmen zur Audiovisualisierung. Hierbei soll erkenntlich gezeigt werden, wie heutige Lichteffekt-Systeme und Sprachanalysen oft funktionieren, welche Techniken dafür angewandt werden und schließlich, wie Musik visualisiert werden kann. Des Weiteren sollen diese Techniken durch ein eigenes Programm am Beispiel einer LED-Leuchte angewandt werden.

Im ersten Teil der Arbeit werden die notwendigen technischen Grundlagen, sowie die Konzepte der Audio-Signalverarbeitung gezeigt und näher erläutert. Dabei wird die Frage behandelt, wie aus einem “rohen” Signal aussagekräftige Daten, wie die Frequenz oder Rhythmik, erfasst werden können. Mathematische Methoden wie die Fourier-Transformation oder die logarithmische Hörwahrnehmung von uns Menschen halten hier Einzug.

Der zweite Teil der Arbeit widmet sich der konkreten Darstellung der analysierten Daten, die Gestaltung verschiedener Effekte sowie Farbmuster werden hier behandelt, ebenso die Entwicklung eines Programmes zur Umsetzung dieser Techniken. Testen der angewandten Konzepte ist hier ebenfalls ein Stichwort.

Diese Seminararbeit soll gut verständlich sein, daher werden hier keine mathematischen Themen näher erläutert oder erklärt, wie zum Beispiel die Summenformel oder Komplexe Zahlen. Zudem werden Themen zur direkten Ton-/ oder Spracherkennung hier keinen großen Platz haben, obwohl viele Themen darauf aufbauen. Die Seminararbeit befasst sich im Kern mit Musikvisualisierung und Darstellung und nicht mit digitaler Spracherkennung.

[1] Vgl.: Huber, Martin, 2018, *Wie Musik entstanden ist.*

[2] Siehe dazu: *Blue Monday on Atari C-240,* Letzter Aufruf am 16.01.23: https://www.youtube.com/watch?v=etZVnILnoSg.

2. Digitale Signale

Dieses Kapitel der Arbeit widmet sich der Darstellung und Speicherung digitaler Signale. Hierbei wird die Frage behandelt, wie ein digitales Signal dargestellt wird und welche Schritte zur Umwandlung aus einem analogen Signal notwendig sind.

2.1 Abtastung

Musik selbst besteht aus verschiedenen Schallwellen, welche der Mensch hören kann. Eine Schallwelle ist ein kontinuierliches Signal, welches zu jedem Zeitpunkt t einen passenden Signalwert $s(t)$ hat.[3] In der Mathematik ist dies vergleichbar mit der Sinus- oder Cosinus-Funktion, wo es zu jedem Zeitpunkt t ein Funktionswert $\sin(t)$ gibt. Problem hierbei ist die Speicherung eines analogen Signales. Da es ein kontinuierliches Signal ist, welches unendlich Stellen hat, ist es nicht möglich dieses Signal in einem digitalen Speicher zu sichern und mit dem Computer zu untersuchen. Das analoge Signal muss hierzu digitalisiert werden.

Von dem analogen Signal werden in einem gewissen Zeitabstand Ts n Punktproben gemacht. Dieser Vorgang wird Abtastung genannt (*Siehe Abbildung 1*).

$$x[n] = s(n * Ts) \qquad n \in \mathbb{N}$$

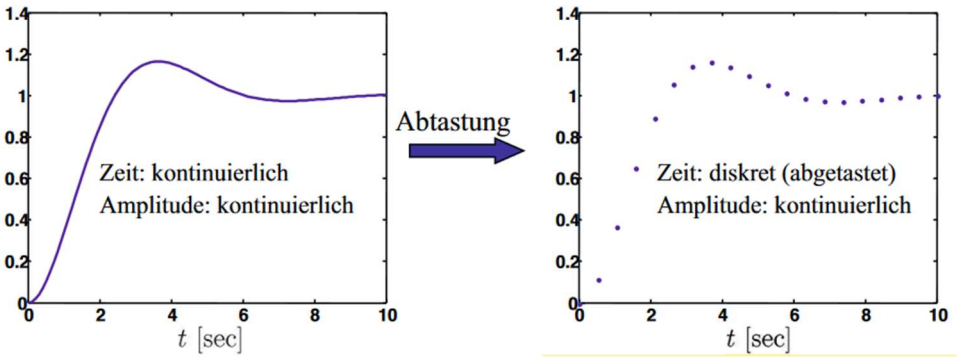


Abbildung 1: Abtastung

Als Ergebnis bekommt man eine diskrete Folge von Punktproben $x[n]$ des analogen Signals. Eine Punktprobe wird im Englischen auch ***Sample*** genannt. Um Verwechslungen zwischen analogen und diskreten digitalen Signalen zu vermeiden, werden hinweg alle digitalen Signale als Folge $x[n]$ geschrieben.[4]

Bei einem digitalen Signal gibt es allerdings keine Zeitangabe mehr. Es ist schließlich nur eine Folge von Proben. Nur der Zeitabstand Ts gibt Auskunft über die gemessene Zeitlänge des Signals. Ein weiterer Wert ist die Abtastrate fs, im Englischen auch ***Samplerate*** genannt. Die Abtastrate beschreibt die Häufigkeit an Punktproben innerhalb einer Sekunde. Sie wird in Hertz angegeben. Berechnen lässt sie sich, indem man 1 durch den Zeitabstand Ts dividert.

$$fs = \frac{1}{Ts}$$

Eine Abtastfrequenz von 44100 Hz bis 48000 Hz sind heute die am häufigsten verwendeten Frequenzen[5]. Bei einer zu niedrigen Abtastfrequenz kommt es allerdings zu Signalfehlern. Es tritt Aliasing[6] auf und eine saubere Frequenzanalyse wird fast unmöglich.

[3] Vgl.: Stroh, o. D, *Akustik, Kapitel 4: Analog und digital.*
[4] Vgl.: TU Wien, o. D, *Abtasttheorem.*
[5] Vgl.: McFee, o. D., *Digital Signals Theory, 2. Digital sampling.*
[6] S.: McFee, o. D., *Digital Signals Theory, 2. Aliasing.*

2.2 Quantisierung

Das kontinuierliche analoge Signal wurde in ein zeitdiskretes Signal, mit einer endlichen Anzahl an Samples, verwandelt.

Die Signalwerte dagegen selbst sind noch kontinuierlich dargestellt. Ein Signalwert kann bis jetzt jede mögliche Zahl, mit beliebig vielen Nachkommastellen, darstellen. Für eine digitale Speicherung muss der Darstellungsbereich auf eine endliche Menge begrenzt werden. Das Signal wird ***quantisiert***.

Die Anzahl an Darstellungsbereichen wird auch als ***Bittiefe*** bezeichnet.

$$L = 2^w$$

L gibt die Anzahl an Quantisierungsniveaus an, während w die Anzahl an verwendeten Bits zur Speicherung darstellt. Bei einer Quantisierung mit 8-Bit, kann ein Signalwert 255 Zustände annehmen. Bei einer Quantisierung mit 16-Bit sind es bereits 65.536 mögliche Zustände. Je weniger mögliche Quantisierungsniveaus es gibt, desto höher ist die Abweichung vom analogen Signal[7]. (*Siehe Abbildung 2*)

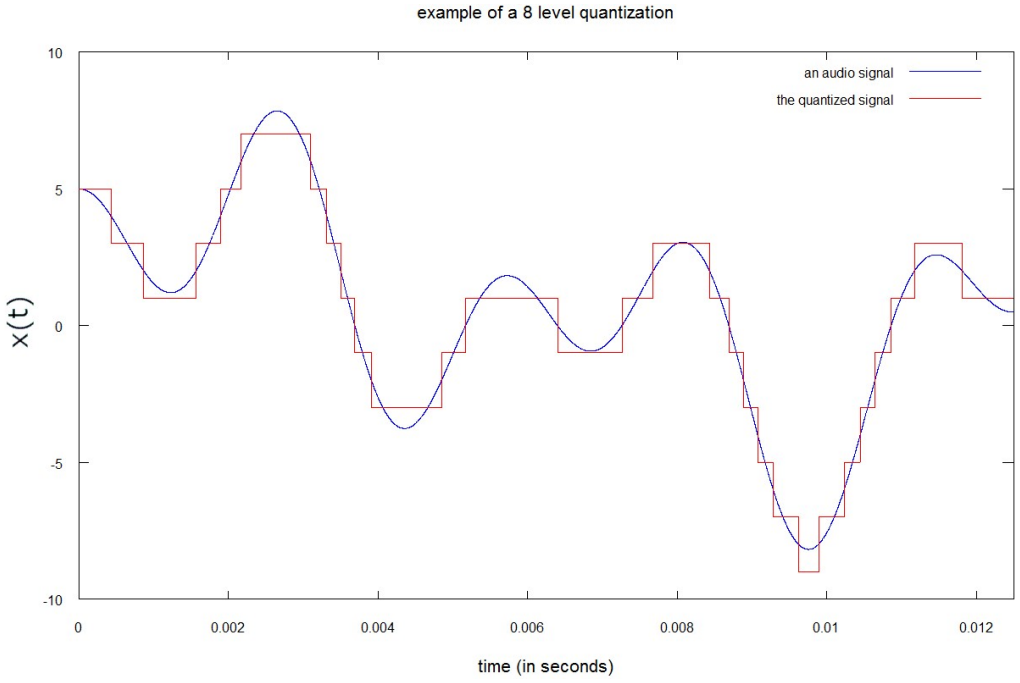


Abbildung 2: 3-Bit Quantisierung[8]

Diese Abweichung vom analogen Signal wird als Quantisierungsfehler bezeichnet. Bei einer hohen Bittiefe gibt es viele Zustände, und eine so geringe Abweichung vom analogen Signal, dass das menschliche Ohr diesen Unterschied nicht mehr wahrnimmt. Bei einer niedrigeren Bittiefe dagegen, kann die Abweichung hörbar sein.

Die Bittiefe muss für den jeweiligen Anwendungsbereich einzeln gewählt werden. Mehr Bits brauchen mehr Speicherplatz. Für eine Spracherkennung reicht eine 8-Bit Quantisierung. Für Musik werden standardmäßig 16-Bits verwendet.[9]

[7] Vgl.: Vary; Heute; Hess, *Digitale Sprachsignalverarbeitung, Seite 238.*
[8] Abbildung 2: s. MatheRetter, o. D., *Quantisierung.*
[9] Vgl.: MatheRetter, o.D., *Quantisierung.*

3. Signalverarbeitung

Das vorliegende Kapitel befasst sich mit der grundlegenden Analyse von Musikdaten, um die einzelnen Eigenschaften, wie Lautstärke, Frequenz, Rhythmik zu erkennen, analysieren und darzustellen.

3.1 Pre-Emphasis

3.1.1 Rauschen

Bei der Umwandlung von einem analogen Signal in ein digitales Signal entsteht immer ein gewisses Störsignal, auch Rauschen genannt. Ein Beispiel hierfür ist der Quantisierungsfehler aus *Kapitel 2.2*. Diese ungewollte Abweichung vom originalen Signal wird als Quantisierungsrauschen bezeichnet. Rauschen entsteht aber auch in elektronischen Bauteilen, wie Transistoren oder Dioden. Thermisches Rauschen bei Elektronik entsteht beispielsweiße bei größerer Wärmeentwicklung bei Bauteilen.[10]

Rauschen kennt man auch vom Radio hören. Bei zu weiter Entfernung vom Sendegerät wird das Signal immer schwächer und trifft auf andere Störsignale. Am Ende hört man mehr Rauschen als Musik.

Gerade hohe Frequenzen sind von Rauschen stärker betroffen, da sie viel weniger Signalenergie als tiefe Frequenzen aufweisen können. Wenn eine Signalquelle, wie eine Musikbox, in weiter Entfernung steht, werden auch tiefe Frequenzen mehr wahrgenommen als hohe Frequenzen.

Um diesem Problem entgegenzuwirken werden hohe Frequenzen beim Übertragen von Signalen angehoben. Dies beschreibt die sogenannte ***Pre-Emphasis***. Diese Anhebung findet vor allem bei der Übertragung von UKW-Funk Anwendung statt, da auch hier hohe Frequenzen bei der Übertragung weniger Energie aufweisen können als tiefe Frequenzen. Um beim Hören das originale Signal auch wieder zu haben, werden beim UKW-Empfänger die hohen Signale wieder abgesenkt. Auch ***De-Emphasis*** genannt. [11]

3.1.2 Anwendung

Auch für eine Visualisierung ist der hohe Energieanteil von tiefen Frequenzen bei manchen Visualisierungen ein Problem.

Bei einer Darstellung der Energie in der Musik ist beispielsweiße dieser hohe Energieanteil erwünscht. Der Bass soll stärker, als hohe Frequenzen, gewichtet werden. Bei einer Darstellung des Frequenzspektrums allerdings, kann dies durch die hohe Energiedifferenz, zu Problemen bei der Visualisierung führen. Frequenzwechsel können durch die Energie in tiefen Frequenzen schlecht, bis gar nicht mehr wahrgenommen werden (*siehe Abbildung 3*).

[10] Vgl. Habil. Neumann, 2010, *Die Wahrnehmung von Rauschen und dessen praktischer Einsatz zur Klanggestaltung.*

[11] Vgl.: https://www.youtube.com/watch?v=5AxB8_L6is8 Letzter Aufruf am 27.01.23.
Vgl.: itwissen.info, 2016, *Emphasis.*
Vgl.: Fayek, 2016, *Speech Processing for Machine Learning.*

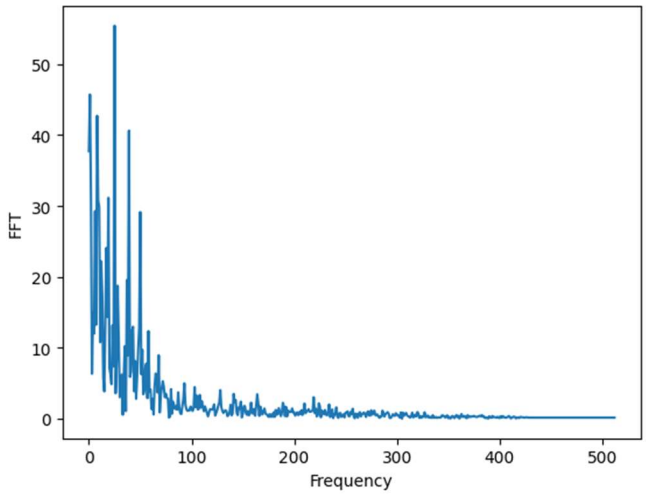


Abbildung 3 Frequenzsektrum ohne Pre-Emphasis

Daher verwenden wir auch hier eine Pre-Emphasis, um die Frequenzen zu normalisieren. Anders als bei einer Übertragung, welche eine Anhebung der hohen Frequenzen benötigt, um Störungen in hohen Frequenzen zu vermeiden, wird hier lediglich eine Normalisierung gebraucht. Anstatt einer Anhebung, nehmen wir eine Senkung der tiefen Frequenzen vor. Ein Pre-Emphasis Filter kann wie folgt implementiert werden: [12]

$$y[n] = x[n] - a * x[n-1]$$

Tiefe Frequenzen haben eine langsamere zeitliche Änderung als hohe Frequenzen. Daher neigen benachbarte Samples zu ähnlichen Werten. Durch die Subtraktion wird dieser Teil der tiefen Frequenz entfernt. a dient hier als Koeffizient und gibt die Stärke des Pre-Emphasis Filters an. Bei einem Wert von $a = 1.0$ werden alle tiefen Frequenzen herausgefiltert während bei einem Wert von $a = 0.1$ fast gar keine Frequenzen herausgefiltert werden. Ein Wert von ca. $a = 0.8/0.9$ haben wir hier als Optimum für eine Normalisierung empfunden.

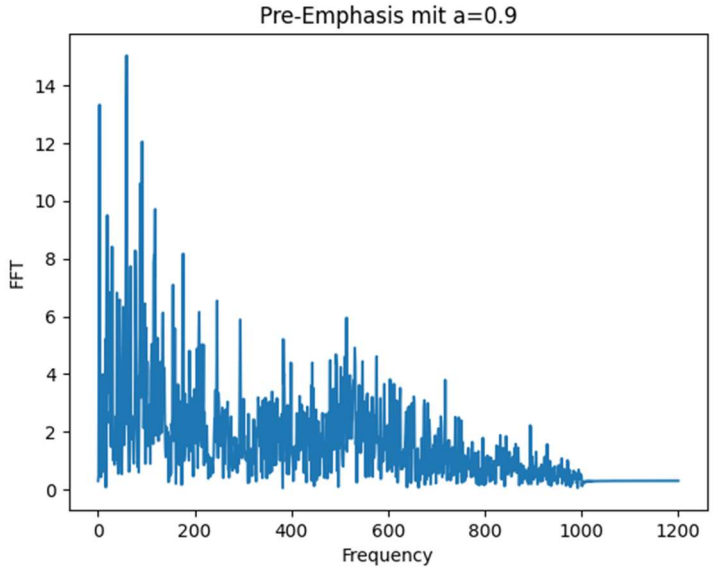


Abbildung 4: Frequenzspektrum mit Pre-Emphasis

[12] Vgl.: Fayek, 2016, *Speech Processing for Machine Learning*.

3.2 Fourier-Transformation[13]

Wenn man Musik visualisieren will, kommt man nicht um die Fourier-Transformation herum. Sie ist ein wichtiges Konzept in der Mathematik, Signalverarbeitung und in vielen anderen Bereichen. Bekannte Anwendungsbeispiele sind Kompression von digitalen Daten, Filtern von Störgeräuschen und die Signalanalyse. Wir benötigen sie, um das rohe Signal in seine Frequenzkomponenten zu zerlegen. Mithilfe dieser Informationen können wir, nach weiterer Verarbeitung, die Musik visuell darstellen. In diesem Kapitel geht es darum welche Arten von Fourier-Transformationen es gibt, wie die Grundlagen funktionieren und welche wir für unser Programm nutzen.

3.2.1 Fourier Reihe[14]

Um die Fourier-Transformation besser zu verstehen, schauen wir uns zunächst die Fourier Reihe an. Mithilfe der Fourier Reihe kann jede periodische Funktion als Summe von unendlich vielen Sinus- und Kosinus Funktionen dargestellt werden.

Die Summenschreibweise für die Fourier Reihe lautet wie folgt:

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k \cdot \cos(k\omega_0 t) + b_k \cdot \sin(k\omega_0 t)) \qquad \omega_0 = \frac{2\pi}{T}$$

Dieser Prozess kann auch umgedreht werden, dass bedeutet ein periodisches Signal kann in seine einzelnen Bestandteile zerlegt werden. Dabei erhält man die Grundwelle und die dazugehörigen Oberwellen. Diese Zerlegung der periodischen Funktion in eine Fourier Reihe wird Fourier Analyse genannt.

Dazu können die Fourier Koeffizienten $a_0, a_1, \dots, b_1, b_2, \dots$ mithilfe den folgenden Integralformeln bestimmt werden:

$$a_0 = \frac{1}{\pi}\int_0^{2\pi} f(x)dx$$

$$a_n = \frac{1}{\pi}\int_0^{2\pi} f(x) \cdot \cos(nx)\, dx \qquad \text{für } n = 1,2,3, \dots$$

$$b_n = \frac{1}{\pi}\int_0^{2\pi} f(x) \cdot \sin(nx)\, dx \qquad \text{für } n = 1,2,3, \dots$$

Es wird das Produkt aus der periodischen Funktion und einer Basisfunktion integriert. Durch die Orthogonalität von Sinus und Kosinus kann erkannt werden, ob die Basisfunktion in der periodischen Funktion enthalten ist oder nicht.

Um das besser zu verstehen, schauen wir uns an was dort eigentlich dahintersteckt. Das kann am besten grafisch erklärt werden. Dazu gehen wir ein Beispiel durch an dem wir die grundlegende Idee der Fourier Analyse erkennen können.

Wir nehmen an wir haben die folgende periodische Funktion:

$$f(x) = \sin(2x) + 2\cos(3x)$$

[13] Vgl.: Nelles, 2019., *Vorlesung Signalverarbeitung, Kapitel 3.*

[14] Vgl.: https://www.youtube.com/watch?v=3Yfj85rlUml zuletzt aufgerufen am 03.04.2023.
Vgl.: https://www.mathe-online.at/mathint/fourier/i.html zuletzt aufgerufen am 03.04.2023.

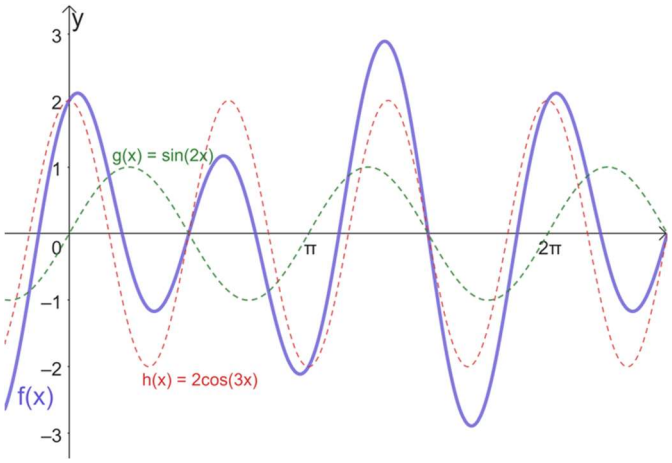


Abbildung 5: Graph zu der Beispielfunktion f(x)

Wenn diese Funktion nun mit einer Basisfunktion multipliziert wird und davon das Integral zwischen 0 und 2π mit $\frac{1}{\pi}$ multipliziert wird, erhält man den jeweiligen Fourier Koeffizienten. Ist dieser 0, ist diese Basisfunktion nicht in der Funktion $f(x)$ enthalten. Wenn der Koeffizient größer ist als 0, ist diese Basisfunktion in $f(x)$ enthalten. Der Koeffizient gibt die Amplitude der jeweiligen Basisfunktion an.

Beispiel zur Berechnung der Fourier Koeffizienten:

Nimmt man die Basisfunktion sin(x), die nicht in der Funktion f(x) enthalten ist, ergibt sich folgendes:

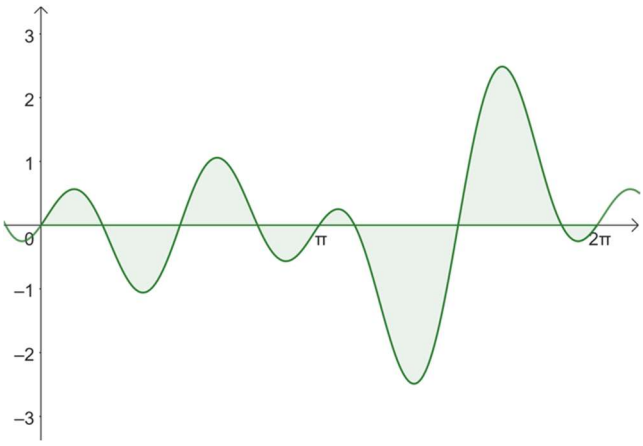


Abbildung 6: Graph von b_1 mit Integral

$$b_1 = \frac{1}{\pi}\int_0^{2\pi} f(x) \cdot \sin(x)\, dx = 0$$

Die positiven und negativen Flächen gleichen sich aus, der Fourier Koeffizient b_1 ist somit gleich 0.

Wird stattdessen die Basisfunktion sin(2x) genommen, die in der Funktion f(x) enthalten ist, ergibt sich folgendes:

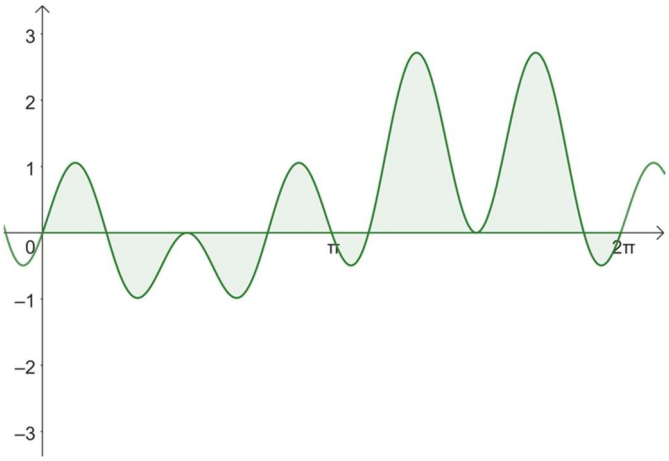


Abbildung 7: Graph von b_2 mit Integral

$$b_2 = \frac{1}{\pi}\int_0^{2\pi} f(x) \cdot \sin(2x)\, dx = 1$$

Die Flächen oberhalb und unterhalb der x-Achse gleichen sich nicht mehr aus, die positive Fläche ist genau um π größer. Daraus ergibt sich der Fourier Koeffizient $b_2 = 1$. Die Basisfunktion sin(2x) ist in der Funktion f(x) mit der Amplitude 1 enthalten. Das gleiche Vorgehen bei der Basisfunktion $\cos(3x)$ ergibt den Fourierkoeffizienten $a_3 = 2$, die Amplitude dieser Basisfunktion ist 2.

Durch die Fourier Analyse erhält man das Spektrum von einer periodischen Funktion. In diesem Spektrum kann man sehen, welche Frequenzen, wie stark in der Funktion vorhanden sind (*siehe 3.4.1 Amplitudenspektrum*).

Es gibt auch eine komplexe Form der Fourier Reihe. Sie enthält anstatt der trigonometrischen Funktionen Sinus und Kosinus die komplexe Exponentialfunktion e. Die komplexe Form wird gebildet, indem die Sinus und Kosinus Funktionen mittels der komplexen e-Funktion ersetzt werden.

Das funktioniert mit der imaginären Einheit i:

$$i = \sqrt{-1}$$

Und der Eulerschen Formel:

$$e^{ix} = \cos(x) + i\sin(x)$$

Damit gilt:

$$\sin(nx) = \frac{1}{2i}\left(e^{int} - e^{-int}\right)$$

$$\cos(nx) = \frac{1}{2}\left(e^{int} + e^{-int}\right)$$

Die komplexe Form der Fourier Reihe für Funktionen mit der Periode p=2π lautet dadurch:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n\, e^{int}$$

Der Vorteil der komplexen Form ist, dass die e-Funktion leichter integriert werden kann und anstatt zwei, nur noch ein Koeffizient, c_n, berechnet werden muss. Der Koeffizient c_n kann wie folgt bestimmt werden:

$$c_n = \frac{1}{2\pi}\int_0^{2\pi} f(t)e^{-int}\,dt$$

Von der reellen zu der komplexen Form kommt man mit den folgenden Formeln:

$$c_0 = \frac{a_0}{2} \qquad c_n = \frac{1}{2}(a_n - ib_n) \qquad c_{-n} = \frac{1}{2}(a_n + ib_n)$$

Und von der komplexen zur reellen Form:

$$a_0 = 2c_0 \qquad a_n = c_n + c_{-n} \qquad b_n = i(c_n - c_{-n})$$

Damit können nur periodische Funktionen zerlegt werden. Zum Zerlegen von nicht periodischen Funktionen wird die Kontinuierliche Fourier-Transformation genutzt, diese wird in dem folgenden Abschnitt erläutert.

3.2.2 Kontinuierliche Fourier-Transformation[15]

Mithilfe der kontinuierlichen Fourier-Transformation kann ein nicht periodisches Signal in seine Frequenzkomponenten zerlegt werden. Es kann damit ein begrenztes Signal, dass nicht, wie bei der Fourier Reihe periodisch weitergeht, analysiert werden. Es werden, wie bei der Fourier Analyse der Fourier Reihe, die Frequenzen mit der dazugehörigen Amplitude berechnet. Diese Frequenzen können in einem Frequenzspektrum (*siehe 3.4.1 Amplitudenspektrum*) dargestellt werden. Die daraus gewonnenen Daten können genutzt werden, um mit verschiedenen Filtern und Effekten die Musik visuell darzustellen.

Die Fourier-Transformation wird üblicherweise in komplexer Form aufgeschrieben und lautet:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}\,dt \qquad \omega = 2\pi f$$

Laut der Eulerschen Formel gilt:

$$e^{-i\omega} = \cos(\omega t) - i\sin(\omega t)$$

Dann lautet die Fourier-Transformation:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)(\cos(\omega t) - i\sin(\omega t))dt$$

$$X(\omega) = \int_{-\infty}^{\infty} x(t)\cos(\omega t)\,dt - i\int_{-\infty}^{\infty} x(t)\sin(\omega t)\,dt$$

Bei der Fourier-Transformation erhält man, im Gegensatz zur Fourier Analyse der Fourier Reihe, ein kontinuierliches Spektrum. Das bedeutet, dass man für jede Frequenz herausfinden kann, wie stark sie in dem Signal vorhanden ist.

Aber unser digitales Audiosignal ist nicht kontinuierlich, sondern diskret. Wir haben also nur Samples (*siehe Kapitel 2.1.1 Abtastung*), um diskrete Signale zu zerlegen, gibt es die Diskrete Fourier-Transformation.

[15] Vgl.: Schmitz, E., *Technik der Fourier-Transformation*.

3.2.3 Diskrete Fourier-Transformation (DFT)

Die Diskrete Fourier-Transformation kann ein Signal zerlegen, bei dem die Zeit und der Wert diskret, das bedeutet nur zu bestimmten Zeitpunkten definiert, sind. Es muss also kein kontinuierliches Signal mehr sein, wie es bei der Kontinuierlichen Fourier-Transformation der Fall war. Dabei verarbeitet die DFT eine Folge von Zahlen, also die diskreten Messwerte. Diese Messwerte entsprechen einer Periode des Signals.

Dazu wird das Integral durch die Summe ersetzt. Die Diskrete Fourier-Transformation lautet dann:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i\omega n} \qquad \omega = 2\pi \frac{k}{N} \quad k = 0,1,\dots,N-1$$

N ist die Anzahl der Samples.

Man summiert die Werte auf, um eine Integrierung zu nähern. Man erhält durch die Berechnung der DFT die Fourier Koeffizienten $X(k)$. Durch diese Fourierkoeffizienten erfährt man, wie viel von der jeweiligen Frequenz in dem Signal vorhanden ist.

Um wieder zurück in den Zeitbereich zu kommen, benötigt man die inverse DFT:

$$x(k) = \frac{1}{N} \sum_{n=0}^{N-1} X(n) \cdot e^{-i\omega} \qquad \omega = 2\pi \frac{n}{N} \qquad k = 0,1,\dots,N-1$$

Um die DFT für $n = 0,1,2,\dots N-1$ zu berechnen, kann folgendes Gleichungssystem geschrieben werden:

$$\begin{pmatrix} X(0) \\ X(1) \\ X(2) \\ \vdots \\ X(N-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N^1 & W_N^2 & \cdots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)^2} \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{pmatrix}$$

Abbildung 8: Gleichungssystem zur Implementierung der DFT

Der mittlere Teil ist die DFT Matrix. Das bedeutet, wenn man die Daten (rechter Vektor) mit der DFT Matrix multipliziert, erhält man als Ergebnis die Fourier-Koeffizienten (linker Vektor).[16]

Diese Matrix Multiplikation ist sehr rechenaufwendig. Deshalb benutzt man einen deutlich effizienteren Algorithmus, die Schnelle Fourier-Transformation (engl. fast Fourier transform (FFT)).

3.2.4 Fast Fourier-Transformation (FFT)

Zur Berechnung der Diskreten Fourier-Transformation kann die Fast Fourier-Transformation genutzt werden. Die FFT ist ein effizienter Algorithmus zur Berechnung der DFT. Der Algorithmus, der 1965 von J. W. Cooley und J. W. Turkey in einem Paper veröffentlicht wurde[17], hat vieles verändert. Es war nun möglich, gewöhnliche Erdbeben von Explosionen zu

[16] *Abbildung 8:* Nelles, 2019., *Vorlesung Signalverarbeitung, Kapitel 3.1, S. 111.*

[17] Vgl.: https://faculty.washington.edu/seattle/physics541/%202010-Fourier-transforms/history-3.pdf zuletzt aufgerufen am 20.05.2023.

unterscheiden. Es konnte somit erkannt werden, wenn unterirdischen Atomwaffentests durchgeführt wurden. Dadurch konnte kein Land mehr unbemerkt Atomwaffen testen[18].

Die FFT benötigt wesentlich weniger Rechenschritte als die Diskrete Fourier-Transformation. Während die DFT N^2 Rechenschritte benötigt, benötigt die FFT nur $N \cdot \log_2(N)$ Rechenschritte. N ist dabei die Anzahl der Samples.

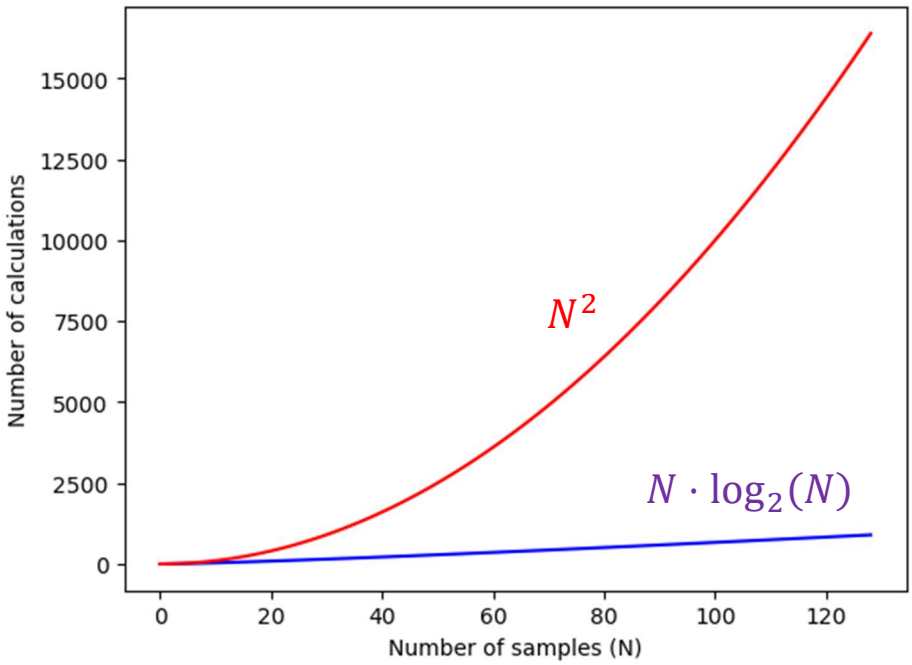


Abbildung 9: Benötigte Berechnungen Vergleich DFT und FFT

Bei 1024 Samples würde die DFT 1.048.576 Rechenschritte benötigen und die FFT nur 10.240. Dieser Unterschied wird immer größer, desto mehr Samples vorhanden sind (*siehe Abbildung 9*).

Deswegen ist es deutlich effizienter die FFT zu benutzen, als die Diskrete Fourier-Transformation direkt zu implementieren. Aus diesem Grund ist sie auch einer der wichtigsten Algorithmen der jemals entwickelt wurde. Sie ist in der Telekommunikation, Signalanalyse und in vielen anderen Bereichen nicht wegzudenken. Ohne die FFT wäre vieles davon nicht so möglich, wie es heute ist. Aus diesen Gründen wird, seit es den Algorithmus gibt, die DFT nicht mehr direkt implementiert, sondern es wird hauptsächlich die FFT benutzt.

Die Idee der FFT ist, dass man die DFT der Größe N in 2 DFTs, mit der Größe N/2, aufteilt. Das funktioniert nur dann, wenn N eine Potenz von 2 ist. Ist dies nicht der Fall, wird das Signal $x(n)$ so lange mit Nullen aufgefüllt, bis die Anzahl der Werte eine Potenz von 2 ist, das Auffüllen mit Nullen wird „zero padding“ genannt. Danach werden die 2 DFTs wieder geteilt in 4 DFTs der Größe N/4, das wird so lange wiederholt, bis $N/2^s = 1$ erreicht ist. S ist dabei die Anzahl der Aufteilungen.[19] Das führt dazu, dass die oben genannten $N \cdot \log_2(N)$ benötigten Berechnungen erreicht werden und die FFT deutlich effizienter berechnet werden kann.

Aus diesen Gründen benutzen wir in unserem Programm die FFT zur Zerlegung von Audiosignalen.

[18] Vgl.: https://www.spektrum.de/kolumne/fast-fourier-transformation-ein-algorithmus-fuer-den-weltfrieden/2078001 zuletzt aufgerufen am 20.05.2023.
[19] Vgl.: Nelles, 2019., *Vorlesung Signalverarbeitung, Kapitel 3.2, S.112.*

3.3 Leck-Effekt

Der Leck-Effekt beschreibt ein Problem bei der Berechnung des Frequenzspektrums mit der diskreten Fourier-Transformation. Die Spektraldarstellung zeigt bei dieser Frequenzanteile an, welche in dem Signal gar nicht enthalten sind.

3.3.1 Entstehung

Das Audiosignal wird, wie bereits in *Kapitel 3.2* erwähnt, in Frames aufgeteilt, welche immer einen bestimmten Zeitbereich des Audiosignals darstellen. Die Abtrennung der einzelnen Frames wird durch die Anzahl an Samples bestimmt und nicht durch eine Periodendauer. Eine Frequenz kann daher inmitten einer Periode abgetrennt sein. Zudem ist Musik grundlegend nicht periodisch. Frequenzen können in einem Frame beginnen und inmitten einer Periode abrupt wieder enden, ohne dass die Periode vollständig war. (*Siehe Abbildung 10*)

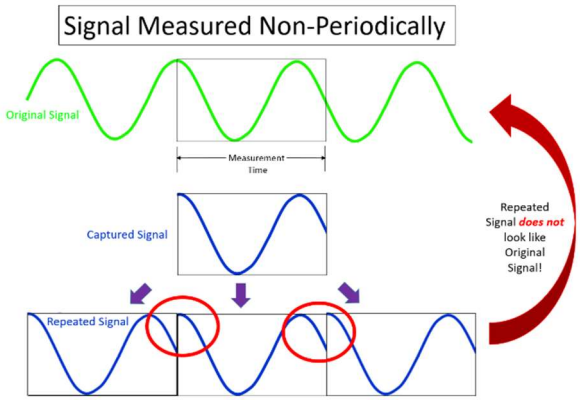
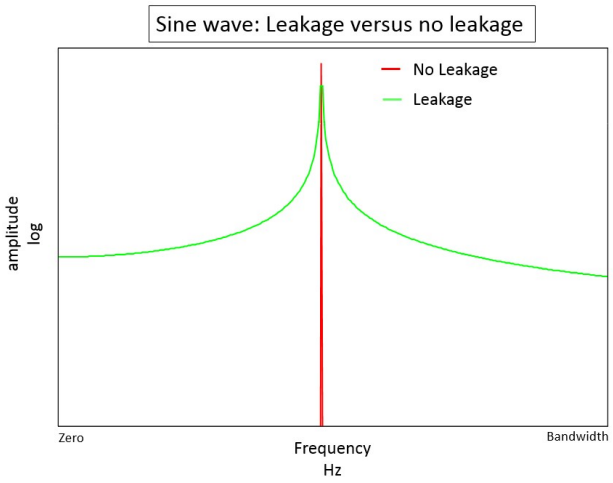


Abbildung 10: Beispiel eines nichtperiodischen Signals

Dies Widerspricht jedoch dem eigentlichen Verständnis der Fourier Analyse. Ein Signal muss periodisch sein. Bei nicht periodischen Signalen, wie bei der Fourier-Transformation, wird daher die Periodendauer auf unendlich gesetzt, um es wieder periodisch zu machen. Eine zeitlich beschränkte Analyse widerspricht dem.

Es wird davon ausgegangen, dass das Signal in *Abbildung 10* auch periodisch ist, was es nicht ist, und an derselben Stelle aufhört, wo es begonnen hat. Die Fourier-Transformation versucht diesen Bereich zwischen periodischem Ende und dem eigentlichen Ende, durch weitere



Frequenzen im Spektrum wieder korrekt darzustellen. Diese Frequenzen stellen den Leck-Effekt dar.[20]

3.3.2 Behebung

Um den Leck-Effekt zu reduzieren wird eine Fensterfunktion mit dem Signal multipliziert. Ein Fenster stellt eine Funktion dar, welche am Anfang genauso wie am Ende 0 werden. Hierdurch werden die einzelnen Frames künstlich periodisiert.

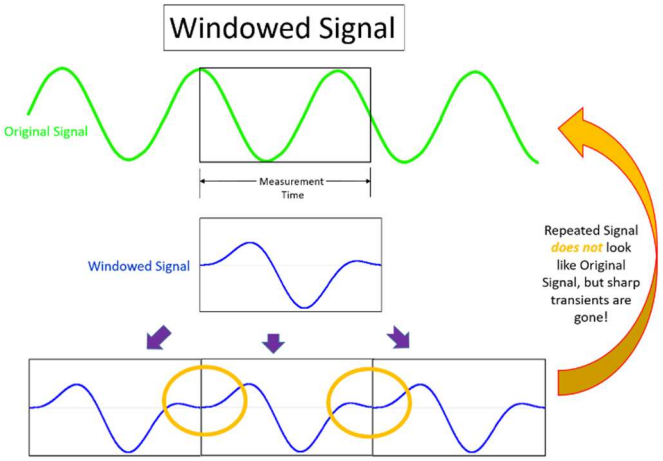


Abbildung 12: Beispiel einer Fensterung mit dem Hanning-Window

Eine vollständige Behebung des Leck-Effekts ist nicht möglich, da auch das Signal mit einem Fenster einen restlichen Fehler aufweist. Zudem sind Musiksignale, wie bereits erwähnt, nicht periodisch, weswegen es immer ein Restfehler gibt, der sich jedoch in Grenzen hält.

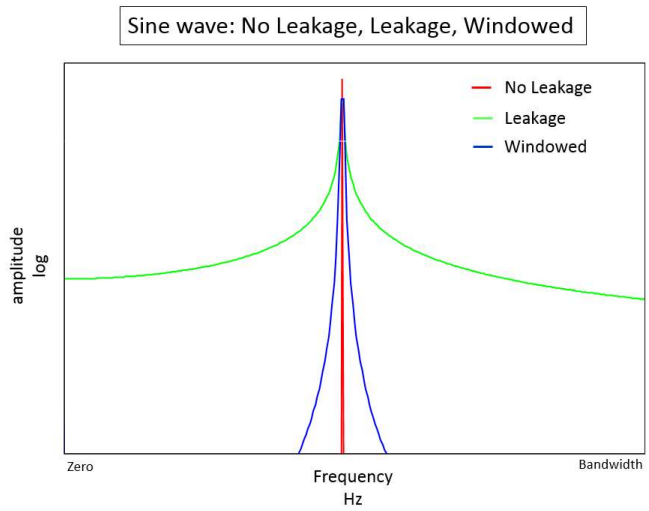


Abbildung 13: Spektrum eines gefensterten Signals

[20] Vgl.: Kumar, 2018, *Lecutre 4 | Discrete Fourier Transform (DFT) with it`s leakage.*

3.3.3 Überlappung

Durch die Fensterung werden jedoch Samples zwischen den einzelnen Frames nicht wahrgenommen, da diese durch das Fenster minimiert werden. Es kommt zu einem Datenverlust. Aus diesem Grund werden, bei der Berechnung der diskreten Fourier Transformation, weiterhin Samples des letzten Frames beibehalten und mit den Samples des momentanen Frames transformiert. Die einzelnen Frames werden hierdurch sozusagen „überlappt“[21].

[21] Vgl.: Valerio Velardo, 2020, *Audio Signal Processing for Machine Learning: Extract Audio Features.*
Vgl.: Siemens, 2019*, Windows and Spectral Leakage.*
Abbildung 4, 5, 6, 7 s.: Siemens, 2019, *Windows and Spectral Leakage.*

3.4 Spektrum

3.4.1 Amplitudenspektrum[22]

Das Amplitudenspektrum ist die Darstellung der Leistung der einzelnen Frequenzen. Es zeigt welche Frequenzen in dem Signal enthalten sind und wie stark diese jeweils sind. Es ist der Betrag des Frequenzspektrums X. Man erhält die Werte für das Amplitudenspektrum durch die Berechnung der Fourier-Transformation (*siehe Kapitel 3.2*).

$$A(f) = |X(f)|$$

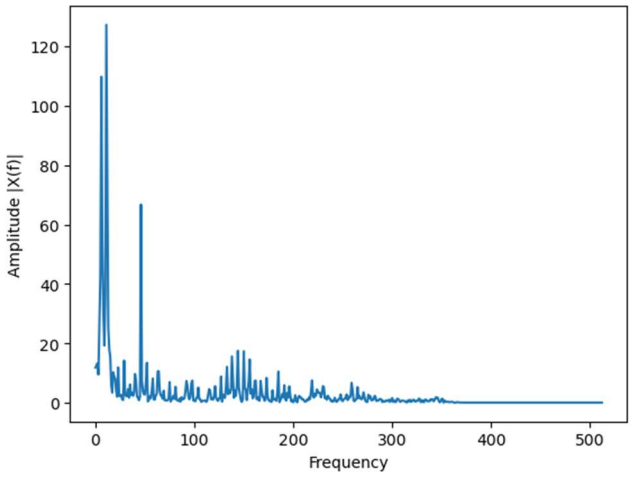


Abbildung 15: Beispiel Amplitudenspektrum

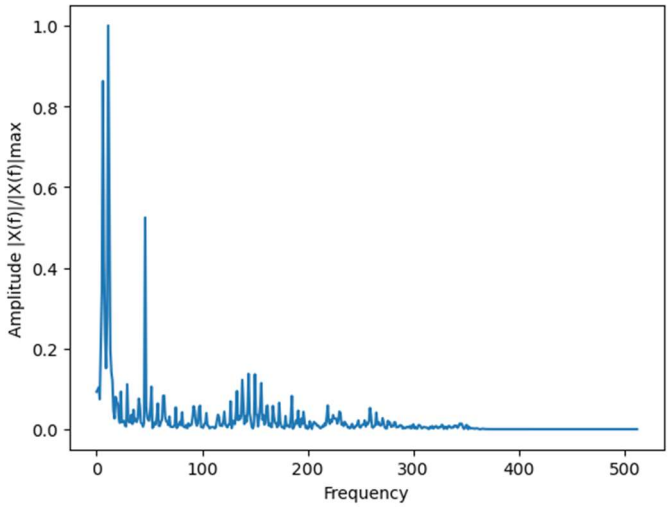


Abbildung 14 : Amplitudenspektrum (Verhältnis)

Auf der x-Achse sind alle Frequenzen der Messung abgebildet und auf der y-Achse die Amplitude, also der Wert, der bei der Fourier-Transformation berechnet wird. In diesem Beispiel eines Amplitudenspektrums kann man erkennen, dass in dem ursprünglichen Signal hauptsächlich tiefe Frequenzen vorhanden sind, da die Amplituden bei Werten kleiner als 200 Hz deutlich größer sind als die Amplituden größer als 200 Hz.

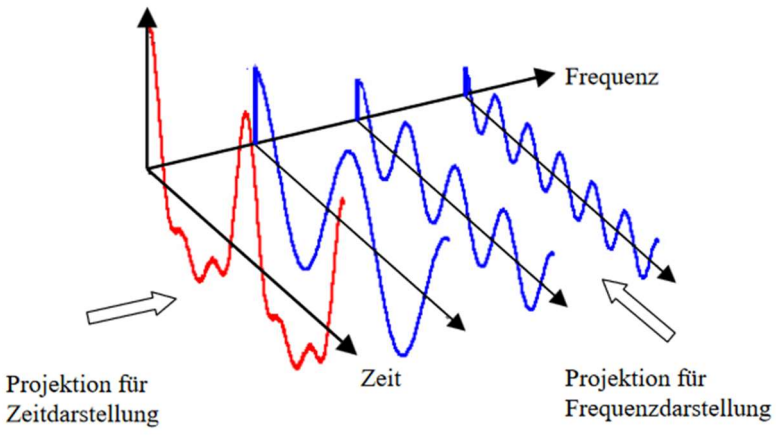


Abbildung 16: Darstellung der Projektionen

Man kann sich das so vorstellen, dass man die einzelnen Frequenzen aus dem ursprünglichen Signal „herauszieht“ und dann von der Seite darauf schaut (*siehe Abbildung 16*). Dadurch sieht man für jede Frequenz die dazugehörige Amplitude, also wie stark diese in dem Signal vorhanden ist.

[22] Vgl.: Maier, Spektrum Fouriertransformation.
Abbildung 16: Maier, Spektrum Fouriertransformation, Seite 1.

Das Amplitudenspektrum ist nützlich, um darzustellen, welche Frequenzen, wie stark in dem vorliegenden Signal vorhanden sind. So können zum Beispiel ungewollte Frequenzen gefunden werden.

3.4.2 Leistungsspektrum

Das Leistungsspektrum (*engl. Power Spectrum*) ist neben dem Amplitudenspektrum X eine weitere Darstellungsform der verschiedenen Frequenzen, welche die Leistung der einzelnen Frequenzen darstellt. Das Leistungsspektrum PS kann durch die Quadrierung der Betragswerte des Amplitudenspektrums berechnet werden.

$$PS = |X|^2$$

Der Satz von Plancherel besagt, dass die Leistung in der Zeitdarstellung ebenso der Leistung in der Frequenzdarstellung entspricht. Dies bedeutet, dass sowohl die Energie eines Frames im Zeitbereich wie auch im Frequenzbereich, die gleiche ist.[23]

$$\sum_{n=1}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=1}^{N-1} |X[k]|^2$$

Das Leistungsspektrum wird hierzu in *Kapitel 4.1 Frequenzdarstellung* wiederverwendet.

[23] Vgl.: Beckhoff, o.D., *Skalierung von Spektren.*
Vgl.: Siemens, 2020, The Autopower Function…Demystified!
S. Wikipedia, Satz von Plancherel.

3.5 Normalisierung & Mel Filter Bank

3.5.1 Mel[24]

Wir Menschen können den Unterschied zwischen 300 Hz und 400 Hz deutlich hören. Allerdings ist dieser Unterschied bei 2000 Hz und 2100 Hz kaum wahrnehmbar, obwohl die Differenz eigentlich gleich ist. Das liegt daran, dass wir die Tonhöhe logarithmisch wahrnehmen. Das bewirkt, dass wir tiefe Frequenzen besser wahrnehmen als hohe Frequenzen. Da das auch die Darstellung der Musik beeinflusst, mussten wir einen Weg finden, wie wir das ändern können.

Die Lösung dafür ist die Mel-Skala:

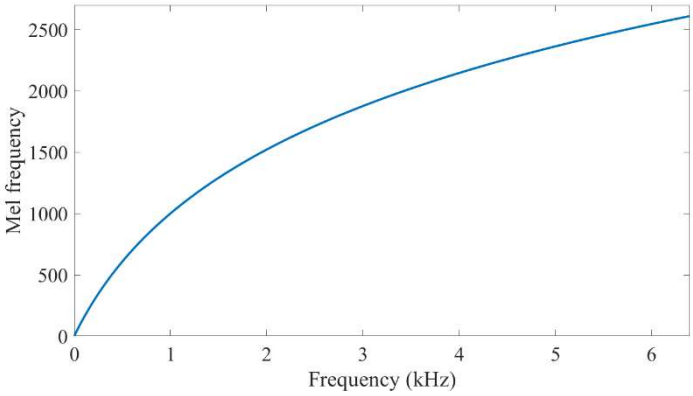


Abbildung 17: Mel-Skala[25]

Die Toneinheit Mel wurde durch Hörversuche bestimmt und beschreibt dadurch die wahrgenommenen Tonhöhen. „Ein Ton, der doppelt so hoch wahrgenommen wird, erhält den doppelten Tonheitswert, ein Ton, der als halb so hoch wahrgenommen wird, den halben Tonheitswert."[26] Der Zusammenhang von Hertz und Mel ist logarithmisch, wie man auch an dem oberen Graph (*Abbildung 17*) erkennen kann.

Die Formel zur Umrechnung von Hertz (f) zu Mel (Z) lautet:

$$Z = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right)$$

Die Formel zur Umrechnung von Mel (Z) zu Hertz (f) lautet:

$$f = 700\left(10^{\frac{Z}{2595}} - 1\right)$$

Die Toneinheit Mel kann nun genutzt werden die Mel Filter Bank zu berechnen.

[24] Vgl.: Valerio Velardo, 2020, *Audio Signal Processing for Machine Learning: Mel Spectrograms Explained Easily.*
[25] Abbildung 6: s.: Bäckström, T., 2019, Cepstrum and MFCC. Mel-Frequency Cepstral Coefficients (MFCCs).
[26] Zit. aus: https://de-academic.com/dic.nsf/dewiki/941108, Abschnitt: Definitionen der Tonheit in Mel, zuletzt aufgerufen am 22.05.2023.

3.5.2 Mel Filter Bank[27]

Da nun klar ist was die Toneinheit Mel ist, schauen wir uns an, wie die Mel Filter Bank funktioniert. Mithilfe dieser Filter Bank können wir das Signal an den menschlichen Hörsinn anpassen. Dadurch können wir die Musik so visualisieren, wie wir Menschen sie auch wahrnehmen

Die Mel Filter Bank ist eine Filter Bank mit dreieckig geformten Filtern (Bins) auf einer Mel-Skala. Um diese zu erstellen, benötigt man die Anzahl der Filter, die maximale Frequenz und die minimale Frequenz auf denen die Filter Bank angewendet werden soll. Die Anzahl der Filter kann selbst bestimmt werden, oft werden 40 Filter benutzt. Die Amplitude gibt an wie viel von der jeweiligen Frequenz durch den Filter gefiltert wird. Bei der Spitze jedes Filters kommt also 100% der Frequenz durch den Filter und an den unteren Ecken jeweils 0%. Die Filter werden bis zur maximalen Frequenz immer größer. Dadurch werden die tieferen Frequenzen mehr gefiltert als die höheren Frequenzen.

Als Beispiel bei der unteren Grafik, kommt bei circa 2000Hz die Stärke dieser Frequenz zu circa 80% durch, die Stärke wird also mit 0,8 multipliziert und kommt in den 29. Filter (blau). Das wird jetzt für jede Frequenz gemacht, dadurch entstehen die verschiedenen Filter. Bei der vorliegenden Grafik sind es 40 Filter.

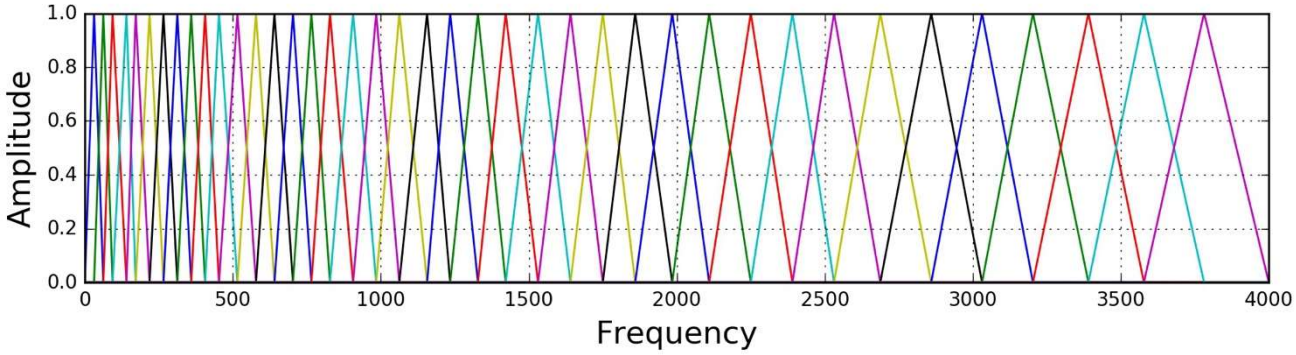


Abbildung 18: Filter-Bank on a Mel-Scale

Um die Filter Bank anzuwenden, muss daraus eine Matrix erstellt werden, die Mel-Matrix (*siehe Abbildung 19*). Jeder Filter, also jedes Dreieck, wird in einer Zeile der Matrix eingefügt. Dazu wird die obige Berechnung für jede Frequenz durchgeführt. Bei der unteren Grafik gibt es also 6 Filter. Es ist zu sehen, dass die Filter wieder bei tieferen Frequenzen deutlich dünner sind als die bei höheren Frequenzen.

[27] Vgl. Gündert, S., 2023, Mel Filter Bank.
Vgl. https://www.youtube.com/watch?v=MoIRG27jD54 zuletzt Aufgerufen am 14.01.2023.
Vgl. Bäckström, T., 2019, Cepstrum and MFCC. Mel-Frequency Cepstral Coefficients (MFCCs).
Vgl. Fayek, H., 2016, Speech Processing for Machine Learning, Filter Banks.
Abbildung 18: Fayek, H., 2016, Speech Processing for Machine Learning, Filter Banks.

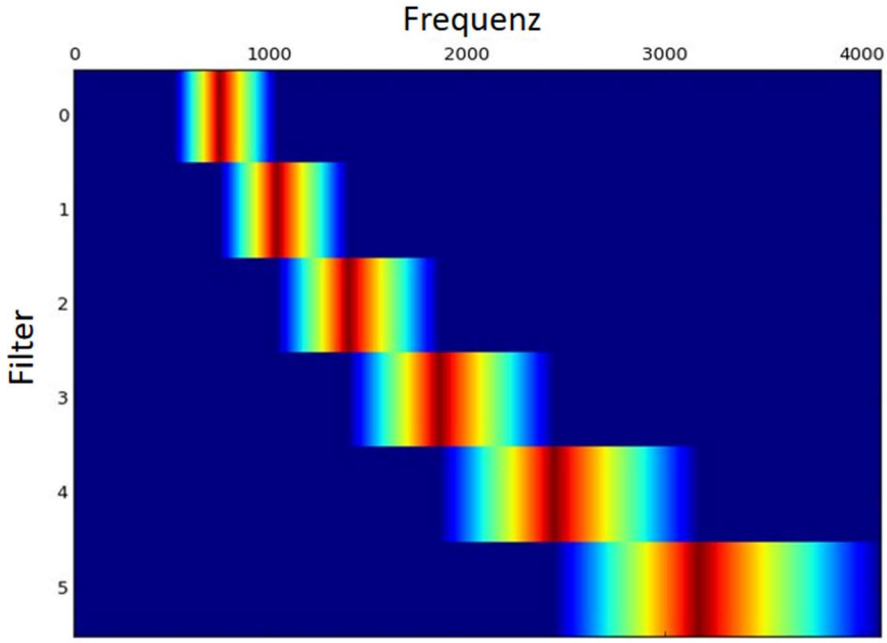


Abbildung 19: Mel-Matrix[28]

Die Mel-Matrix kann jetzt mit dem Frequenz-Spektrum, das mithilfe der Fast-Fourier-Transformation erstellt wurde, verrechnet werden. Das ergibt das Mel-Spektrum.

Da wir auch die Lautstärke logarithmisch wahrnehmen müssen wir das Spektrum jetzt noch in die logarithmische Form umrechnen. Dazu wird der dekadische Logarithmus verwendet. Damit ist das Signal an den menschlichen Hörsinn angepasst und wir können das Signal zur weiteren Verarbeitung nutzen.

[28] Abbildung 8: s.: Gündert, S,. 2023, Mel Filter Bank.

3.6 Quadratisches Mittel

Um den Mittelwert eines Signals zu berechnen kann das quadratische Mittel (oder der quadratische Mittelwert (QMW), engl. root mean square (RMS)) genutzt werden. Der QMW ist ein statisches Maß und wird mit der folgenden Formel berechnet:

$$QMW = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_n^2}{n}}$$

Es werden zuerst alle quadrierten Werte x_i addiert und, danach durch die Anzahl der Werte n geteilt. Die Quadratwurzel davon ergibt den QMW. Dadurch, dass die Werte quadriert werden, werden die größeren Werte mehr gewichtet als die kleineren Werte, da die quadrierten größeren Werte einen größeren Beitrag zur Gesamtsumme beitragen, als die kleineren Werte. Durch das Quadrieren werden außerdem alle negativen Werte positiv[29].

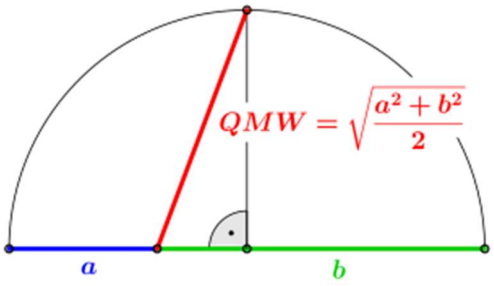


Abbildung 20: Quadratisches Mittel

Um den QMW von dem Frame t zu berechnen kann die folgende Schreibweise der Formel genutzt werden:

$$QMW_t = \sqrt{\frac{1}{K} \cdot \sum_{k=t\cdot K}^{(t+1)\cdot K-1} s(k)^2}$$

Hier gilt das gleiche Prinzip wie oben. Die Amplitude des Samples k von dem Frame, $s(k)$, wird quadriert, das entspricht der Energie von diesem Sample k. Es wird die Summe der Energie aller Samples in Frame t gebildet und davon das Mittel berechnet. Die Quadratwurzel, davon ergibt den QMW des Frames t. Der QMW ist nicht empfindlich zu Ausreisern, da alle Werte in dem Frame berücksichtigt werden[30]. Wir können den QMW nutzen, wenn wir die Lautstärke des aktuellen Frames bestimmen wollen.

[29] Vgl.: https://www.cosmos-indirekt.de//Physik-Schule/Quadratisches_Mittel zuletzt aufgerufen am 28.05.2023.

[30] Vgl.: Willig, H.-P., Quadratisches Mittel; Valerio Velardo, 2020, *Audio Signal Processing for Machine Learning: Understanding Time Domain Audio Features.*

Abbildung 20: https://upload.wikimedia.org/wikipedia/commons/thumb/e/e7/01_Quadratisches_Mittel.svg/330px-01_Quadratisches_Mittel.svg.png.

3.7 Exponentielle Filter

3.7.1 Theorie[31]

Exponentielle Filter sind eine einfache und effektive Methode, um Audiosignale zu glätten. Das Signal hat so einen geglätteten Übergang und wechselt nicht abrupt, wenn die nächsten Werte deutlich größer oder kleiner sind. Das ist wichtig, um das Signal visuell darzustellen, da abrupte Änderungen als nervig, unschön oder “laggy” wahrgenommen werden können. Dieser Filter ist dadurch essenziell, um Musik schön und gleichmäßig visuell darzustellen.

Es funktioniert mithilfe einer rekursiven Formel, die damit eine Vorhersage (engl. forecast) für den nächsten Wert berechnet. Die Formel zu der Berechnung der Vorhersage lautet:

$$y(t) = \alpha \cdot x(t) + (1 - \alpha) \cdot y(t - 1)$$

Dabei ist $y(t)$ die berechnete neue Vorhersage für den nächsten Wert. Alpha α ist der Glättungsfaktor, er gibt die Gewichtung an, wie stark der neue Messwert mit einberechnet werden soll. Umso größer Alpha ist, umso mehr Gewichtung hat der neue Messwert auf die Vorhersage. Ist Alpha gleich 1, ist die Vorhersage jeweils der neue Messwert, es wird nicht geglättet. Ist Alpha gleich 0, bleibt die Vorhersage unverändert. Dabei muss durch Testen individuell der richtige Wert gefunden werden, um das optimale Ergebnis zu erhalten.

Es wird der neue Messwert $x(t)$ mit Alpha α multipliziert und der vorherige Wert $y(t - 1)$ wird mit $(1 - \alpha)$ multipliziert, die Summe davon bildet die neue Vorhersage und somit den neuen geglätteten Wert.

3.7.2 Anwendung

Wir haben den Filter mit zwei verschiedenen Glättungsfaktoren implementiert. Ein Faktor ist für Werte, die größer sind als die letzte Vorhersage und der andere für Werte kleiner als die letzte Vorhersage. Dadurch lässt sich das Verhalten deutlich modularer anpassen. Es lässt sich so einstellen, ob der Wert schneller steigen oder abnehmen soll, was in unserem Programm mehr Möglichkeiten bietet, das Signal zu verändern und anzupassen. Zum Beispiel kann so ein Effekt erzeugt werden, bei dem es bei einem Schlag schnell aufleuchtet, dann aber nur langsam wieder abflacht.

Zudem nutzen wir diesen Filter für eine Min-Max Normalisierung. Das Programm weiß nicht direkt, wie laut etwas ist, deswegen wird der lauteste Wert als Referenz verwendet. Dieser Wert wird mit einem Exponentialfilter geglättet, dass keine zu abrupten Änderungen entstehen. Der Frame wird dann durch diesen Gleitfaktor geteilt, um so einen normalisierten Wert zwischen 0 und 1 zu erhalten. Durch das Glätten des Maximums sind so weiterhin Unterschiede in der Lautstärke erkennbar.

Wir haben außerdem jeweils eine Funktion zur Glättung eines einzelnen Wertes und für die Glättung eines ganzen Arrays. Dadurch kann nur ein einzelner Wert geglättet werden, zum Beispiel der QMW (*Kapitel 3.6*), oder das gesamte Signal über die Zeit.

[31] Vgl.: Uni Kassel, *Exponentielle Glättung.*

4. Programmaufbau

Dieses Kapitel beschäftigt sich mit dem Aufbau und der Programmierung eines Programms für das Testen und Anwenden des theoretischen Wissens.

4.1 Plattform

Zur Programmierung der Anwendung verwenden wir die Programmiersprache Python. Python hat im Gegensatz zu anderen Programmiersprachen eine sehr einfache und dynamische Syntax, was sie sehr beliebt für die Anwendung im Bereich der Datenanalyse oder des wissenschaftlichen Arbeitens in jeglicher Form macht. Weswegen auch wir sie für dieses Projekt einsetzen. Dennoch ist Python eine Sprache, welche durch die Verwendung eines Interpreters, anstatt eines Compilers, recht langsam ist.

Daher verwenden wir für jegliche mathematischen Berechnungen die Bibliotheken numpy und scipy. Deren Algorithmen sind in der Sprache C geschrieben und daher deutlich schneller in der Anwendung im Gegensatz zu den standardmäßigen Python Befehlen. Wichtig ist uns hierbei ausschließlich mathematische Bibliotheken zu verwenden und nicht auf weitere Hilfsbibliotheken zur Audioanalyse zurückzugreifen, da diese, wie bereits erwähnt, selbst geschrieben werden sollen.

Für die graphische Darstellung wird außerdem die Bibliothek pyqtgraph verwendet. pyqtgraph ermöglicht eine Echtzeit-Darstellung der analysierten und berechneten Daten. Wir verwenden pyqtgraph für das Testen jeglicher Effekte sowie Algorithmen.

4.1.1 Einsatzgebiete

Das Programm soll außerdem an so vielen Geräten wie möglich betrieben werden können. Dank der Programmiersprache Python ist es möglich sowohl Windows, Linux und MacOS als auch einen RaspberryPi zu unterstützen.

4.1.2 Testgerät

Das Signal wird schließlich an einem LED-Stripe visualisiert. Hierzu haben wir den LED-Stripe mit einem ESP32-Mikrocontroller auf einer Platine verkabelt. Der ESP32 verbindet sich mit dem WLAN und wartet auf ein Signal des Hauptprogramms. In *Kapitel 4.3* wird näher auf die Übertragung eingegangen.

4.2 Audioeingang

Um ein Audiosignal visualisieren zu können wird eins benötigt. Zur Wiedergabe von Musik gibt es heutzutage hauptsächlich zwei Methoden, welche ich kurz näher erläutern werde.

Eine Möglichkeit ist die Anwendung eines gespeicherten Signals in Form einer Audiodatei auf einem Speichermedium. Formate hierfür sind WAV, MP3 oder AAC. Eine weitere Möglichkeit, welche in den letzten Jahren stark an Popularität bekommen hat, ist das direkte „streamen“ aus dem Internet. Große Anbieter in diesem Bereich sind zum Beispiel Spotify, iTunes oder YouTube Music. Streaming-Anbieter haben sich schnell verbreitet, da Musik nicht zuvor heruntergeladen werden und manuell gespeichert werden muss. Der eigentliche Prozess vom Hören ist wesentlich einfacher und flexibler geworden.

Dasselbe gilt es auch für unsere Anwendung zu ermöglichen. Statt einer manuellen Eingabe der Audiodatei durch den Benutzer, haben wir uns für das „streamen“ des Audioausgangs am PC geeinigt. Das bedeutet, dass die Anwendung jegliche interne Musik auf dem Computer wahrnimmt und verarbeitet. Hierunter fallen auch dann interne Programme wie Spotify und so weiter.

Um nun den Audioausgang eines Computers „streamen“ zu können, muss dieser als Audioeingang wieder deklariert werden. Für diese Aufgabe gibt es verschiedene Programme, welche dies umsetzen. Bei Windows gibt es standardmäßig das Eingangsgerät „Stereomix“, welches hierfür gedacht ist. Stereomix ist ein Feature von dem Soundkarten-Hersteller Realtek und kann in allen Computern mit Realtek-Soundkarten verwendet werden.[32]

4.2.1 PyAudio

Für das Einlesen des Audiosignals in Python nutzen wir PyAudio[33]. PyAudio ist ein Wrapper für die C-Bibliothek PortAudio, welche das Streamen von verschiedenen Eingangsgeräten auf den Plattformen Windows, Linux als auch MacOS ermöglicht.

Um einen Stream zu öffnen, werden folgende Parameter benötigt:

1.format
Unter dem Format ist die digitale Darstellung nach der Quantisierung gemeint *(Siehe Kapitel 2.2)*. Wir verwenden hierfür einen 32-Bit Float. Dieser hat ein hohes Quantisierungsniveau und ermöglicht die Rechnung mit Kommazahlen.

2.rate
Unter rate ist die Samplerate gemeint *(Siehe Kapitel 2.1)*. Hier nehmen wir einen Standartwert von 48.000 Hz. Kleiner Unterschiede sind hier aber von keiner Bedeutung. Es kann genauso gut einen Wert von 41.000 Hz genommen werden. Jedoch sollte er nicht zu klein sein, da wie bereits in Kapitel 2.1 erwähnt, sonst Informationen verloren gehen.

3.channels
Das Attribut Channel gibt die Anzahl an verwendeten Tonspuren an. Standardmäßig werden 2 Channels für einen Stereo-Sound bei der Musikwiedergabe verwendet. Für die Visualisierung wird allerdings nur eine Spur gebraucht.

4.frames_per_buffer
Das Attribut frames_per_buffer gibt die Anzahl an Samples in einem Frame an. Jenes Frame wird später im Programm verarbeitet und visualisiert. Der Wert lässt sich mit der Samplerate und der Bildwiederholungsrate pro Sekunde fps berechnen.

$$frames\_per\_buffer = \frac{sample\_rate}{fps}$$

Je höher die Samplerate, desto mehr Samples werden pro Frame gespeichert. Wie bereits erwähnt, können durch mehr Samples mehr Informationen über das eigentliche Audiosignal analysiert werden.

4.2.2 Bildwiederholungsrate

Die Bildwiederholungsrate fps gibt die Anzahl an dargestellten Frames pro Sekunde an. Die Bildwiederholungsrate ist essenziell wichtig, da eine zu niedrige Wiederholungsrate als unschön und „laggy“ empfunden wird. Eine zu große Wiederholungsrate könnte dagegen zu Latenzproblemen bei der Übertragung an den LED-Streifen führen. Bei Monitoren und Bildschirmen wird eine Bildwiederholungsrate von 60 FPS meistens als Optimum genommen. Während dem Testen haben wir als dementsprechend auch circa 50 bis 60 FPS als Bestens empfunden.

[32] S.: Zafar, 2022, *Fix Missing or Disable Realtek Stereo Mix in Windows 11/10.*
[33] S.: o. V, *PyAudio Documentation.*

4.2.2 Callback

Der geöffnete Audiostream wird schließlich in einer Schleife, mit der angegebenen Bildwiederholungsrate, abgelesen. Der momentane Frame wird zwischengespeichert und über ein Callback an das Hauptprogramm, zur weiteren Analyse, übergeben.

4.3 Datenübertragung

Wie in *Kapitel 4.1* bereits erwähnt, soll die Software plattformunabhängig auf verschiedenen Geräten laufen können. Das bedeutet zugleich, dass eine Übertragung der Daten an den LED-Streifen erfolgen muss. Hierfür verwenden wir das Streaming Protokoll E1.31[34]. Das Protokoll wird hauptsächlich für die Vernetzung größerer Bühnentechnik mittels WLAN und LAN benutzt. Auch für kleinere Anwendungen eignet sich E1.31 gut, weshalb ich kurz auf die Vorteile und den Aufbau des Protokolls eingehen werde.

4.3.1 Aufbau

E1.31 basiert auf dem DMX-Protokoll. DMX ist ein Standard, welcher ebenfalls in der Bühnen- und Lichttechnik verwendet wird. Hierfür werden die Geräte mit einem Kabeln verbunden. Der Ansatz von E1.31 ist es hier Kabel zu vermeiden. So wird intern ebenfalls mit DMX-Paketen gearbeitet, welche aber über das Netzwerk geschickt werden. Ein einzelnes DMX-Paket kann $512\ Datenbytes$ transportieren.

Jedes Paket wird ebenfalls mit einem $Universum$ adressiert. Ein $Universum$ kann als eine Art Channel verstanden werden. Ein Empfänger bekommt nur jene Daten, welche auf sein $Universum$ adressiert sind. Dies erlaubt die Übertragung verschiedene Pakete an unterschiedlichen Geräten.

E1.31 unterstützt ebenfalls die Techniken $Unicast$, $Broadcast$ und $Multicast$ für die Datenübertragung. Bei $Unicast$ erfolgt die Übertragung vom Sender zu maximal einem Empfänger. Mehrere Empfänger sind somit nicht möglich. $Broadcast$ kann als Gegenteil verstanden werden und beschreibt die Übertragung an alle Geräte im Netzwerk. Auch an jene Geräte, welche diese Daten nicht verarbeiten können. Eine zu schnelle Netzwerkauslastung ist hier ein entscheidender Nachteil. Bei $Multicast$ erfolgt die Übertragung ebenfalls an mehrere Geräte. Anders als bei $Broadcast$ empfangen bei $Multicast$ nur jene Geräte die Daten, welche sie auch bekommen wollen. Multicast ermöglicht somit die Übertragung verschiedener Datenpakete an mehrere Geräte, bei einer stabilen Netzwerkauslastung. [35]

4.3.2 Anwendung

Auch bei unserer Anwendung setzen wir aus folgenden Gründen auf E1.31 mit einer Multicastübertragung. Für die Übertragung stehen $512\ Bytes$ in einem Universum zu Verfügung. Ein Farbwert besteht aus Rot, Grün, Blau und braucht daher $3\ Bytes$. Theoretisch ist somit eine Übertragung an 170 LEDs in einem Universum möglich. Für die endgültige Übertragung werden die Farbwerte folgend für das Testgerät nacheinander angereiht. (*Siehe Abbildung 21).*

LED 0 Red	LED 0 Green	LED 0 Blue	LED 1 Red	LED 1 Green	LED 1 Blue	LED 2 Red	...

Abbildung 21: Aufbau Sendedaten

[34] S.: ESTA, 2018, *ANSI E1.31.*

[35] S.: Artistic License Integration, o. D., *sACN.*

Für das Testgerät verwenden wir, wie bereits erwähnt einen ESP-32 Mikrocontroller. Auf diesem läuft die Software WLED[36]. WLED kommt mit allen gängigen LED-Stripes zurecht und hat eine stabile E1.31 Unterstützung. Sobald eine Übertragung stattfindet, erkennt WLED dies und verarbeitet die ankommenden Daten.

4.4 Programmstruktur

Das Entwickeln des Programms hat sich am Anfang als recht schwierig erwiesen. Es musste eine recht hohe Flexibilität bieten, um unkompliziert weitere Effekte und Funktionen einbauen zu können. Zum anderen waren einzelne Prozesse stark voneinander abhängig, weswegen sich die Entwicklung einer festen Struktur als Anspruchsvoll erwiesen hat. Während der Entwicklung haben wir außerdem andere Projekte angeschaut, um weitere Ideen zu bekommen und alternative Herangehensweisen auszuprobieren. Hierbei möchte ich die Projekte LedFX und audio-reactive-led-strip nennen, welche uns bei unserem Aufbau inspiriert haben. Diese Projekte hatten allerdings keinen weiteren Einfluss unsern Programmcode und haben uns ausschließlich weitere Konzepte zur Umsetzung unserer Anwendung nähergebracht. Folgend werde ich auf den Aufbau der Effekte eingehen und diesen erklären. Das *Klassendiagramm* hierzu befindet sich auf der nächsten Seite.

4.4.1 Effekte

Die Effekte haben wir grundsätzlich nach dem Prinzip der Vererbung aufgebaut. Es gibt die Hauptklasse *AudioEffect*, von welcher alle Effekte erben müssen. Die Klasse selbst ist allerdings eine abstrakte Klasse und kann nicht direkt als Effekt genutzt werden. Die Klasse ist für die eigentliche Audioanalyse verantwortlich und stellt, für alle Effekte, hierzu Hilfsfunktionen, wie die $power\_spectrum()$ oder $rms()$ Funktion bereit. Sie kann von dem Hauptprogram mit der Funktion $activate(\dots)$ gestartet werden und mit der Funktion $process(\dots)$ das momentane Audiosignal analysieren.

Alle Effekte erstellen standardmäßig eine Visualisierung ohne Farbe. Die Farbe wird anschließend in einem späteren Prozess hinzugefügt. Allerdings, wie bereits erwähnt, gibt es auch hier eine hohe Abhängigkeit bei den Farben zu den Effekten. Für eine direkte Farbgebung in der Effektklasse gibt es daher die *ColorRender* Klasse. *ColorRender* stellt ein Interface dar, welches eine Methode zur Visualisierung mit Farbe bereitstellt. Hierdurch können Effekte selbständig die Farbwerte zu den Effekten anpassen.

4.4.2 Dokumentation

Der Aufbau der Effekte, sowie die bereits erklärten Konzepte wie der Datenübertragung oder dem Audioeingang stellen den Kern des Programms dar. Dennoch enthält das Programm viele weitere „kleinere“ Funktionen. Hierbei handelt es sich auch um einfach angewandte Konzepte aus *Kapitel 3.* Folgend wird noch auf Erstellung und Auswertung unserer Tests sowie der Entwicklung der Effekte eingegangen. Bei weiterem Interesse möchte ich auf unsere GitHub-Repository[37] des Projekts verweisen, bei welchem das ganze Programm inklusive Tests dokumentiert sind.

[36] S.: WLED, o. D., *E1.31 (DMX) / Art-Net.*

[37] S.: Zimmermann F., Dold J., 2023, *AudioVisualizer.*

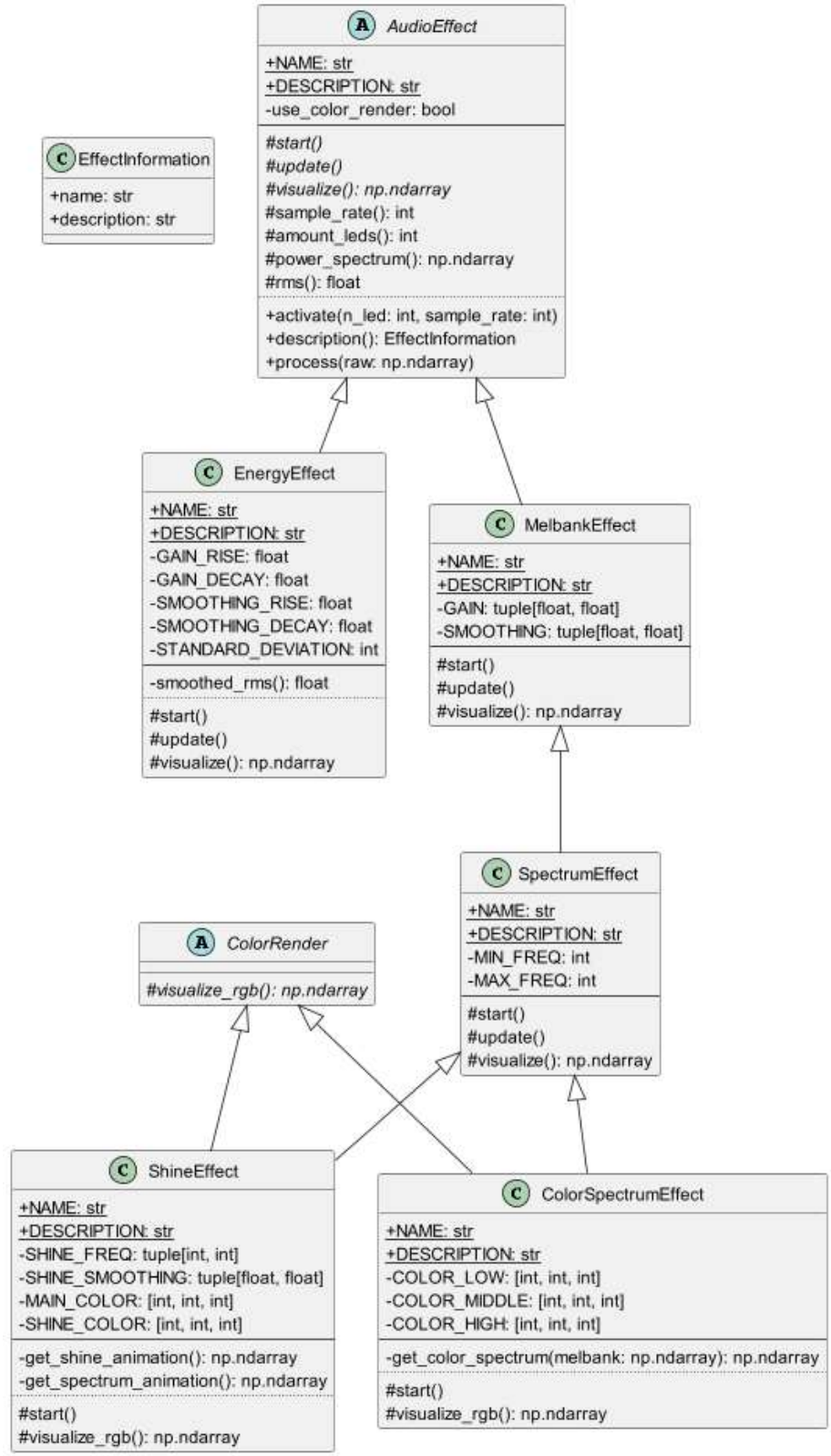


Abbildung 22: Klassendiagramm Effekte

4.4.3 Grafische Benutzeroberfläche

Um das Wechseln der Effekte zu vereinfachen haben wir eine Grafische Benutzeroberfläche (engl. graphical user interface (GUI)) erstellt. Die GUI soll eine Übersicht bieten welche Effekte es gibt und es ermöglichen den gewünschten Effekt einfach wechseln zu können.

Zu dem Erstellen der GUI nutzen wir die Bibliothek CustomTkinter[38]. Sie basiert auf der Python Standard GUI Bibliothek Tkinter. CustomTkinter erleichtert das Erstellen einer schönen und modernen GUI in Python. Die Bibliothek bietet moderne Widgets, die einfach hinzugefügt werden können und auch die Möglichkeit bieten mit normalen Tkinter Elementen zusammen genutzt zu werden.

Die GUI hat ein Auswahlmenü, bei dem automatisch alle Effekte hinzugefügt werden, die es in dem Programm gibt. Es ist so auf einen Blick ersichtlich, welche Effekte es gibt und es kann ein Effekt ausgesucht werden. Nach dem Auswählen des Effektes, kann durch das Drücken auf den Set-Knopf auf ausgewählten Effekt gewechselt werden. Es werden außerdem die Informationen zu dem aktuellen Effekt angezeigt. Die GUI ermöglicht es so die Effekte schnell und einfach zu wechseln, ohne das Programm neu starten zu müssen.

4.5 Testung

Um die theoretischen Grundlagen in das Programm einzubauen, mussten wir erst einmal den Einfluss verschiedener Techniken und deren Ergebnisse herausfinden und dokumentieren. Ohne funktionierende Algorithmen kann schließlich kein Programm entstehen. Um jegliche Techniken auszuprobieren haben wir verschiedene Testungen gemacht. Für die Tests haben wir den Algorithmus geschrieben, welcher getestet werden soll und schließlich mit einer grafischen Oberfläche, von *pyqtplot,* und einem Audiosignal das Ergebnis analysiert. So waren wir in der Lage verschieden Algorithmen und Techniken auszutesten und stetig zu verbessern.

Im Folgenden möchte ich auf ein paar Testungen eingehen, welche wir durchgeführt haben und hierbei den Aufbau sowie das Ergebnis kurz erläutern. Alle Testungen sind außerdem in der GitHub-Repository[39] enthalten und können bei Interesse auch selbst durchgeführt werden.

4.5.1 FFT / Leistungsspektrum

Als erstes war uns eine Auftrennung eines Signals in seine Frequenzanteile wichtig. Wir haben hier verschiedene FFT-Algorithmen mit $scipy$ sowie $numpy$ ausprobiert, welche das Signal sauber auftrennen sollen. Zudem sollte noch als Vergleich das Leistungsspektrum dargestellt werden, um die Unterschiede zum normalen Amplitudenspektrum erkennen zu können. Die $rfft()$ Funktion von $numpy$, welche die FFT für ein reales Signal bildet, hat sich hier als am Besten erwiesen. Zudem waren beim Leistungsspektrum die enthaltenden Frequenzen besser unterscheidbar vom restlichen Rauschen als beim Amplitudenspektrum.

4.5.2 Fensterung

Bei diesem Test wollten wir den Nutzen eines Fensters vor der FFT herausfinden. Wir haben hierzu eine FFT ohne jegliche Fensterung erstellt, sowie eine FFT mit einem Hanning-Fenster. Bei der Darstellung des kompletten Frequenzspektrums mit Musik, konnte man kein Unterschied erkennen. Jedoch konnten wir bei einheitlichen Frequenzen[40], anstatt Musik, einen klaren Unterschied feststellen. Die FFT ohne Fenster hat hierbei eine viel stärkere Streuung auf benachbarte Frequenzen gehabt anstatt die FFT mit Hanning-Fenster. Der Leck Effekt hat sich somit hier auch bestätigt.

[38] S.: TomSchimansk,. o. D., *CustomTkinter.*
[39] S.: Zimmermann F., Julian D., 2023, *AudioVisualizer.*
[40] S.: YouTube, adminofthissite, o. D., https://www.youtube.com/watch?v=H-iCZElJ8m0.

4.5.3 Melbank

Der Test eines funktionierenden Melbank-Filters ist ohne Probleme verlaufen. Die Frequenzen wurden sauber von der FFT aufgeteilt. Hier wurde lediglich die Theorie bestätigt, dass Melbank-Filter sich hervorragend für Auftrennungen im Frequenzbereich eignen.

4.5.4 Lautstärke-Normalisierung

Die Größe des Betrags vom Amplitudenspektrum/Leistungsspektrum hat sich jedoch als Problem erwiesen. Um den Betrag zu visualisieren zu können, muss dieser zwischen 0 und 1 liegen. Jedoch hängt dieser letzendes von der Lautstärke des Benutzers ab, welcher die Musik laufen lässt. Als ersten Ansatz haben wir hier eine Minimum-Maximum-Normalisierung versucht. Bei dieser Normalisierung wird das komplette Frame des Spektrums durch den höchsten enthaltenden Wert geteilt. Jedoch hat dies zur Folge, dass Unterschiede der Lautstärke über die Zeit nicht mehr wahrnehmbar sind. Daher musste eine andere Lösung her. Wie bereits in *Kapitel 3.7* erwähnt, haben wir für die Normalisierung der Lautstärke einen Exponentialfilter verwendet. Im Gegensatz zu einer Minimum-Maximum-Normalisierung besteht bei dieser Methode der Vorteil, dass eine Anpassung nicht abrupt bei jedem Frame erfolgt. Statt durch das Maximum wird durch den gleitenden Mittelwert vom Filter geteilt. Durch die Glättungsfaktoren kann so die Dauer und Schnelligkeit der Anpassung an die Lautstärke erfolgen.

Als Ergebnis bekamen wir ein Glättungsfaktor von 0.99 bei einem Anstieg, sowie ein Glättungsfaktor von 0.1 bei einem Abfall. Durch einen hohen Glättungsfaktor beim Anstieg, welcher fast einer Minimum-Maximum Normalisierung entspricht, kann garantiert werden, dass der größte Wert niemals den Betrag 1 übersteigt. Bei einem Abfall dagegen passt er sich nur langsam dem Signal an, was zur Folge hat, dass die Unterschiede bei der Lautstärke auch über die Zeit hinweg weiterhin erkennbar bleiben.

5 Effekte

In diesem Kapitel werden wir alle entwickelten Effekte vorstellen und deren Aufbau erklären. Bei allen Beschreibungen empfiehlt es sich jedoch den jeweiligen Effekt dazu anzuschauen, um den Aufbau besser zu verstehen. Hierzu befinden sich auch Beispiele in dem GitHub-Repository.

5.1 Frequenzdarstellung

5.1.1 Melbank Effekt

Der einfachste Effekt in unserem Programm ist der Melbank Effekt. Er zeigt das Leistungsspektrum (*siehe Kapitel 3.4.2*), das durch die Melbank (*siehe Kapitel 3.5*) gefiltert wurde. Zudem lassen sich jeweils die zwei Werte der Exponentialfilter, *alpha_rise* (Faktor für Werte größer als die letzte Vorhersage) und *alpha_decay* (Faktor für Werte kleiner als die letzte Vorhersage), für einen Gain-Normalisierungs-Filter und einen Glättungsfilter (Smoothing-Filter) einstellen.

Der Gain-Filter ist für die Lautstärke-Normalisierung zuständig, siehe *Kapitel 4.5.4*. Die Werte zwischen 0 und 1 geben die Helligkeit der einzelnen LEDs auf dem LED-Streifen an. Da sie durch das Verhältnis zu dem gleitenden Mittelwert berechnet werden, entspricht die Helligkeit der jeweiligen LED der Energie dieses Frequenzbereichs. Damit, wie schon in *Kapitel 4.5.4* beschrieben, weiterhin die Unterschiede in der Lautstärke erkennbar sind müssen die Werte für *alpha_rise* hoch einstellt und den Wert für *alpha_decay* niedrig.

Der Smoothing-Filter glättet das Signal über die Zeit. Dadurch kann das abrupte Wechseln zwischen den einzelnen Frames verhindert werden. Ein zu hoher Wert für *alpha_decay* führt zu dem oben genannten abrupten Wechseln und ein zu geringer Wert führt zu einer zu geringen Veränderung im Signal. Wir haben hier die Werte 0.99 für *alpha_rise* und 0.2 für *alpha_decay* als am besten empfunden. Dadurch entsteht der Effekt, dass es bei einem Schlag schnell aufleuchtet, dann aber nur langsam wieder abflacht.

5.1.2 Spektrum Effekt

Der Spektrum Effekt erbt von dem Melbank Effekt. Er zeigt das aktuelle Spektrum von der Mitte aus symmetrisch. Dafür wird das Spektrum mit dem gleichen gedrehten Spektrum zusammengefügt. So entsteht ein schöner symmetrischer Effekt, bei dem die tiefen Frequenzen in der Mitte angezeigt werden und nach außen hin die immer höheren Frequenzen. Es lässt sich außerdem die minimale und maximale Frequenz einstellen, die in dem Spektrum berücksichtigt werden sollen.

5.2 Energie

Der Energie Effekt stellt die Energie, oder Lautstärke, des Signals dar. Dazu wird der QMW (*Kapitel 3.6*) des Signals berechnet, dazu nutzen wir die $rms(\ )$ Funktion, der geerbten Klasse *AudioEffect*. Dieser QMW wird mithilfe eines Exponentialfilters normalisiert, indem der neue QMW durch die letzte Vorhersage geteilt wird. Der normalisierte QMW wird durch einen anderen Exponentialfilter geglättet, um ein gleichmäßiges Ergebnis zu erhalten. Um das optimale Ergebnis zu erhalten, mussten wir die richtigen Werte für beide Filter finden. Bei dem Gain-Filter, zur Normalisierung, haben wir festgestellt, dass der Faktor für steigende Werte mit 0.9 deutlich höher sein sollte als der Faktor für fallende Werte, der mit 0,001 deutlich niedriger ist. Für den Smoothing-Filter erhalten wir das beste Ergebnis, wenn wir beide Faktoren identisch, auf 0.4, einstellen.

Zum Darstellen von dem geglätteten QMW, nutzen wir die Gauß-Kurve. Es wird eine symmetrische Gauß-Kurve mit der Anzahl der LEDs und einer einstellbaren Standartabweichung erstellt. Diese Gauß-Kurve wird, dann mit dem QMW multipliziert. So

entsteht eine Kurve, die je nach Größe des QMW größer oder kleiner wird. Dadurch lässt sich die Energie auf dem LED-Streifen anzeigen. Je größer die Energie ist umso weiter breiten sich die LEDs von der Mitte aus. Auf diese Weise lässt sich die Lautstärke des Audiosignals gut erkennbar darstellen.

5.3 Shine-Effekt

Der Shine Effekt ist der erste größere Effekt, welcher aus zwei unterschiedlichen Animationen besteht und eine eigene Farbgebung hat.

5.3.1 Peak-Erkennung

Der Shine-Effekt hat einen kurzen aufhellenden Farbenwechsel als Besonderheit, welcher bei jedem Peak in der unteren Bassfrequenz auftritt, sozusagen bei jedem „Bassschlag“. Für diese Erkennung haben wir einen kleinen Algorithmus entwickelt welche grundlegend zwei verschiedenen Signale als Basis nimmt. Er vergleicht hierbei das Eingangssignal mit einem gleitenden Mittelwert. Wenn das Eingangssignal an einem Punkt sehr stark von dem gleitenden Signal abweicht, erkennt er dies als Peak. Des Weiteren überprüft er, ob der Wert von dem gemessenen Peak, dem Wert der letzten Peaks entspricht. Falls dies nicht der Fall ist, wertet er dies als ein Fehler und gibt diesen nicht an das Programm zurück. Dieser Algorithmus hat sich bei den meisten Musikgenres als sehr akkurat herausgestellt.

5.3.2 Animation

Als Grundlage basiert der Effekt auf dem Frequenz-Effekt, welcher die Frequenzen, wie bereits erwähnt, symmetrisch darstellt. Dieser wird in einer gewünschten Farbe dargestellt. Bei einem Peak, wechselt allerdings die Animation zu einer aufhellenden zweiten Farbe. Zusammenfassend soll der Effekt eine schöne Farbendarstellung zu dem Audiosignal geben, welcher jedoch besonders den Bass betont und diesen kennzeichnet.

5.4 Color-Spectrum-Effekt

Ein weiterer größerer Effekt ist der Color-Spectrum-Effekt. Er erzeugt durch das Aufteilen des Spektrums und mithilfe von verschiedenen Farben einen bunten Effekt.

5.4.1 Aufteilung

Dazu wird das Spektrum in drei verschiedene Teile aufgeteilt. So erhalten wir jeweils ein Teil für die tieferen, mittleren und höheren Frequenzen. Diese Teile werden einzeln wieder zusammengefügt, so, dass jedes Teil dreimal nebeneinander abgebildet wird. Jedes Teil wird mit einer anderen Farbe abgebildet, so haben die tieferen, mittleren und höheren Frequenzen jeweils eine andere Farbe. Die Farben lassen sich nach Wunsch einstellen.

5.4.2 Animation

Diese drei Teile werden nun so übereinandergelegt, dass die übereinanderliegenden Teile dreimal auf dem LED-Streifen angezeigt werden. Dadurch entsteht der Effekt, dass jeweils die Farbe, von dem Teil angezeigt wird, dessen Frequenz an dieser Stelle gerade am stärksten ist. Zusammengefasst soll durch das Übereinanderlegen der aufgeteilten Teile in tiefere, mittlere und höhere Frequenzen, die jeweils eine andere Farbe haben, ein schöner bunter Effekt entstehen.

6. Resümee und Ausblick

Die vorliegende Arbeit beschäftigte sich nach der Frage, wie Musik visualisiert werden kann und wie heutige Lichteffekt-Systeme und Sprachanalysen oft funktionieren. Dazu wurden die grundlegenden Konzepte der Audio-Signalverarbeitung erklärt und in einem Programm praktisch angewandt. Aus den Ergebnissen unserer Arbeit lässt sich schließen, dass die Anwendung genannter Konzepte aus Mathematik sowie Informatik es relativ einfach ermöglichen Musik zu analysieren und visuell darzustellen.

Hierbei mussten wir mehrere Dinge beachten. Hierzu gehörte die digitale Darstellung von Signalen, Normalisierung von Frequenzen, das Auftrennen des Signals in seine Frequenzanteile mittels einer Fourier Transformation, das Filtern von Frequenzen sowie die menschliche Hörwahrnehmung. Alle Konzepte tragen einen wichtigen Beitrag zur Audio-Analyse bei, ohne welche es nicht funktionieren würde. Viele Konzepte und Theorien können nicht allein funktionieren, sondern hängen immer voneinander ab

Mit diesen Erkenntnissen konnte ein Programm entwickelt werden, welches durch verschiedene Algorithmen und Prozesse ein Audiosignal zerlegen und analysieren kann, um schließlich einen LED-Streifen aufleuchten zu lassen. Die Algorithmen und Visualisierungen sind hierbei relativ akkurat geworden und können auch kleinere Unterschiede, wie z.B. eine Veränderung im Frequenzspektrum, darstellen.

Die Arbeit zeigt, dass aussagekräftige Darstellungen und Visualisierungen von Musik, mit verschiedenen Techniken aus der Mathematik sowie der Informatik, schnell und effektiv entwickelt werden können. Hierbei können verschiedene Entwicklungshilfen, wie das Darstellen und Auswerten von Diagrammen in pyqtgraph, bei der Entwicklung neuer Algorithmen in der Audioanalyse helfen. Diese Erkenntnisse eröffnen neue Möglichkeiten zur weiteren Forschung und Entwicklung zu diesem Thema. Wie können diese Techniken in eingebetteten Systemen auf Mikrocontrollern in Geräten funktionieren? Wie sieht die Unterstützung für Audioanalyse bei anderen Programmiersprachen aus? Oder wie könnte man eine Drei-Dimensionale Visualisierung erstellen?

Das Spektrum an Möglichkeiten ist groß. Neue Untersuchungen sowie Forschungen in diesem Bereich können langsam diese Fragen beantworten. Weitere Investition in diesen Themenbereich lohnt sich außerdem, da die Verfügbarkeit von OpenSource-Software in diesem Segment, nach unseren Recherchen, noch überschaubar ist. Neue Anwendungen würden einen großen Mehrwert für viele Personen und Entwickler bieten und könnten weiterhin dazu beitragen, die Kunst hinter der Musik jedem Menschen näher zu bringen. Ob in der eigenen Wohnung oder auf Festivals mit den größten Lichtanlagen, die je gebaut wurden.

7. Versicherung

Wir versichern, dass wir diese schriftliche Seminararbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt haben und dass wir alle Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, durch Angabe der Quellen als Entlehnung kenntlich gemacht haben.

Wenn nachweißlich ein Täuschungsversuch durchgeführt worden ist, z.B. Text aus Quellen übernommen und diese Quellen nicht benannt worden sind, wird der gesamte Seminarkurs mit **0 Punkten** (Note 6) bewertet und kann nicht mehr als Ersatz einer Abiturprüfung eingesetzt werden. Diese Note erscheint dann im Abitur Zeugnis.

Mit einer Veröffentlichung der Arbeit erklären wir uns einverstanden. Ich werde mich an alle vorgegebenen Regeln halten und akzeptiere bei Nichtbeachtung die Konsequenzen.

Kapitel 1, 6: Zusammen

Kapitel 2, 3.1, 3.3, 3.4.2, 4, 5.3: Felix Zimmermann

Kapitel 3.2, 3.4.1, 3.5- 3.7, 4.4.3, 5.1, 5.2, 5.4: Julian Dold

Emmendingen, 13.01.2023 Felix Zimmermann _____

Julian Dold_____

8. Abbildungsverzeichnis

9. Literaturverzeichnis

Artistic Licence Integration. (kein Datum). Abgerufen am 27. 05 2023 von artisticlicenceintegration.com: https://artisticlicenceintegration.com/technology-brief/technology-resource/sacn-and-art-net/

Bäckström, T. (21. Mai 2019). *Cepstrum and MFCC*. Abgerufen am 18. Januar 2023 von aalto.fi: https://wiki.aalto.fi/display/ITSP/Cepstrum+and+MFCC

Beckhoff. (kein Datum). *Skalierung von Spektren*. Abgerufen am 27. 02 2023 von infosys.beckhoff.com: https://infosys.beckhoff.com/index.php?content=../content/1031/tf3600_tc3_condition_monitoring/2803646091.html

ESTA. (7. November 2018). *ANSI E1.31.* Abgerufen am 14. Januar 2023 von tsp.esta.org: https://tsp.esta.org/tsp/documents/docs/ANSI_E1-31-2018.pdf

Fayek, H. (21. April 2016). *Speech Processing for Machine Learning*. Abgerufen am 14. Januar 2023 von haythamfayek.com: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

Gründert, S. (11. Januar 2023). *Mel Filter Bank*. Abgerufen am 17. Januar 2023 von github.io: https://siggigue.github.io/pyfilterbank/melbank.html

Hubert, M. (26. Juli 2018). *Wie die Musik entstanden ist*. Abgerufen am 16. Januar 2023 von deutschlandfunk.de: https://www.deutschlandfunk.de/forschung-wie-die-musik-entstanden-ist-100.html

itwissen.info. (15. November 2016). *Emphasis.* Abgerufen am 27. Januar 2023 von itwissen.info: https://www.itwissen.info/Emphasis-emphasis.html

Kumar, A. (6. 10 2018). *Lecture 4 | Discrete Fourier Transform (DFT) with it's Leakage*. Abgerufen am 26. 2 2023 von youtube.com: https://www.youtube.com/watch?v=2c4LXoDqr04

Maier. (kein Datum). *Spektrum Fouriertransformation*. Abgerufen am 17. 05 2023 von http://dodo.fb06.fh-muenchen.de/maier/analytik/Blaetter/A04_Spektrum_Fouriertransformation_a_BA.pdf

MatheRetter. (kein Datum). *Quantisierung*. Abgerufen am 23. Januar 2023 von matheretter.de: https://www.matheretter.de/wiki/shazam-quantisierung

McFee, B. (kein Datum). *Digital Signals Theory*. Abgerufen am 14. Januar 2023 von brianmcfee.net: https://brianmcfee.net/dstbook-site/content/intro.html

Nelles, P. D.-I. (27. Juni 2019). *Vorlesung Signalverarbeitung.* Abgerufen am 14. Januar 2023 von Uni Siegen: https://www.mb.uni-siegen.de/mrt/lehre/sv/signalverarbeitung_2019_06_26.pdf

Neumann, T. (13. März 2010). *Die Wahrnehmung von Rauschen und dessen praktischer Einsatz zur Klanggestaltung.* Abgerufen am 27. Januar 2023 von hdm-stuttgart.de: https://curdt.home.hdm-stuttgart.de/PDF/Neumann.pdf

NumPy. (kein Datum). *NumPy*. Abgerufen am 28. 05 2023 von numpy.org: https://numpy.org/

Pfeilsticker, D. (kein Datum). *Preemphasis Pre-Emphasis Emphasis DAO auf Audio-CDs*. Abgerufen am 14. Januar 2023 von pfeilsticker.com: https://www.pfeilsticker.net/elektronik/preemphasis/

PyAudio Documentation. (kein Datum). Abgerufen am 17. Januar 2023 von https://people.csail.mit.edu/hubert/pyaudio/docs/#

Satz von Plancherel. (kein Datum). Abgerufen am 27. 02 2023 von de.wikipedia.org: https://de.wikipedia.org/wiki/Satz_von_Plancherel

Schmidt, P. D. (kein Datum). *Vorlesung Gruppenkommunikation im Internet.* Abgerufen am 14. Januar 2023 von haw-hamburg.de: https://inet.haw-hamburg.de/teaching/ws-2010-11/technik-und-technologie-i/08_mcast.pdf

Schmitz, E. (kein Datum). *Technik der Fourier-Transformation*. Abgerufen am 06. 04 2023 von https://www.uni-muenster.de/imperia/md/content/physikalische_chemie/praktikum/fourier__transformation__kr_mer__elisabeth__schmitz__rene_.pdf

Siemens. (29. 8 2019). *Windows and Spectral Leakage*. Abgerufen am 26. 2 2023 von community.sw.siemens.com: https://community.sw.siemens.com/s/article/windows-and-spectral-leakage

Siemens. (29. 07 2020). *The Autopower Function… Demystified!* Abgerufen am 27. 02 2023 von community.sw.siemens.com: https://community.sw.siemens.com/s/article/the-autopower-function-demystified

Steinberg. (kein Datum). *Pre-Emphasis*. Abgerufen am 14. Januar 2023 von steinberg.help: https://steinberg.help/wavelab_pro/v9.5/de/wavelab/topics/writing_operations/pre_emphasis_c.html

Stroh, W. M. (kein Datum). *Akustik Kapitel 4: Analog und digital.* Abgerufen am 21. Januar 2023 von musik-for.uni-oldenburg.de: https://www.musik-for.uni-oldenburg.de/akustik/einzelkapitel/kapitel4.pdf

TomSchimansky. (kein Datum). *CustomTkinter*. Abgerufen am 02. 06 2023 von github.com: https://github.com/TomSchimansky/CustomTkinter

TU Wien. (kein Datum). *Abtasttheorem.* Abgerufen am 21. Januar 2023 von ti.tuwien.ac.at: https://ti.tuwien.ac.at/cps/teaching/courses/dspv/files/abtasttheorem.pdf

Uni Kassel. (kein Datum). *Exponentielle Glättung*. Abgerufen am 29. 05 2023 von https://www.unikassel.de/fb07/index.php?eID=dumpFile&t=f&f=2956&token=72b0e9c2d07d4d5a9be05482adbad5a140a89bf2

Vary, P., Heute, U., & Hess, W. (1998). *Digitale Sprachsignalverarbeitung.* B. G. Teubner Stuttgart.

Velardo, V. (19. 10 2020). *Audio Signal Processing for Machine Learning*. Abgerufen am 26. 2 2023 von youtube.com: https://www.youtube.com/playlist?list=PL-wATfeyAMNqIee7cH3q1bh4QJFAaeNv0

Willig, H.-P. (kein Datum). *Quadratisches Mittel*. Abgerufen am 26. 05 2023 von https://www.cosmos-indirekt.de//Physik-Schule/Quadratisches_Mittel

WLED. (kein Datum). *E1.31 (DMX) / Art-Net*. Von kno.wled.ge: https://kno.wled.ge/interfaces/e1.31-dmx/ abgerufen

Zafar, S. (18. November 2022). *Fix Missing or Disable Realtek Stereo Mix in Windows 11/10*. Abgerufen am 17. Januar 2023 von itechtics.com: https://www.itechtics.com/stereo-mix/

Zimmermann, F., & Dold, J. (29. 05 2023). *AudioVisualizer*. Von github.com: https://github.com/felix0351z/AudioVisualizer abgerufen