Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik

Bachelorarbeit Informatik

Entwicklung eines echtzeitfähigen Systems zur Transkription von Musik

Vorgelegt von: Lukas Graber

15. August 2019

Gutachter

Prof. Dr. Oliver Bringmann

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik

Lehrstuhl Technische Informatik
Universität Tübingen

Betreuer

Adrian Frischknecht

Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik

Lehrstuhl Technische Informatik
Universität Tübingen

Graber, Lukas:
Entwicklung eines echtzeitfähigen Systems zur Transkription von Musik
Bachelorarbeit Informatik
Eberhard Karls Universität Tübingen
Bearbeitungszeitraum: 16. April 2018 - 15. August 2019

Abstract

In dieser Bachelorarbeit wurde ein rudimentäres System zur Transkription von Musik entwickelt. Mithilfe des Systems soll die Komposition von Musik automatisiert werden und ausgehend von einem Audiosignal ein digitales Notenblatt generiert werden. Weitere Implementierungsziele waren die Echtzeitfähigkeit der Pipeline und eine laufzeiteffiziente Implementierung. Dadurch soll die Voraussetzung geschaffen sein das System auch auf einem Eingebetteten System (*ES*) ausführen zu können.

Die entwickelte Pipeline ermöglicht eine Ton-, Akkord- und Melodieerkennung. Für viele Instrumente müsste das System allerdings speziell konfiguriert werden. Weiterhin wurden verschiedene Varianten der Pipeline implementiert, welche unterschiedliche Qualitäts- und Performanzanforderungen erfüllen. In Bezug auf Eingebettete Systeme ist die entwickelte Pipeline auf einem *Raspberry Pi 3B* ausführbar.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ALSA	Advanced Linux Sound Architecture
ASDR	Attack, Decay, Sustain, Release
BPM	Beats Per Minute, Schläge pro Minute
CSV	Comma-Seperated Values
DFT	Diskrete Fourier-Transformation
ES	Eingebettetes System
FFT	Fast Fourier-Transform
MIDI	Musical Instrument Digital Interface
PCM	Pulse Code Modulation
RIFF	Resource Interchange File Format
STFT	Short-Time Fourier-Transform
WAV	Waveform Audio File Format

1 Einleitung

1.1 Motivation

Heutzutage besteht die Komposition von Musik größtenteils aus der manuellen Eingabe und Positionierung von Notenwerten auf einem virtuellen Notenblatt in einem Musikeditierungsprogramm [1]. Da solche Programme meist über eine steile Lernkurve verfügen und lediglich als Desktopanwendung existieren, wird die Komposition von Musik unnötig erschwert. Dadurch wird für Komponisten der Übergang vom Papier auf digitale Medien unattraktiv. Weiterhin ist eine gewisse musikalische Expertise in Bezug auf die musikalische Notation Voraussetzung, um ein Musikstück zu komponieren.

In dieser Arbeit soll daher ein System entwickelt werden, welches den Kompositionsprozess automatisiert. Damit würde die Notwendigkeit von musikalischem Vorwissen entfallen, da die Identifizierung der einzelnen Notenwerte in der Melodie und deren Platzierung auf dem Notenblatt durch das System und nicht durch den Komponisten gehandhabt wird. Die Komposition wäre damit nicht nur Komponisten vorbehalten, sondern eröffnet sich einem breiteren Markt.

Für den potenziellen Einsatz in Eingebetteten System oder als mobile Anwendung, soll das System auf einer lokalen Verarbeitung der Audiodaten in Echtzeit beruhen. Es wird dabei bewusst auf eine cloudbasierte oder intelligente Verarbeitung durch neuronale Netze und Methoden des Maschinellen Lernens verzichtet.

1.2 Vorstellung der Bachelorarbeit

Im Zuge dieser Bachelorarbeit wurde eine Pipeline entwickelt, welche den Kompositionsprozess von Musik automatisiert. Es sind drei verschiedene Implementierungen der Pipeline entstanden, welche unterschiedliche qualitäts- und performanzkritische Anforderungen erfüllen. Ein Hauptaugenmerk der Pipeline ist die Echtzeitfähigkeit des Systems. Die drei Varianten können wie Module in automatisierte Tests integriert werden, sodass die Qualität und Performanz der Pipeline beurteilt werden kann.

1.3 Aufbau

Die vorliegende Arbeit untergliedert sich in sechs Kapitel. Es werden zunächst die Grundlagen geklärt, auf denen das entwickelte System aufsetzt. Nach dem Grundlagenkapitel wird das Forschungsgebiet genauer untersucht und der aktuelle Stand der Forschung betrachtet. Es werden sowohl akademische Entwicklungen, wie auch kommerzielle oder öffentlich zugängliche Produkte betrachtet, die in einer bestimmten Form mit dem in dieser Arbeit betrachteten Themengebiet zusammenhängen. Es wird versucht die wesentlichen Trends im Bereich der digitalen Musiktranskription und -verarbeitung zu identifizieren. Anschließend wird die eigentliche Pipeline vorgestellt und deren Bestandteile betrachtet. Dabei werden sowohl die Konzepte, wie auch die Implementierung dargestellt.

Nach der Implementierung wird die Qualität und Performanz des Systems erfasst. Weiterhin werden Erkenntnisse und Ergebnisse vorgestellt, die sich über Benchmarking-Tests und theoretische Überlegungen ergeben haben. Es werden alle drei verschiedenen Implementierungsvarianten der Pipeline untersucht. Die Tests werden auf einem Standardsystem und auf einem Eingebetteten System (*Raspberry Pi 3B*) durchgeführt und die Ergebnisse miteinander verglichen. Das letzte Kapitel fasst die Erkenntnisse der Bachelorarbeit zusammen und schließt mit einem Ausblick ab.

2 Grundlagen

In diesem Kapitel sollen die theoretischen und technischen Grundlagen gelegt werden, welche im weiteren Verlauf dieser Arbeit benötigt werden. Neben mathematischen Grundlagen, soll auch die digitale Signalverarbeitung betrachtet und die Grundlagen der Musiklehre geklärt werden.

2.1 Musiklehre

Das entstehende System soll letztendlich eine Transkription von Musik von einem Audiosignal in eine digitale Notation ermöglichen. An dieser Stelle wird deshalb zunächst ein Exkurs in den Bereich der musikalischen Notenlehre unternommen. Mit diesen Informationen sollen anschließende musikalische Referenzen verstanden werden. Die folgenden Theorien beziehen sich auf [2].

Notation

Die Benutzung von Notenblättern stellt eine Notwendigkeit in der Musik dar. Sämtliche Melodien eines Musikstücks werden auf einzelnen Notenblättern notiert und verschiedenen Instrumenten zugewiesen.
Melodien werden generell in einem Notensystem festgehalten. Das Notensystem besteht grundlegend aus fünf gestapelten horizontalen Linien. Diese bieten dem Komponisten und Musiker ein Koordinatensystem für die einzelnen Töne. Melodien bestehen aus einzelnen Notenwerten und Pausen, die im Notensystem platziert werden. Je höher die Notenwerte im Notensystem liegen, desto höher ist der Ton. Notenwerte und Pausen weisen eine Notenlänge auf, wobei Notenwerte neben der Tonlänge zusätzlich eine Tonhöhe aufweisen.
Die Notenwerte können dabei weit unter oder über dem eigentlich abgebildeten fünf Linien des Notensystems liegen. Zur besseren Orientierung für den Musiker werden für entsprechende Notenwerte zusätzliche Hilfslinien eingeführt.

Notenwerte unterscheiden sich in der Tonhöhe und der Tonlänge. Für die Tonhöhe ist der Halbton die kleinste zu unterscheidende Noteneinheit. Weiterhin werden zwölf aufeinanderfolgende Halbtöne zu einer Oktave zusammengefasst.(2.1)

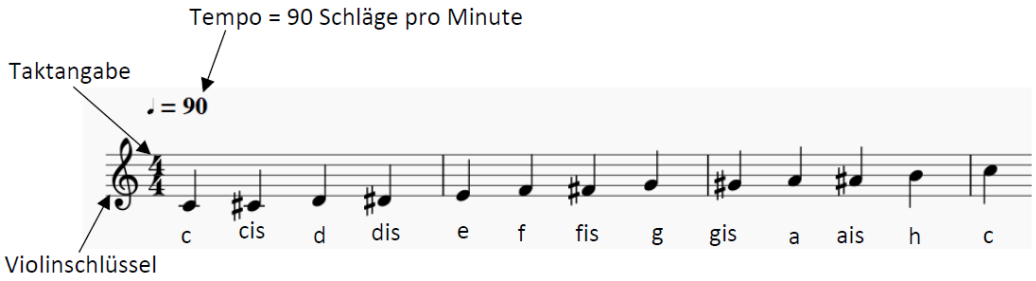


Abbildung 2.1: Tonleiter.

Zu Beginn jedes Notensystems findet sich außerdem ein Notenschlüssel. Dieser gibt den Referenzton im Notensystem an, an dem sich das dafür vorgesehene Instrument orientieren muss.(2.1)
Melodien werden zur besseren Strukturierung in Takte aufgeteilt. Die Noten und Pausen innerhalb eines Taktes stehen zwischen zwei Taktstrichen. Eine Melodie besteht grundsätzlich aus Notenwerten und Pausen, die eine unterschiedliche Notenlänge aufweisen.(2.2)

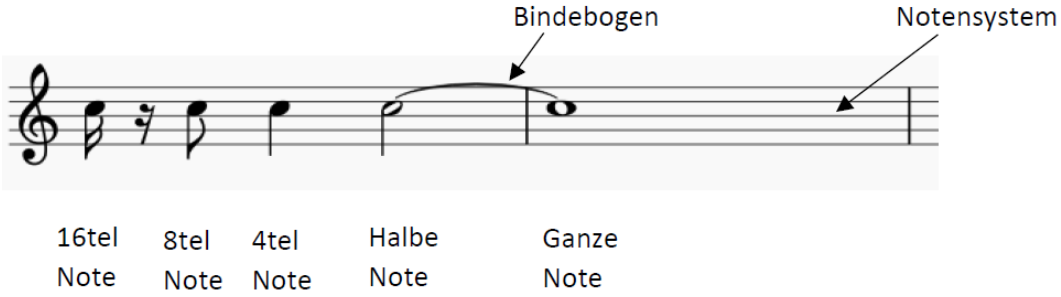


Abbildung 2.2: Notenlänge.

Die Notenlänge orientiert sich dabei an der Grundeinheit des Musikstücks. Jedes Notensystem ist dazu mit einer Taktangabe versehen.(2.1) Dabei handelt es sich meist um eine Bruchangabe, wobei der Denominator die Länge der Grundeinheit angibt. Dies können Achtel, Viertel oder Halbe sein. Der Numerator gibt die Anzahl der Grundeinheiten innerhalb eines Taktes an.
Zur zeitlichen Orientierung des Musikers innerhalb Notensystems wird ein Tempo vorgegeben. Die Einheit des Tempos ist eine bestimmte Anzahl Schläge pro Minute notiert. Je höher diese Zahl ist, desto mehr Schläge werden pro Minute durchgeführt. Es werden dadurch mehr Takte durchschritten und damit das Tempo erhöht. Der Dirigent oder ein Metronom gibt dabei das Tempo vor. Musiker orientieren sich an der Taktvorgabe und können dadurch die Ton- und Pausenlängen relativ abschätzen. Das Orchester wird durch die Taktvorgabe synchronisiert.
Da nicht jedes Instrument über die gesamte Laufzeit des Musikstückes eine Melodie spielen muss, ist es außerdem von Bedeutung Pausenwerte zu notieren. Auch diese können unterschiedliche Längen aufweisen.(2.3)

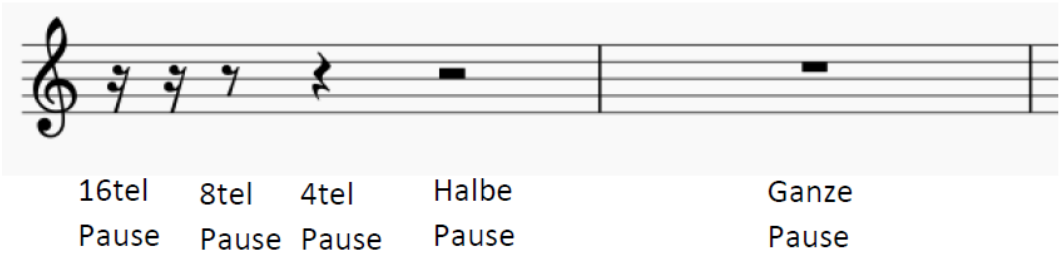


Abbildung 2.3: Pausenwerte.

Auch die Lautstärke kann über die Notation geregelt werden. Dabei steht *p* für leise (*piano*), **mf** für mittel laut (*mezzo forte*) und *f* für laut (*forte*).(2.4)

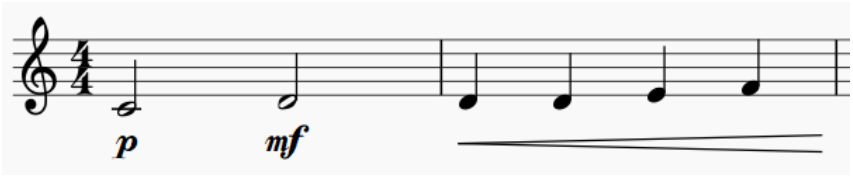


Abbildung 2.4: Dynamik.

Weiterhin kann zu einem bestimmten Zeitpunkt des Stückes mehrere Melodien unterschiedliche Töne gleichzeitig spielen. Dadurch überlagern sich diese Schwingungen. Diese Eigenschaft von Schwingungen wird Interferenz genannt und führt zur Erzeugung von Akkorden. Akkorde können in unterschiedliche Formen auftreten (Dur, Moll,...).(2.5)

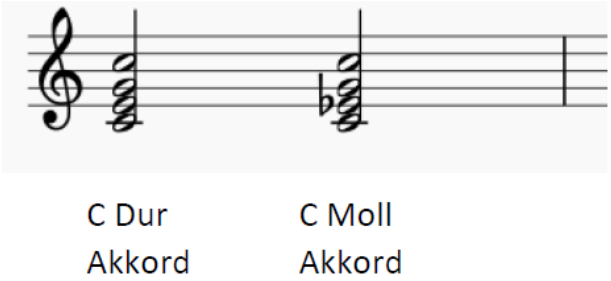


Abbildung 2.5: Akkorde.

2.2 Physik

Um die benötigten Konzepte hinter dem entwickelten System verstehen zu können, wird im Folgenden eine Lösung der Problemstellung über einen Top-Down-Entwurf

konstruiert. An dieser Stelle wird daher zunächst versucht den Begriff Musik zu formalisieren und wissenschaftlich aufzuarbeiten.

Wellenkonzepte

Musik ist zunächst nichts anderes als Schall, welches physikalisch gesehen in den Bereich der Wellentheorie fällt. Wellen beschreiben Vibrationen in Raum und Zeit, die Energie in sich tragen. Die für die Musik interessante Wellenart sind die Longitudinalwellen, zu denen auch Schall und damit Musik gehört. Longitudinalwellen schwingen in Ausbreitungsrichtung mit einer bestimmten Frequenz und beschreiben Druckwellen. Je nach Wert der Frequenz schwingt das Medium unterschiedlich schnell. Dies führt zu einer unterschiedlich starken Kompression des umliegenden Mediums an einem bestimmten Ort.[3]

Das menschliche Ohr ist dazu in der Lage diese Druckänderungen in einem Bereich von 20Hz bis 20kHz wahrzunehmen (2.6). Frequenzen über (Ultraschall) und unter (Infraschall) diesem Bereich können vom menschlichen Ohr nicht mehr aufgelöst werden.[4] Laut Abbildung 2.6 liegt die Musikwahrnehmbarkeit zwischen 50 Hz und 10 kHz.

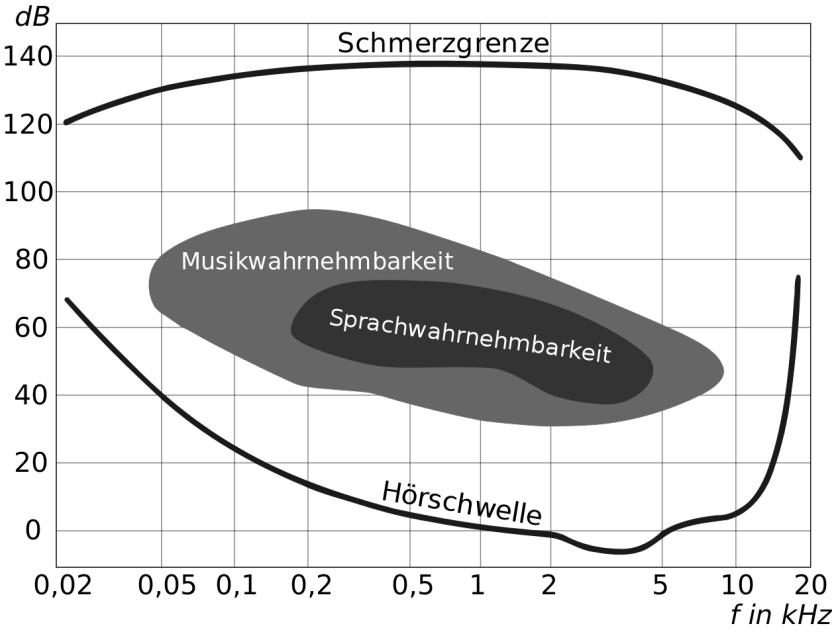


Abbildung 2.6: Auditiver Bereich.[4]

Wellen beziehungsweise Schwingungen weisen neben Frequenzen auch Amplituden auf. Amplituden beschreiben die Größe des maximalen Ausschlag der Welle. Diese manifestiert sich für die Musik als Lautstärke. Wellen mit unterschiedlichen Frequenzen und Amplituden können in Abbildung 2.8 betrachtet werden.

Frequenz

Für die Applikation ist es von großer Bedeutung die Tonhöhe eines Tones bestimmen zu können. Die Tonhöhe ist durch die Luftkompressionsänderung aufgrund unterschiedlicher Frequenzen von Signalen gegeben, welche durch das menschliche Ohr wahrgenommen werden kann. Dabei gilt, je höher die Frequenz, desto höher ist der Ton.

Es zeigt sich, dass die Frequenzen von Tönen mathematisch erfasst werden können. Die einzelnen Frequenzen werden dabei in Bezug auf einen Referenzton (Kammerton) definiert. Der international definierte Kammerton ist die Note $A4$ mit einer Frequenz von 440Hz [5]. Dabei beschreibt A den Notennamen und 4 die entsprechende Oktave, in der der Ton zu finden ist. Der Eintrag für den Kammerton $A4$ ist in Tabelle (2.1) gelb hinterlegt.
Ausgehend von diesem Referenzton werden die Frequenzen aller Töne darüber und darunter berechnet. Die entsprechenden Frequenzen von Tönen können über folgende Formel berechnet werden [6]:

$$f_n = f_0 \cdot (a)^n \tag{2.1}$$

Dabei beschreibt f_0 die Frequenz des Referenztones. Dieser ist im Allgemeinen der Kammerton $A4$ mit einer Frequenz von $f_0 = 440$Hz. Der Exponent n gibt an wie viele Halbtöne über oder unter dem Referenzton die Frequenz berechnet werden soll. Gilt $n < 0$, dann werden die Frequenzen von Tönen berechnet, die unterhalb des Referenztones liegen. Für $n > 0$ werden Frequenzen von Tönen berechnet, die oberhalb des Referenztones liegen.

Die Basis $a = (2)^{\frac{1}{12}}$ beschreibt die zwölfte Wurzel von 2. Damit muss diese Zahl zwölf mal mit sich selbst multipliziert werden, damit sich exakt die Zahl 2 ergibt. Damit wird erreicht, dass die Skala auf zwölf Halbtöne angepasst wird und die Frequenz sich alle zwölf Halbtöne verdoppelt. Als Ergebnis der Formel 2.1 resultiert die Frequenz f_n des zu berechnenden Tones. Dieser liegt, abhängig vom Vorzeichen von n, n Halbtöne unter oder über dem Referenzton. Es zeigt sich weiterhin, dass die Töne mit dem gleichen Notennamen aus unterschiedlichen Oktaven jeweils über den Faktor 2 frequenztechnisch miteinander in Beziehung stehen. In Tabelle 2.1 sind die gerundeten Frequenzwerte für alle Halbtöne innerhalb der ersten 10 Oktaven aufgetragen. Über Tabelle 2.1 kann damit zu einem Ton die entsprechende Frequenz ermittelt werden. Weiterhin können zu bestimmten Frequenzen die zugehörigen Notenwerte bestimmt werden.

Cent-Skala

Die Frequenzen der einzelnen Töne befinden sich über Formel 2.1 auf einer exponentiellen Skala. Dadurch sind die Frequenzunterschiede zwischen den Halbtönen nicht ohne Weiteres miteinander vergleichbar. Je höher die Töne, desto weiter lie-

Note/Oktave	0	1	2	3	4	5	6	7	8	9	10
C	16	33	65	131	262	523	1047	2093	4186	8372	16744
C#	17	35	69	139	277	554	1109	2217	4435	8870	17740
D	18	37	73	147	294	587	1175	2349	4699	9397	18795
D#	19	39	78	156	311	622	1245	2489	4978	9956	19912
E	21	41	82	165	330	659	1319	2637	5274	10548	21096
F	22	44	87	175	349	698	1397	2794	5588	11175	22351
F#	23	46	93	185	370	740	1480	2960	5920	11840	23680
G	25	49	98	196	392	784	1568	3136	6272	12544	25088
G#	26	52	104	208	415	831	1661	3322	6645	13290	26579
A	28	55	110	220	440	880	1760	3520	7040	14080	28160
A#	29	58	117	233	466	932	1865	3729	7459	14917	29834
B	31	62	123	247	494	988	1976	3951	7902	15804	31608

Tabelle 2.1: Auftragung der gerundeten Frequenzen von 10 Oktaven abgeleitet aus dem Kammerton $A4$ - 440Hz (gelb markiert).

gen deren Frequenzen auseinander. Um diese Abhängigkeiten untereinander auf eine lineare Skala zu transformieren, kann folgende Formel [7] verwendet werden:

$$c = 1200 \cdot log_2(\frac{f_2}{f_1}) \text{ cent} \qquad \textbf{2.2}$$

Diese Formel liefert einen Frequenzunterschied in der Einheit cent, wobei die exponentielle Skala mithilfe eines Logarithmus einbezogen wurde. Weiterhin werden auf die zwölf Halbtöne in einer Oktave 1200 cent verteilt. Als Ergebnis liegen zwei Halbtöne exakt 100 cent auseinander. Damit bietet die Formel 2.2 eine Möglichkeit die Frequenzunterschiede zwischen zwei Frequenzen genauer zu fassen, als dies mit einer Auflösung auf Halbtonebene möglich wäre.
Die Formel 2.2 wird später dazu verwendet werden zwei Frequenzen miteinander zu vergleichen. Die Einheit cent wird als frequenzunabhängiges Maß verwendet, um Frequenzunterschiede von Tönen unterschiedlicher Oktaven effektiv miteinander zu vergleichen.

Lautstärke

Lautstärke entspricht der Amplitude eines Signals und besitzt die Einheit Dezibel. Dies ist eine relative Einheit und hat daher keine absolute Aussagekraft. In der

Realität werden Intensitäten F von Signalen mit Referenzintensitäten F_0 verglichen, um daraus die relative Einheit Dezibel herzuleiten [8][9]:

$$L_F = 10 \cdot log_{10}(\frac{F^2}{F_0^2})\text{dB} = 20 \cdot log_{10}(\frac{F}{F_0})\text{dB} \quad \textbf{2.3}$$

Die berechneten Werte liegen auf einer logarithmischen Skala und können negative Werte annehmen. Dies liegt daran, dass sich die Werte immer auf einen Referenzwert beziehen. In diesem Fall bedeutet 0dB, dass sich das Signal an der Grenze des von dem Aufnahmemedium (menschliches Ohr, Mikrofon) auflösbaren Lautstärkebereichs befindet. Ein positiver Wert bedeutet, dass der Schall lauter ist, als der Referenzwert. Ein kleinerer Wert bedeutet, dass der Schall leiser als der Referenzwert.

Harmonische, Obertöne und Untertöne

Das bisher erörterte physikalische Modell geht davon aus, dass jedem Ton eine Frequenz zugeordnet ist. In der Realität besitzt jedes Instrument eine individuelle Klangfarbe. Dadurch weist der gleiche Notenwert ein unterschiedliches Klangverhalten für verschiedene Instrumente auf. Der Grund liegt darin, dass neben der Hauptfrequenz des eigentlichen Tones zusätzliche Frequenzen im Signal enthalten sind, welche zum Beispiel durch die Bauform des Instrumentes gegeben sind. Jedes Instrument besitzt also ein eigenes Klangprofil.[10]
Um das bisher kennen gelernte Modell an diese Beobachtung anzupassen, wird ein Verständnis von Ober- und Untertönen benötigt. Dies sind Schwingungen, die entweder mit einer niedrigeren Frequenz (Untertöne) oder mit einer höheren Frequenz (Obertöne) als eine vorgegebene Frequenz (Grundschwingung) f_1 schwingen. Harmonische Schwingungen beschreiben dabei Schwingungen, die eine ganzzahlig vielfache Frequenz der Grundfrequenz f_1 aufweisen. Die n-te Harmonische kann über folgende Formel berechnet werden [11]:

$$f_n = f_1 \cdot n, n \in \mathbb{N} \quad \textbf{2.4}$$

2.3 Digitale Signalverarbeitung

Nyquist-Shannon Abtasttheorem

Das Abtasttheorem von Nyquist und Shannon ist eines der entscheidenden Theoreme der digitalen Signalverarbeitung. Das Theorem besagt, dass ein Frequenzbereich von $[0; f_{max}]$Hz lediglich dann durch ein Analog-Digital-Wandler (Mikrofon,...) aufgelöst werden kann, falls die Abtastrate Fs eine Frequenz von $Fs \geq 2 \cdot f_{max}$ aufweist. Oder anders gesagt, ein Signal mit einer Frequenz f kann nur dann vollständig digital rekonstruiert werden, falls für die Samplerate ein Wert von $Fs \geq 2 \cdot f$ gewählt

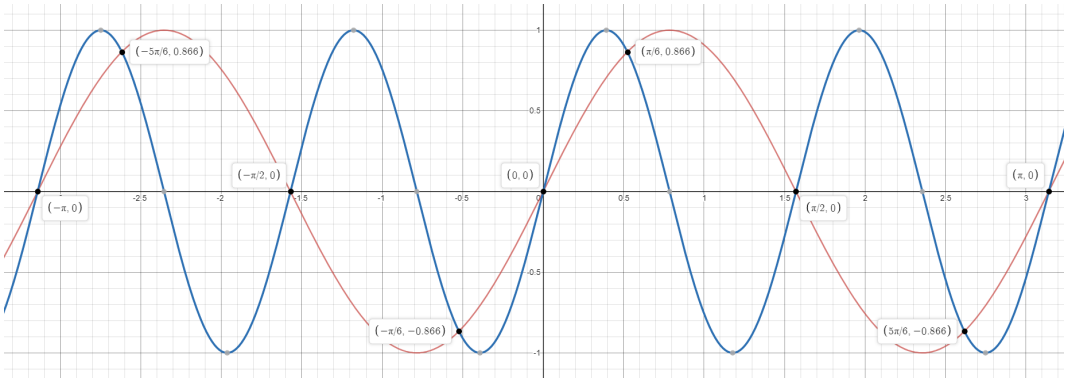


Abbildung 2.7: Demonstration des Nyquisttheorems anhand der Funktionen $\sin 2x$ und $\sin 4x$

wird.[12][13]

Um die Idee des Abtasttheorems zu demonstrieren, wird Abbildung 2.7 angeführt. In Abbildung 2.7 sind die Funktion $\sin 2x$ (rote Kurve) und $\sin 4x$ (blaue Kurve) aufgetragen. Die blaue Kurve besitzt damit eine doppelt so hohe Frequenz wie die rote Kurve. Weiterhin sind acht Messpunkte (Samples) aufgenommen worden und in der Abbildung markiert. Diese acht Samples liegen exakt auf beiden Kurven. Das heißt, falls lediglich diese acht Messpunkte enthalten sind, kann nicht entschieden werden, ob diese zu der roten oder der blauen Kurve gehören. Die blaue Kurve besitzt damit eine zu hohe Frequenz, als dass sie mit den acht Messwerten rekonstruiert werden könnte.

Mithilfe des Nyquisttheorems lässt sich auch eine in Audioaufnahmegeräten verbreitete Sampling Rate ableiten. Das menschliche Gehör besitzt einen auflösbaren Bereich von 20Hz-20kHz 2.6. Um Frequenzen von 20 kHz mit der FFT auflösen zu können, werden also $> 2 \cdot 20000$ Samples pro Sekunde benötigt, um das ursprüngliche Signal rekonstruieren zu können. Aus sicherheitstechnischen Gründen wird die Samplingrate für Audio auf 44100 Hz gesetzt.

Periodische Signale und Fourier

In der Realität treten keine reinen Töne auf, sondern es werden Tongemische, so genannte Klänge wahr genommen. Diese bestehen jeweils aus Einzelschwingungen, die sich überlagern und einen charakteristischen Klang ergeben. Jede Einzelschwingung trägt damit zum Gesamtsignal bei. Dieser Sachverhalt ist im Folgenden demonstriert. Abbildung 2.8 zeigt die Einzelfrequenzen eines Audiosignals. Abbildung 2.9 ist die Überlagerung, also die Summe der Einzelfrequenzen.
Die an dieser Stelle gewonnene Erkenntnis macht das Erkennen von Tönen, Akkorden und Melodien zu einem komplexen und potenziell rechenintensiven Unterfangen. Es wird also ein effizientes Verfahren benötigt, welches es ermöglicht prägnante

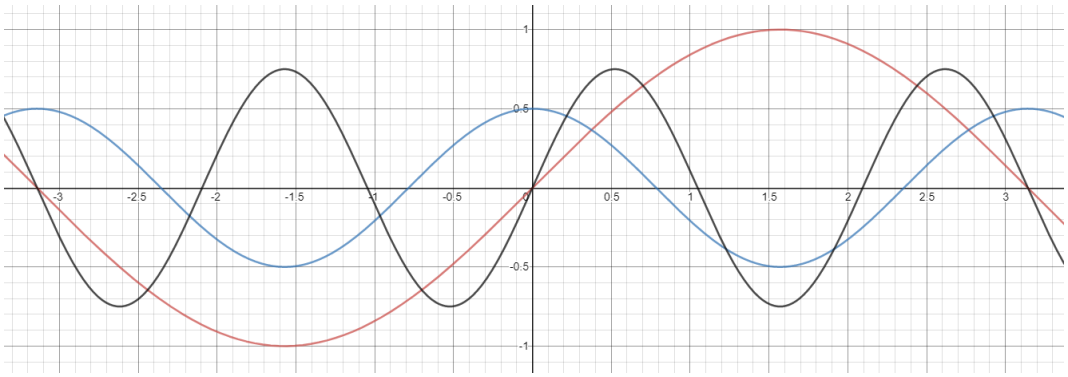


Abbildung 2.8: Auftragung der Einzelschwingungen $\sin x$(rote Kurve),$0.5 \sin 2x$ (blaue Kurve),$0.75 \sin 3x$ (schwarze Kurve) im Intervall $[-\pi, \pi]$.

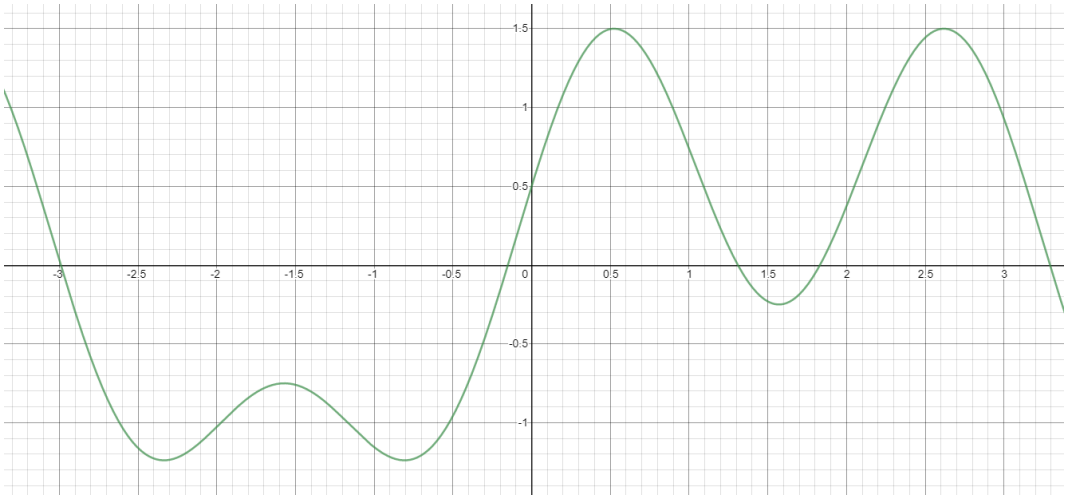


Abbildung 2.9: Auftragung der Gesamtschwingung $\sin x + 0.5 \sin 2x + 0.75 \sin 3x$ im Intervall $[-\pi, \pi]$.

Schwingungen aus dem Gesamtsignal herauszufiltern. Dies führt in den Bereich der Fourier-Analysis.

2.4 Mathematik

Die mathematische Grundlage für die Transkription von Audiosignalen liefert die Fourier-Analysis. Diese gliedert sich in zwei Bereiche: Fourier-Synthese und Fourier-Transformation. Der Prozess ausgehend von den Einzelschwingungen zur Gesamtschwingung wird als Fourier-Synthese bezeichnet. Der Vorgang zur Filterung der Einzelfrequenzen aus dem Gesamtsignal wird als Fourier-Transformation bezeichnet.[14]

2.4.1 Fourier-Analysis

Mit der Fourier-Analyse wird versucht periodische Vorgänge über trigonometrische Funktionen zu approximieren. Es ist folgender endlichdimensionaler Unterraum gegeben, der eine Orthogonalbasis zu einem Vektorraum W bildet [15][16]:

$$\{cos(0x), cos(1x), cos(2x), ...,$$
$$sin(0x), sin(1x), sin(2x), ...\}$$

Ein Signal soll damit als Linearkombination von Sinus- und Kosinusfunktionen unterschiedlicher Frequenzen dargestellt werden [17]. Dabei können die einzelnen Linearfaktoren durch Faktoren unterschiedlich gestreckt sein. Die Werte $sin(0x) = 0, cos(0x) = 1$ können dazu verwendet werden die approximierende Funktion auf der y-Achse zu verschieben, um diese weiter an das Ausgangssignal anzunähern. Da es sich um periodische Signale handelt, wiederholen sich die Funktionswerte. Die größte Periode besitzt die Funktion $sin(1x)$ beziehungsweise $cos(1x)$. Diese besitzt ein Intervall von $[-\pi, \pi]$. Für die restlichen Funktionen ergeben sich jeweils Signale mit kleinerer Periode. Diese sind dadurch bereits komplett im Intervall $[-\pi, \pi]$ enthalten.

Mathematische Herleitung

Die verschiedenen Funktionen können mithilfe der Orthonormalsysteme in einem Term zusammengeführt werden. Vektoren $(\overrightarrow{v_1}, \overrightarrow{v_2}, ..., \overrightarrow{v_n})$ bilden ein Orthonormalsystem, falls die Norm der einzelnen Vektoren jeweils 1 ergibt und gleichzeitig die Vektoren v_k paarweise orthogonal zueinander stehen.[18] Ein gesuchter Punkt p' im Vektorraum kann über folgende Formel ermittelt werden [19]:

$$\overrightarrow{p'} =< \overrightarrow{p}, \overrightarrow{v_1} > \cdot \overrightarrow{v_1} + < \overrightarrow{p}, \overrightarrow{v_2} > \cdot \overrightarrow{v_2} + ... + < \overrightarrow{p}, \overrightarrow{v_n} > \cdot \overrightarrow{v_n} \quad \textbf{2.5}$$

$$= \overrightarrow{p'} = a_1 \cdot \overrightarrow{v_1} + a_2 \cdot \overrightarrow{v_2} + ... + a_n \cdot \overrightarrow{v_n} \quad \textbf{2.6}$$

Dabei beschreibt $\overrightarrow{p}$ den gesuchten Punkt im Vektorraum, der über die Vektoren $\overrightarrow{v_i}$ approximiert werden soll. Es ergibt sich ein approximierter Punkt $\overrightarrow{p'}$ mit geringstem Abstand zu allen Vektoren. Das Skalarprodukt $< \overrightarrow{p}, v_k >$ kann als Faktor a_k bezeichnet und beschreibt den Anteil von v_k an der Approximation von $\overrightarrow{p'}$. Überträgt man diese Überlegungen auf akustische Signale, so soll die mathematische Repräsentation des Signals der Funktion $f(x)$ approximiert werden. Für die Vektoren werden die Sinus- und Kosinusfunktionen verwendet. Die Approximation wird durch ein Fourierpolynom $F_n f(x)$ beschrieben. Es ergibt sich [15]:

$$F_n f(x) = \frac{a_0}{2} + \sum_{k=1}^{n} (a_k \cdot \cos kx + b_k \cdot \sin kx) \tag{2.7}$$

$$a_k =< f, \cos kx >= \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} f(x) \cos kx \ \mathrm{d}x \tag{2.8}$$

$$b_k =< f, \sin kx >= \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} f(x) \sin kx \ \mathrm{d}x \tag{2.9}$$

Wird die Formel auf ein Audiosignal übertragen, dann bezeichnet $F_n f(x)$ die Approximation der unbekannten Funktion f durch ein Fourierpolynom. Die Funktion f beschreibt eine Schwingung und kann aus einer Kombination aus Sinus- und Kosinusfunktionen dargestellt werden. Die Funktion f beschreibt damit das Eingabesignal, wobei lediglich eine endliche Menge mit n Messwerten erhalten wird. Daher kann ein Summe über n verwendet werden. Die Faktoren a_k und b_k beschreiben dabei die Anteile der k-ten Kosinus- und Sinusfunktionen am Signal f. Die Faktoren a_k und b_k werden durch ein Skalarprodukt berechnet. Dabei werden Sinusfunktionen und Cosinusfunktionen einzeln berücksichtigt. Damit entspricht $< f, g >= \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} f(x)g(x)dx$ dem Skalarprodukt zwischen zwei Funktionen [20]. Da $b_0 = sin(0x) \cdot < f, sin(0x) >= 0$, fällt der Sinusterm aus der Gleichung. Der Vorfaktor $\frac{a_0}{2}$ ergibt sich als normierte Variante für $cos(0x) \cdot < f, cos(0x) >= \frac{a_0}{2}$. Die restlichen Skalarprodukte sind bereits auf 1 genormt.[14]

Die eigentliche Fourier-Analysis findet im Komplexen Zahlenraum statt. Im Folgenden werden die bisherigen Überlegungen auf den komplexen Zahlenraum überführt $\mathbb{R} \longrightarrow \mathbb{C}$. Dies wird über trigonometrische Polynome vom Grad n realisiert [21][22]:

$$x \longrightarrow \sum_{k=-n}^{n} c_k \cdot e^{ikx} \tag{2.10}$$

Für die weiteren Umformungen wird die Kenntnis der Eulerschen Formel benötigt [23]:

$$e^{ix} = \cos x + i \cdot \sin x \tag{2.11}$$

$$e^{i\pi} = -1 \tag{2.12}$$

Aus der Formel 2.10 kann die diskrete Form hergeleitet werden. [21][14]

$$\begin{aligned}
\sum_{k=-n}^{n} c_k \cdot e^{ikx} &= c_0 + \sum_{k=1}^{n}(c_k e^{ikx} + c_{-k}e^{-ikx}) \\
&= c_0 + \sum_{k=1}^{n}(c_k(\cos kx + i\sin kx) + c_{-k}(\cos -kx + \sin -kx)) \quad | \text{ Eulersche Formel 2.11} \\
&= c_0 + \sum_{k=1}^{n}(c_k(\cos kx + i\sin kx) + c_{-k}(\cos kx - \sin kx)) \\
&= c_0 + \sum_{k=1}^{n}((c_k + c_{-k})\cos kx + i(c_k - c_{-k})\sin kx) \\
&= \frac{a_0}{2} + \sum_{k=1}^{n}(a_k \cos kx + b_k \sin kx) \quad | \mathbb{C} \to \mathbb{N}
\end{aligned}$$

Durch den Übergang aus dem komplexen in den reellen Zahlenraum kann das Signal damit in Imaginär- und Realteil aufgeteilt werden.

$$\frac{a_0}{2} = Re(c_0), a_k = Re(c_k + c_{-k}), b_k = Im(c_{-k} - c_k)$$

Damit ergibt sich die ursprüngliche Formel. Weiterhin gilt für das Skalarprodukt zweier Funktionen im Komplexen folgende Gleichung [20].

$$< f, g >= \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)\overline{g(x)}\,\mathrm{d}x \qquad \textbf{2.13}$$

Aus diesen Informationen kann das eigentliche Fourierpolynom gebaut werden [14]:

$$F_n f(x) = \sum_{k=-n}^{n} c_k e^{ikx} \qquad \textbf{2.14}$$

$$c_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-ikx}\,\mathrm{d}x \qquad \textbf{2.15}$$

Als weitere theoretische Grundlage sind die komplexen Einheitswurzeln von Bedeutung. Im Folgenden ist die primitive n-te Einheitswurzel aufgezeigt [24]:

$$\omega_n = e^{\frac{2\pi i}{n}} \qquad \textbf{2.16}$$

Ausgehend von diesen mathematischen Grundlagen kann nun die Diskrete Fourier Transformation (DFT) konstruiert werden.

2.4.2 Diskrete Fourier-Transformation (DFT)

Die DFT bildet ein zeitdiskretes periodisches endliches Signal auf ein diskretes periodisches Frequenzspektrum ab. Dadurch wird eine Transformation eines Signals von der Zeitdomäne in die Frequenzdomäne ermöglicht. Die DFT spaltet ein periodisches Signal in seine einzelnen Frequenzanteile auf und übernimmt dabei die Funktion eines Frequenzspektrogramms.

Das bisher ermittelte Integral 2.14 stellt in der Praxis zwei Probleme dar [14]:

- Ein Integral kann nicht rechnerisch bestimmt werden.
- f(x) ist keine Funktion, sondern ein Vektor der Länge n.

Die Annahme ist nun, dass die aufgenommenen Messpunkte im Messintervall gleichmäßig verteilt (äquidistant) sind. Die einzelnen Punkt sind also jeweils folgendermaßen über das Intervall $[-\pi; \pi]$ verteilt:

$$x_j = -\pi + j \cdot \frac{2\pi}{n}, j = 0, 1, 2, ..., n-1 \qquad \textbf{2.17}$$

Dabei beschreibt $-\pi$ den Startpunkt im Intervall $[-\pi, \pi]$. Das Intervall ist über $\frac{2\pi}{n}$ auf n Punkte aufgeteilt. Damit können n Messwerte im Signal abgedeckt werden. Zur praktischen Berechnung wird anschließend das Integral über eine Summe von Rechtecksflächen approximiert [14].

Approximation des Integrals

$$
\begin{aligned}
c_k &= \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-ikx}\,\mathrm{d}x \\
&= \frac{1}{2\pi}\sum_{j=0}^{n}(f(x_j)e^{-ikx_j}\cdot\frac{2\pi}{n}) && \text{| kontinuierlich} \rightarrow \text{diskret} \\
&= \frac{1}{n}\sum_{j=0}^{n} f(x_j)e^{-ikx_j} \\
&= \frac{1}{n}\sum_{j=0}^{n} f(x_j)e^{k(-i(-\pi+j\cdot\frac{2\pi}{n}))} && \text{| Formel 2.17} \\
&= \frac{1}{n}\sum_{j=0}^{n} f(x_j)e^{ki\pi-kij\cdot\frac{2\pi}{n}} \\
&= \frac{1}{n}\sum_{j=0}^{n} f(x_j)(e^{i\pi})^k\cdot e^{-ikj\cdot\frac{2\pi}{n}} \\
&= \frac{1}{n}\sum_{j=0}^{n} f(x_j)(-1)^k\cdot e^{-ikj\cdot\frac{2\pi}{n}} && \text{| Eulersche Identität 2.11} \\
&= \frac{1}{n}\sum_{j=0}^{n} f(x_j)(-1)^k\cdot\omega^{-kj} && \text{| Einheitswurzel 2.16} \\
&= \frac{(-1)^k}{n}\sum_{j=0}^{n} f(x_j)\cdot\omega^{-kj}
\end{aligned}
$$

Der Koeffizient c_k des Koeffizientenvektors c kann also über Formel 2.18 berechnet werden. Dabei beschreibt f ein Vektor der Länge n, wobei dieser zum Beispiel den in einer Zeiteinheit durch das Mikrofon gesampleten Daten entspricht.

$$
c_k = \frac{(-1)^k}{n}\sum_{j=0}^{n} f(x_j)\cdot\omega^{-kj} = \frac{(-1)^k}{n}\sum_{j=0}^{n} y_j\cdot\omega^{-kj} \qquad \textbf{2.18}
$$

Matrix-Schreibweise

Die Formel 2.18 kann in eine Matrix umgewandelt werden [14]. Dabei entspricht die Formel 2.18 einer Multiplikation einer Matrix, dessen Einträge Einheitswurzeln darstellen mit einem Vektor $y = f$. Dabei ist der Faktor $(-1)^k$ bereits im Koeffizi-

entenvektor c verrechnet.

$$\frac{1}{n}\begin{bmatrix} 1 & \omega_n^{\lfloor \frac{n}{2} \rfloor} & \omega_n^{2\cdot\lfloor \frac{n}{2} \rfloor} & \cdots & \omega_n^{(n-1)\lfloor \frac{n}{2} \rfloor} \\ 1 & \omega_n^{\lfloor \frac{n}{2} \rfloor - 1} & \omega_n^{2\cdot(\lfloor \frac{n}{2} \rfloor - 1)} & \cdots & \omega_n^{(n-1)(\lfloor \frac{n}{2} \rfloor - 1)} \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 1 & \omega_n^{-\lfloor \frac{n}{2} \rfloor + 1} & \omega_n^{2\cdot(-\lfloor \frac{n}{2} \rfloor + 1)} & \cdots & \omega_n^{(n-1)(-\lfloor \frac{n}{2} \rfloor + 1)} \\ 1 & \omega_n^{-\lfloor \frac{n}{2} \rfloor} & \omega_n^{2\cdot(-\lfloor \frac{n}{2} \rfloor)} & \cdots & \omega_n^{(n-1)(-\lfloor \frac{n}{2} \rfloor)} \end{bmatrix} \cdot \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} = \begin{bmatrix} c_{-\lfloor \frac{n}{2} \rfloor} \\ -c_{-\lfloor \frac{n}{2} \rfloor + 1} \\ \vdots \\ -c_{\lfloor \frac{n}{2} \rfloor - 1} \\ c_{\lfloor \frac{n}{2} \rfloor} \end{bmatrix} \quad \textbf{2.19}$$

Die obige Matrix kann vereinfacht und in die folgende Form gebracht werden [25]:

$$\frac{1}{n}\begin{bmatrix} \omega_n^0 & \omega_n^0 & \omega_n^0 & \cdots & \omega_n^0 \\ \omega_n^0 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{-(n-1)} \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ \omega_n^0 & \omega_n^{-(n-2)} & \omega_n^{-2\cdot(n-2)} & \cdots & \omega_n^{-(n-1)(n-2)} \\ \omega_n^0 & \omega_n^{-(n-1)} & \omega_n^{-2\cdot(n-1)} & \cdots & \omega_n^{-(n-1)(n-1)} \end{bmatrix} \cdot \begin{bmatrix} y_0 \\ \omega_n^2 y_1 \\ \vdots \\ \omega_n^{(n-2)\cdot 2} y_{n-1} \\ \omega_n^{(n-1)\cdot 2} y_n \end{bmatrix} = \begin{bmatrix} c_{-\lfloor \frac{n}{2} \rfloor} \\ -c_{-\lfloor \frac{n}{2} \rfloor + 1} \\ \vdots \\ -c_{\lfloor \frac{n}{2} \rfloor - 1} \\ c_{\lfloor \frac{n}{2} \rfloor} \end{bmatrix} \quad \textbf{2.20}$$

Für die Transkription von Musik gilt, dass y_j reell ist. Damit gilt, dass $\overline{y_j} = y_j$. Daraus folgt, dass $\overline{c_k} = c_{-k}$. c_k sind die Koeffizienten für das komplexe Fourierpolynom. Bei der DFT werden aus einer Multiplikation einer nxn-Matrix M und einem Vektor $\overrightarrow{v}$ n Koeffizienten berechnet.

Laufzeitbetrachtung

Für jeden der n Koeffizienten müssen n Multiplikationen durchgeführt werden: Für den j-ten Koeffizienten wird die j-te Zeile aus der Matrix M mit dem Vektor $\overrightarrow{v}$ mit der Länge n multipliziert. Diese n berechneten Werte werden anschließend summiert, sodass $n-1$ Additionen für die Berechnung eines Koeffizienten c_j, $n+(n-1)$ Operationen von Nöten sind. Dies wird für alle n Zeilen der Matrix durchgeführt. Damit ergibt sich eine Laufzeit von $T(n) = O(n(n+(n-1))) = O(2n^2-n) = O(n^2)$.

Da eine Echtzeitanwendung implmentiert werden soll, in der regelmäßig Fourier-Transformationen berechnet werden sollen, ist eine Verbesserung der Laufzeit erwünscht. Dies führt zur Fast-Fourier-Transformation (FFT).

2.4.3 Fast Fourier-Transformation (FFT)

Die DFT beruht auf der sturen Berechnung der Matrixmultiplikation. Damit die Berechnung der Fourieranalyse beschleunigt werden kann, entwickelten Cooley und Tukey 1965 den Radix-2-Algorithmus [26]. Es handelt sich bei dem Verfahren um

eine Beschleunigung der Multiplikation von Matrizen mit Vektoren, falls die Matrizen eine bestimmte Form aufweisen. Es wird dabei eine Matrix M mit einem Vektor $\overrightarrow{v}$ multipliziert werden, wobei folgende Bedingungen gelten müssen [14]:

- M ist nxn-Matrix
- n ist eine Zweierpotenz
- Die Einträge v sind n-te Einheitswurzeln

Zur Beschleunigung der FFT wurden Symmetrien in der Matrix 2.20 ausgenutzt. Dadurch ist die gesamte Matrix duch ein Viertel der Einträge determiniert [14]. Aus dieser Erkenntnis wurde der Radix-2 Algorithmus hergeleitet.

Ablauf

Zur Veranschaulichung des Verfahrens wird im Folgenden ein Schmetterlingsgraph verwendet [27]:

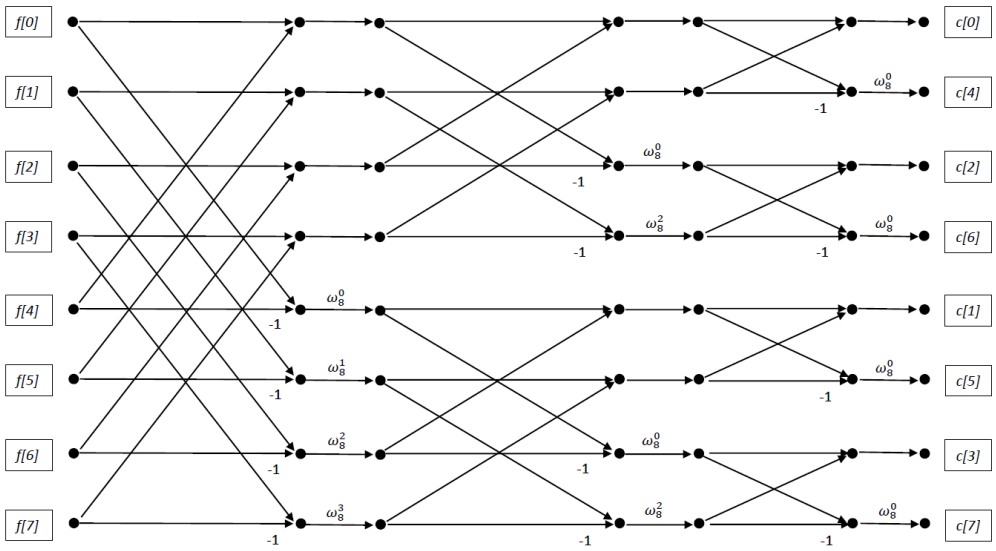


Abbildung 2.10: Schmetterlingsgraph für die FFT auf eine Datenmenge von $n = 8$

In Abbildung 2.10 ist das Radix-2 Verfahren anhand eines Schmetterlingdiagramms dargestellt. Als Eingabe dient ein Vektor f der Länge n, der n Samples der zu approximierenden Funktion $f(x)$ enthält. Als Ausgabe resultiert ein Vektor c, der die Werte der Koeffizienten c_k enthält. Diese beschreiben den Anteil der jeweiligen Sinus- und Kosinusfunktionen vom Grad k an der Gesamtschwingung $f(x)$.

Laufzeit-Betrachtungen

Mithilfe des Schmetterlingdiagramms 2.10 lässt sich auch die Laufzeit über die verschiedenen Rekursionsschritte abschätzen. Dies soll folgende Abbildung verdeutlichen:

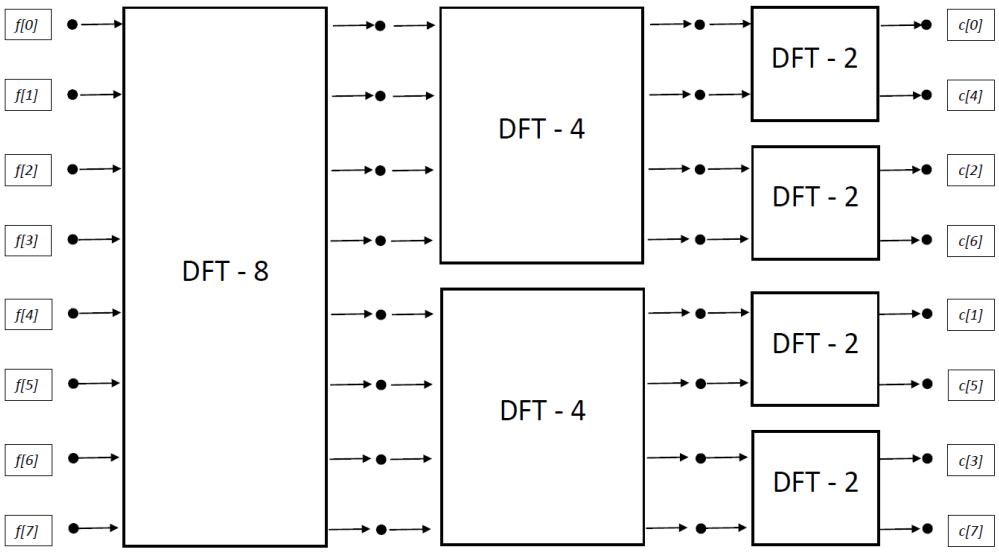


Abbildung 2.11: Verdeutlichung des Aufwands in jedem Rekursionsschritt zum besseren Verständnis der Laufzeit.

Anhand 2.11 zeigt sich dass die FFT auf eine rekursiven Lösung der Matizenmultiplikation besteht, wobei in jedem Rekursionsschritt die DFT lediglich auf die Hälfte der Einträge angewendet werden muss. Die asymptotische Lafuzeit kann also durch folgende Rekursionsgleichung angegeben werden:

$$T(n) = 2T(\frac{n}{2}) + n \qquad \textbf{2.21}$$

Mithilfe des Mastertheorems lässt sich die Laufzeit abschätzen. Diese Rekursionsgleichung fällt unter den zweiten Fall des Mastertheorems [28]:

$$\begin{aligned} a = 2, b = 2, f(n) = n \\ \longrightarrow f(n) = n^1 = n^{log_2(2)} = n^{log_b(a)} \\ \longrightarrow T(n) \in \Omega(n^{log_b(a)} log(n)) \\ \longrightarrow T(n) \in \Omega(nlog(n)) \end{aligned}$$

Bei genauerer Betrachtung des Verfahrens mithilfe von 2.10 ist erkennbar, dass pro Rekursionsschritt $\frac{n}{2}$ Additionen, $\frac{n}{2}$ Subtraktionen und $\frac{n}{2}$ Multiplikationen benötigt werden. Beim Radix-2-Algorithmus werden $log_2(n)$ Rekursionsschritte benötigt, um das Problem atomar aufzuteilen. Es ergibt sich also eine Notation von

$T(n) = O(3\frac{n}{2} \cdot log_2(n)) = O(nlog(n))$. Um die berechneten Faktoren den entsprechenden Koeffizienten zuzuordnen entspricht einer einmaligen Operation am Ende der DFT Berechnung. Dieser Aufwand gehört nicht zur eigentlichen Berechnung und wird an dieser Stelle vernachlässigt.

2.4.4 DFT vs. FFT

Beim Vergleich der konkreten Laufzeiten im praktischen Einsatz, zeigt sich die tatsächliche Beschleunigung des FFT-Verfahrens [14]:

Samples n	DFT $2n^2 - n$	FFT $3\frac{n}{2}log_2(n)$	relative Anteil %
8	120	36	30
256	130816	3072	2,34833659
1024	2096128	15360	0,73277968
2048	8386560	33792	0,4029304
4096	33550336	73728	0,21975339
8192	134209536	159744	0,11902582

Tabelle 2.2: Vergleich der tatsächlichen Laufzeit in Abhängigkeit von der Anzahl n der Messwerte.

Die in der Tabelle 2.2 eingetragenen Werte verdeutlichen welche Einsparungen die FFT ermöglicht. Es sind dabei Messwertanzahlen in der Größenordnung, wie sie in der Applikation verwendet werden aufgetragen. Besonders für große Samplegrößen n sind die Laufzeit-Einsparungen durch die FFT deutlich.

Neben dem ursprünglichen Radix-2-Verfahren von Cooley und Tukey, sind im Laufe der Zeit weitere Algorithmen für die FFT entstanden. Der Bluestein-Algorithmus (Chirp-Z-Transformation) ermöglicht zum Beispeil eine FFT für eine Datenmenge beliebiger Größe[29]. Damit würde die Bedingung wegfallen, dass die Datenmengen-Größe immer eine Zweierpotenz aufweisen muss.

2.4.5 Übergang in die Audiodatenverarbeitung

An dieser Stelle sollen die erlangten mathematischen Erkenntnisse in die Praxis übertragen werden. Ein Audiosignal wird von einem Mikrofon mit einer bestimmten Samplerate Fs abgetastet. Es ist weiterhin eine Samplegröße n festgelegt. Das Mikrofon liefert damit n Audiorohdaten alle $\frac{n}{Fs}$ Sekunden. Diese n Audiodaten entsprechen dem Vektor f. Der Vektor kann in eine FFT gegeben werden. Die FFT-

Implementierung [30], die im entwickelten System verwendet wird, liefert nicht direkt den Koeffizientenvektor c. Es werden zwei Vektoren zurückgegeben, wobei einer die Koeffizienten a_k der Kosinusschwingungen (Realteil) trägt und der andere die Koeffizienten b_k der Sinusschwingungen (Imaginärteil). Um die einzelnen Einträge des Koeffizientenvektor c aus dem Realteil a_k und dem Imaginärteil b_k zu berechnen, kann folgende Formel verwendet werden [31]:

$$|c_k| = \sqrt{(c_k)_{re}^2 + (c_k)_{im}^2} = \sqrt{(a_k)^2 + (b_k)^2} \quad \textbf{2.22}$$

Als nächstes wird die Bedeutung des Koeffizientenvektors c und seiner Einträge c_k (Frequency Bins) besprochen.

2.4.6 Koeffizienten-Vektor

Der Koeffizientenvektor c enthält die relativen Anteile der einzelnen Kosinus- und Sinusschwingungen am Gesamtsignal f. Die einzelnen Einträge c_k aus c werden Frequency Bins genannt. Die Werte, die in den Frequency Bins stehen können über Formel 2.3 in Lautstärken in dB umgewandelt werden. Da alle Funktionen als Bedingung für die FFT auf 1 normiert sind, gilt für die Referenzintensität $F_0 = 1$. Die Lautstärke einer Frequency Bin berechnet sich somit über:

$$L_k = 20 \cdot log_{10}(c_k) \text{ dB} \quad \textbf{2.23}$$

Die Samplingrate Fs des Mikrofons definiert über das Nyquist-Theorem einen maximal auflösbaren Frequenzbereich $[0; \frac{Fs}{2}]$Hz. Dieser Frequenzbereich wird gleichmäßig über c verteilt. Der Frequenzbereich wird damit äquidistant auf die einzelnen Frequency Bins c_k verteilt, sodass jede Frequency Bin denselben Frequenzbereich abdeckt. Über die Formel 2.24 kann die einer Frequency Bin zugeordnete Frequenz berechnet werden. Dabei steht der Index k für die k-te Frequency Bin im Koeffizientenvektor c.[32]

$$f(c_k) = k \cdot \frac{Fs}{n} \quad \textbf{2.24}$$

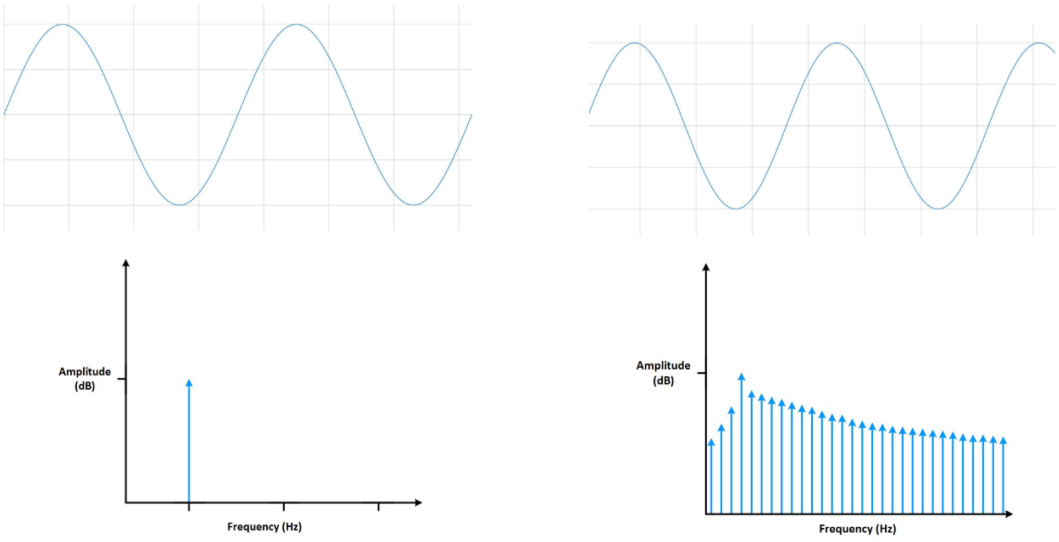
Wird die Anzahl n der Messwerte erhöht, erhält die FFT also eine größere Anzahl an Messwerten, wodurch ein größerer Ausgabevektor c entsteht. Da sich die einzelnen Bins gleichmäßig über den gesamten Frequenzbereich verteilen, deckt jedes c_k ein kleineres Frequenzintervall $\frac{Fs}{n}$ ab. Die Frequenzauflösung der FFT wird dadurch erhöht. Weiterhin wird bei größerer Samplegröße n die zeitliche Auflösung gemindert. Bei einer Samplegröße n und einer Samplerate Fs deckt die FFT eine Zeitspanne von $\frac{n}{Fs}$ Sekunden ab.

2.4.7 Fenster-Verfahren

Alle Daten, die in eine FFT gegeben werden, entsprechen dem sogenannten Fenster (Window). Ein Fenster sind alle betrachteten Datenpunkte zu einem bestimmten

Zeitpunkt in der Messung. Das Fenster weist eine Größe auf, die der Samplegröße n entspricht. Je größer das Fenster und damit die Samplegröße ist, desto rechenintensiver wird die FFT. Wie bereits erwähnt, ist die Wahl der Größe des Fensters eine wichtige Designentscheidung für das finale System.
Bei der Anwendung der FFT wird von einem periodischen Signal ausgegangen, welches eine ganzzahlig vielfache Frequenz der Grundfrequenz aufweist. Im gemessenen Intervall wird davon ausgegangen, dass die Endpunkte zwischen zwei Messungen jeweils verbunden sind. Damit ist die FFT genau dann ideal, wenn eine ganzzahlige Anzahl an Perioden das Messintervall besetzen.

In der Realität besitzen die gemessenen Audiosignale allerdings keine ganzzahlige Periodenzahl. Dies resultiert in einem abgeschnittenen Signalverlauf, welches zu scharfen Übergängen im gemessenen Signal führen kann. Sind die jeweiligen Signale also keine ganzzahligen Perioden, so sind die Endpunkte nicht kontinuierlich. Diese Diskontinuitäten werden in der FFT als Hochfrequenz-Komponenten angezeigt, welche im eigentlichen Signal nicht enthalten sind. Dabei können diese Frequenzanteile höher als die Nyquistfrequenz sein. Dies führt wiederum zu einer Spiegelung in den niederfrequenten Bereich zwischen 0Hz und der Hälfte der Samplingrate. Damit sind im Spektrum Frequenzen enthalten, die nicht im ursprünglichen Signal zu finden sind. Die Energie einer Frequenz scheint dadurch in unterschiedliche Frequenzen zu verlaufen. Das Phänomen wird spektrale Streuung genannt, da die feinen Spektrallinien als breitere Signale dargestellt werden. [33]
Dieser Sachverhalt ist in Abbildung 2.12 abgebildet.



(a) Das Messen einer ganzzahligen Anzahl von Perioden (oben) resultiert in einer idealen FFT (unten).[33]

(b) Das Messen einer nicht ganzzahligen Periodenanzahl (oben) resultiert in spektraler Streuung in der FFT (unten).[33]

Abbildung 2.12: FFT ohne (links) und mit (rechts) spektraler Streuung.

Damit die Streuungseffekt verringert werden können, wird das so genannte Fensterverfahren angewendet. Dabei wird auf die Messwerte eine Fensterfunktion angewendet, welche die Messwerte dahingehend verändert, dass diese an den Enden des Messintervalls graduell gegen Null streben. Damit gehen die Endpunkte zweier Messungen nahtlos ineinander über, ohne Diskontinuitäten zu verursachen. Durch das Fensterverfahren wird der Effekt der Phantom-Frequenzen gemindert und die spektrale Streuung wird geringer. Der Effekt des Fensterverfahrens ist in Abbildung 2.13 demonstriert.

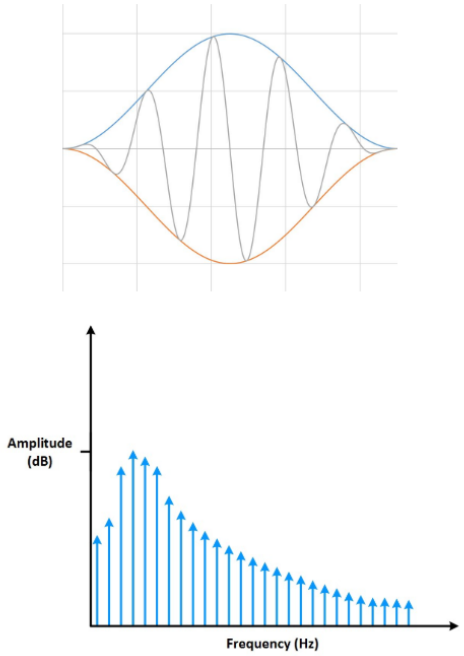


Abbildung 2.13: Fensterverfahren zur Minimierung der spektralen Streuung.[33]

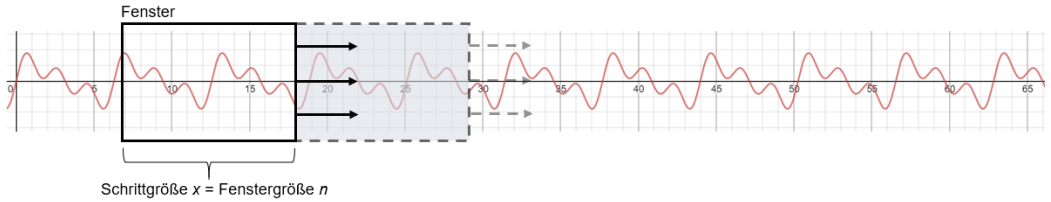
Das Fensterverfahren wird über so genannte Fensterfunktionen realisiert. Die bekanntesten Fensterfunktionen sind das Rechteckfenster, das Hann-Fenster, das Hamming-Fenster, das Blackman-Harris Fenster, das Kaiser-Fenster und das Flattop-Fenster [34]. Unterschiedliche Funktionen weisen verschiedene Effekte auf, die für verschiedene Ziele angepasst werden können.[33]

2.4.8 Short-Time Fourier-Transformation (STFT)

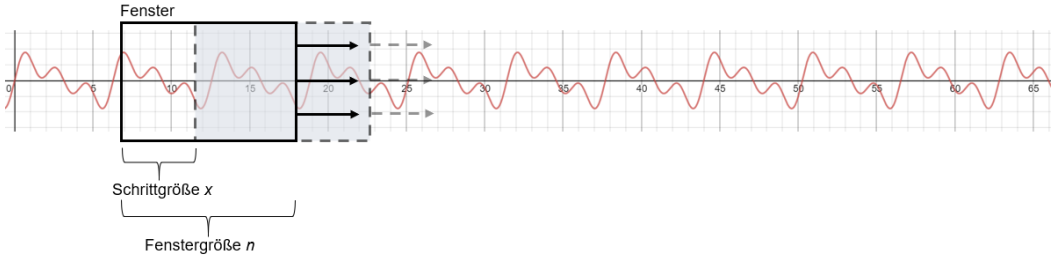
Mit der FFT ist es möglich bei Steigerung der Samplegröße n die Frequenzauflösung zu steigern. Damit geht allerdings auch immer mehr Information über die Zeit verloren, da bei einer konstanten Samplingrate Fs ein immer größerer Zeitabschnitt betrachtet wird. Die Zeitauflösung verringert sich also bei besserer Frequenzauflösung.

Für die Erkennung von Melodien ist es besonders wichtig Tonlängen und abrupte Änderungen im Signal erkennen zu können, weshalb auch auch eine entsprechende Auflösung auf der Zeitachse erfolgen muss. Es muss damit ein Weg gefunden werden, um sowohl eine ausreichende Zeit-, wie auch Frequenzauflösung zu erreichen. Dabei soll eine gewünschte Frequenzauflösung und damit eine bestimmte Samplinggröße n nicht verringert werden.

Eine bessere Zeitauflösung soll dadurch gegeben sein, dass das Fenster graduell verschoben wird. Um diesen Effekt zu ermöglichen, wird eine Schrittgröße x eingeführt, welche das Fenster immer um x Samples fortbewegt und anschließend die FFT auf das Fenster anwendet. Als Resultat wird nicht alle $\frac{n}{Fs}$ Sekunden eine FFT durchgeführt, sondern alle $\frac{x}{Fs}$ Sekunden. Die Samplegröße wird dabei nicht verändert. Als Resultat wird die FFT, bei gleichbleibender Samplegröße n, öfter ausgeführt. Damit wird die Zeitauflösung bei gleichbleibender Frequenzauflösung verbessert. Diese Umsetzung der Fouriertransformation wird auch Short-Time Fourier Transformation (STFT) genannt und ist in Abbildung 2.14b abgebildet. Zum Vergleich wurde das ursprüngliche Konzept in Abbildung 2.14a gegenüber gestellt.[35][36]



(a) FFT-Verfahren



(b) STFT-Verfahren

Abbildung 2.14: Gegenüberstellung des (a) ursprünglichen FFT-Verfahrens und des (b) STFT-Verfahrens.

In der Realität muss ein Kompromiss zwischen Schrittgröße x und Samplegröße n gefunden werden. Eine größere Samplegröße n führt zu einem größeren Rechenaufwand pro FFT und eine kleinere Schrittgröße x führt zu einer gesteigerten Anzahl an FFT-Berechnungen pro Zeiteinheit.

2.4.9 Abschließende Bemerkungen

Im Laufe des Kapitels wurden mehrere Parameter eingeführt, die für die Umsetzung der Transkription von Musik auf einem technischen System von Bedeutung sind. Diese Größen sollen nun im Folgenden zusammengetragen werden:

- Sample-Rate Fs: wird für den weiteren Verlauf fest als 44100Hz definiert.
- Sample-Größe n: Je größer n ist, desto genauer ist die Frequenzauflösung, desto ungenauer wird jedoch die Zeitauflösung und desto rechenintensiver die FFT.
- Schritt-Größe x: Je kleiner x ist, desto öfter wird die FFT pro Zeiteinheit ausgeführt und desto besser wird die Zeitauflösung. Mit einer höheren Anzahl an FFT-Berechnungen nimmt allerdings auch der Rechenaufwand für das verwendete technische System zu.
- Windowing-Funktion f_{win}: Unterschiedliche f_{win} führen zu unterschiedlichen FFT-Ergebnissen und damit zu unterschiedlichen Ergebnissen der Transkription der Musik.
- Frequenzintervall Δf in cent: Je kleiner Δf gewählt wird, desto genauer muss die Frequenzauflösung der FFT sein und desto größer muss die Sample-Größe n sein. Dies führt erneut zu einem größeren Rechenaufwand für das technische System.
- Tempo *tempo*: Je größer das Tempo gewählt wird, desto kürzer ist die Dauer eines Notenwertes und desto rechenaufwendiger wird es Töne mit einem niedrigen Notenwert zu identifizieren.
- Notenwertauflösung: Je höher die Notenwertauflösung sein soll, desto mehr FFT-Berechnungen müssen pro Zeiteinheit ausgeführt werden. Dadurch muss n, x und *tempo* entsprechend angepasst werden.

Aus diesen Größen gilt es solche Werte zu ermitteln, welche eine ausreichende Frequenz- und Zeitauflösung aufweisen und gleichzeitig eine laufzeiteffiziente Möglichkeit für das ausführende technische System bieten.

2.5 Digitale Verarbeitung

Das System soll später auf einem digitalen Endgerät ausführbar sein. Als solches muss es Audiodaten verarbeiten und digital darstellen können. Aus diesem Grund soll an dieser Stelle ein Exkurs in den Bereich der digitalen Verarbeitung von Audiodaten unternommen werden.

Mikrofon & PCM

Ein Mikrofon kann ein akustisches Signal abtasten und und digitalisieren. Das zeit- und wertkontinuierliche Signal muss dabei in ein zeit- und wertdiskretes Signal umgewandelt werden. Dazu wird das Pulse-Code Modulationsverfahren (PCM) auf das Audiosignal angewendet. Mit einer zeitlich konstanten Abtastrate wird zunächst das akustische Signal in ein zeitdiskretes und wertkontinuierliches Signal transformiert. Eine Quantisierung der gemessenen Werte auf endlich viele Stellen liefert eine wert- und zeitdiskrete Darstellung des Signals. Die Audiodaten können anschließend digital verarbeitet werden.[37]

Audiospeicherformate

Audiodaten können prinzipiell in unterschiedlichen Formaten gespeichert werden. Dabei gibt es einerseits die Möglichkeit die Audiorohdaten zu speichern, andererseits können die Audiodaten komprimiert gespeichert werden. Sollen Rohdaten gespeichert werden, so wird das durch das Mikrofon aufgezeichnete Audiosignal ohne Veränderung gespeichert. Der bekannteste Vertreter dieser Speicherart ist die WAV-Datei. Im Bereich der Audiokompression gibt es beispielsweise das MP3-Format. Hierbei handelt es sich um verlustbehaftete Aufzeichnungen eines Audiosignals. Es kann dabei eine bis zu 80-prozentige Verringerung der Dateigröße erreicht werden, allerdings verlieren die Audiodaten damit an Qualität. Es werden deshalb größtenteils die nicht hörbaren Audioanteile herausgefiltert.[38]

WAV-Datei

Damit Audiosignale digital verarbeitet werden können, soll ein direkter Zugang zu den Audiorohdaten ermöglicht werden. Das PCM-Verfahren [37] ermöglicht dem Mikrofon Audiorohdaten zu digitalisieren. Mithilfe einer WAV-Datei kann dieses Datenformat abgespeichert werden.

Die WAV-Datei beginnt mit einem Header, über den die einzelnen Eigenschaften des ursprünglichen Audiosignals bestimmt werden kann. Auf den Header folgen anschließend die eigentlichen Audiorohdaten. Tabelle 2.3 zeigt die Bedeutung der einzelnen Felder einer WAV-Datei, deren Größe und Verwendungszweck [39][40][41][42].
Der Kopf einer WAV-Datei weist eine Größe von 44 Byte auf. Aus den einzelnen Positionen der Felder lässt sich akkumulativ die Startposition des jeweiligen Feldes bestimmen. Die WAV-Datei liegt selbst wieder als Resource Interchange File Format (RIFF) vor. Dabei handelt es sich um ein Containerformat zur Speicherung von Multimedia-Daten.[43]
Für eine Verarbeitung der Audiodaten können aus dem WAV-Header verschiedene Eigenschaften des aufgenommen Audiosignals gelesen werden. Besonders interessant für die digitale Verarbeitung von WAV-Dateien ist die Dateigröße, die Kanalanzahl,

die Anzahl Bits pro Sample und die Samplingrate. Die Dateigröße geteilt durch die Angabe der Bytes pro Sekunde ergeben die Aufnahmedauer des in der WAV-Datei einthaltenen Audiosignals. Die Kanalanzahl muss bekannt sein, um die Verarbeitung der WAV-Datei auf mehrere Kanäle anzupassen.

Position	Beispielwert	Beschreibung
1-4	"RIFF"	Deklariert die Datei als RIFF-Datei.
5-8	Dateigröße	Dateigröße abzüglich der ersten 8 Bytes.
9-12	"WAVE"	Dateityp.
13-16	"fmt"	Es folgen Felder, die das Format spezifizieren.
17-20	16	Länge des Format Chunks.
21-22	1	Datenformat. 1-pcm, 6-alaw, 7-mulaw
23-24	2	Kanalanzahl. 1-mono, $\geq$ 2-stereo
25-28	44100	Abtastrate, Anzahl an Samples pro Sekunde.
29-32	176400	Bytes pro Sekunde - $Abtastrate \cdot FrameSize$
33-34	4	Bytes pro Sample - $(BitsPerSample \cdot Channels)/8$
35-36	16	Bits pro Sample.
37-40	"data"	Markiert den Beginn des Audiodatenbereichs.
41-44	Data File size	Größe des Datenabschnitts.

Tabelle 2.3: Bedeutung der einzelnen Felder im WAV-Datei-Header.

MIDI-Datei

Der Begriff MIDI steht für Musical Instrument Digital Interface. MIDI Dateien dienen zum Austausch musikalischer Notation zwischen computergesteuerten Instrumenten und Computern. In einer MIDI Datei werden dazu musikalische Informationen exakt kodiert, sodass diese auf einer entsprechenden Hardware ausgeführt werden können.[44]
Midi-Dateien können in Musikeditierungsprogramme (MuseScore,...) geöffnet werden und dort manuell weiterverarbeitet werden.[45]

LilyPond

LilyPond ist ein freies Notensatzprogramm, welches mithilfe von textuellen Eingaben dazu in der Lage ist ein Notenblatt zu generieren. Der Grund für die Wahl von LilyPond als Notengenerierungsprogramm liegt in der einfachen Erzeugung von Partituren, die auch programmatisch möglich ist. Gerade zur Automatisierung der entstehenden Pipeline ist es nützlich ein Tool zu besitzen, welches über Kommando-

zeilenbefehle bedient werden kann. Ein weiterer Grund für die Wahl von LilyPond als Notengenerierungssystem ist die Tatsache, dass das System mitunter das Bekannteste seiner Art ist. Es gibt sowohl LilyPond Installationen für Unix(GNU/Linux, FreeBSD), MacOS wie für Windows. [46]

Die Generierung eines Notenblattes als PDF, sowie einer MIDI Datei, wird über ein LilyPond Datei ermöglicht. Dabei handelt es sich um eine Textdatei, in der die gewünschte Melodie als Zeichenkette notiert ist. Weiterhin kann über das Setzen von bestimmten Steuersignalen das Aussehen und die Eigenschaften des Notenblattes verändert werden. Dieses Framework ermöglicht es die aufgenommene Melodie über eine Zeichenkette in musikalische Notation umzuwandeln. Zur Ermöglichung dieser Funktionalität wurden einige Funktionen im Laufe der Bachelorarbeit geschrieben werden.[46] Im Folgenden soll der Beispielinhalt einer LilyPond-Datei dargestellt werden.

```
\version "2.18.2"
\header { title="Title" subtitle="Subtitle" composer="Composer" }
\score {
  \new Staff
  \with {instrumentName = "trumpet" }
  {
    \set Staff.midiInstrument = #"trumpet"
    \absolute {
      \clef treble
      \key c
      \major
      \time 4/4
      \tempo "Allegro" 4 = 90
      c''4~ c''4 r2 g,2 <c' e' g'>4
    }
  }
  \layout {}
  \midi {}
}
```

Das LilyPond-Framework muss in der Datei eine bestimmte Form vorfinden, damit ein Notenblatt generiert werden kann. Als erstes wird die LilyPond-Version angegeben. Anschließend folgt eine Headerzeile, in der Metainformationen, wie der Titel des Musikstücks, den Namen des Komponisten und viele weitere Optionen hinterlegt sind. Darauf folgend wird über *score* eine Partitur eingeleitet. Es wird für das Notenblatt ein Instrument festgelegt. In diesem Beispiel handelt es sich um eine Trompete. Über *new Staff* wird eine neue Stimme/Instrument angelegt. In den geschwungenen Klammern werden sämtliche Informationen spezifiert wie Tonart, Taktart und Tempo.

Weiterhin wird die eigentliche Melodie in LilyPond-Notation eingetragen. Dabei bestehen die einzelnen Notenwerte jeweils aus dem Halbton (*c,cis,d,...*), der Tonhöhe (*...;„;,; ;';";..*), gefolgt von der Tonlänge (*1,2,4,8,...*). Pausen werden über *r* signalisiert und Bindebögen mit . Akkorde können in eckigen Klammern notiert werden (<>).
Am Ende der Datei wird spezifiziert, dass durch den Kompiliervorgang neben einer PDF (*layout*) auch eine MIDI-Datei (*midi*) erzeugt werden soll. Weitere Informationen zu Konventionen und Syntax des LilyPond-Formats sind online erhältlich.[47][48]

3 Stand der Forschung

Alle Signale, die über Sinnesorgane erfasst werden können, bieten grundlegend ein großes Potenzial zur digitalen Verarbeitung, da diese mit einem digitalen Äquivalent eingefangen und von einem technischen System verarbeitet werden können. Das Äquivalent zu den Augen ist eine Kamera, das Äquivalent zu den Ohren ist das Mikrofon.
Die Bedeutung von Musik lässt sich durch die Geschichte verfolgen. Ein Großteil der Bevölkerung kommt zumindest in den Kontakt mit Musikinstrumenten [49]. Die Wahrscheinlichkeit ist damit sehr hoch, dass jede Person eine andere Person kennt, die ein Instrument spielt.
Der Einsatz digitaler Geräte in der Musik eröffnete neue Möglichkeiten zur Darstellung, Analyse und Transkription von Musik. Im Folgenden soll der aktuelle Stand der Forschung, in den für die Bachelorarbeit wichtigen Forschungsgebieten, betrachtet werden.

Digitale Signalverarbeitung

Die entwickelten Theorien und Techniken im Bereich der digitalen Signalverarbeitung ermöglichten es erst, analoge Signale digital zu repräsentieren, diese zu analysieren und zu verarbeiten [50]. Im Folgenden sollen nennenswerte Entwicklungen und Erkenntnisse aus diesem Forschungsbereich angesprochen werden.

Eine wichtige und vielfach verwendete Erkenntnis ist das Nyquist-Shannon Abtasttheorem. Dieses besagt, dass eine bestimmte Frequenz f durch ein digitales Medium lediglich dann rekonstruiert werden kann, falls die Abtastrate mit mindestens der doppelten Frequenz das Signal abtastet.[12]
Mikrofone müssen analoge wert- und zeitkontinuierliche Signale in eine digitale Repräsentation umwandeln können. Diese Transition ist über ein Pulse-Code Modulationsverfahren (PCM) möglich. Dabei entsteht zunächst ein zeitdiskretes und wertkontinuierliches Signal durch eine zeitlich konstante Abtastrate. Anschließend werden die aufgenommenen Werte durch eine Quantisierung diskretisiert. Damit entsteht eine digitale diskrete Repräsentation des Signals.[37].
Im Bereich der digitalen Signalverarbeitung geht es weiterhin um die Betrachtung von Sensoren, die eine bestimmte Signalart erfassen können. Es werden Techniken zur Verstärkung von aufgenommenen Signalen untersucht, sowie die Digitalisierung mithilfe eines Analog-Digitalwandler durchgeführt. Erst nach diesen Schritten, ist es möglich das jeweilige Signal digital zu verarbeiten. [50]

Die an dieser Stelle beschriebenen Techniken und Erkenntnisse müssen für die Bachelorarbeit nicht explizit verstanden sein, da diese durch das Mikrofon oder die Audioschnittstelle gehandhabt werden. Damit ist das genaue Verständnis für den Programmierer nur von geringer Bedeutung. Dennoch sollte das Kapitel zur Vollständigkeit an dieser Stelle angeführt werden.

3.1 Audioanalyse

Das Hauptaugenmerk dieses Forschungsgebiets ist die Extraktion und Filterung von bestimmten Information aus einem Audiosignal. Als einfachste Möglichkeit kann dazu ein Audiospektrogramm verwendet werden. Dieses zeigt den jeweiligen Anteil von Einzelfrequenzen an einem aufgenommenen Signals. Die Frequenzwerte für die einzelnen Schwingungen verändern sich dabei über die Zeit.[51]
Es werden zum Teil bereits Theorien des Maschinellen Lernens verwendet, um bestimmte Attribute aus dem Signal herauszufiltern.[52]. Weiterhin werden für solche Aufgaben auch neuronale Netze eingesetzt [53].
Neben der Forschung werden die hier besprochenen Techniken bereits in kommerziellen Produkten verwendet [54]. Darunter befinden sich auch Digitale Audio Workstations wie *Audacity* [55].

3.2 Kompositions-Tools

Durch die Einführung des Computers im Bereich der Musik können Tools entwickelt werden, die den Prozess der Komposition übernehmen und den Workflow im Vergleich zum händischen Prozess beschleunigen. Die bisher etablierten Tools für den Kompositionsprozess verlagern den händischen Prozess auf die Maus und die Tastatur zur Manipulation des Notenblattes nach den Wünschen des Komponisten. Diese Musikeditierungsprogramme können die Arbeitsgeschwindigkeit des Komponisten zwar beschleunigen und bieten eine digitale Möglichkeit für das Komponieren. Allerdings verfügen diese Anwendungen oft auch über eine steile Lernkurve. Erwähnenswerte Tools sind zum Beispiel *MuseScore* oder *Sibelius* [1].
Als mobile Anwendung ist *NotateMe* für das IPad entstanden. Bei dieser Anwendung ist es möglich über einen Display-Stift Notenwerte zu zeichnen, die anschließend von einem internen KI-System erkannt und korrekt im Notensystem platziert werden.[56]
Eine weitere Möglichkeit ist die Verwendung eines MIDI-Keyboards zur Eingabe der Melodie. Dabei wird die Melodie manuell eingegeben und über eine MIDI-Schnittstelle eingefangen und dadurch digital aufbereitet. Ein solches Verfahren ermöglicht eine exakte Repräsentation der gespielten Melodie ohne akustische Störgeräusche. Kompositionstools bieten häufig eine solche Schnittstelle.[1]
Die vorgestellten Techniken bieten damit eine digitale Kompositionsmöglichkeit, haben jedoch nur bedingt etwas mit Audioverarbeitung zu tun. Die Funktionsweise

der Werkzeuge beruht auf der manuellen Eingabe des Komponisten. Weiterhin weisen diese Anwendungen häufig eine steile Lernkurve auf, sodass die Benutzung zum Teil nur erfahrenen Musikern eröffnet wird. Auch wird zum Beispiel für die MIDI-Variante zusätzliche Hardware benötigt.

3.3 Tools für die Transkription

An dieser Stelle werden Projekte betrachtet, die versuchen eine automisierte Transkription von Musik durch Verarbeitung der Audiodaten zu erreichen. Die vorgestellten Anwendungen verfolgen damit ein ähnliches Ziel wie jenes dieser Bachelorarbeit. Durch diese Tools wird die Lernkurve für den Anwender reduziert und einem breiteren Publikum wird die Möglichkeit zur Komposition von Musik eröffnet.

Eine komplette Softwarelösung zur Transkription von Musik ist komplex, sodass viele Softwareanwendungen nur auf bestimmte Instrumente zugeschnitten sind oder weitere Einschränkungen bezüglich der Notation vornehmen müssen. Die Anwendung *Melody Scanner* adaptiert die verwendeten Algorithmen für jedes Instrument und führt damit einen zusätzlichen Konfigurationsschritt ein. Aktuell besitzt das Tool Lösungen für die Instrumente Klavier, Gitarre, Flöte und Geige. Dabei sind die Algorithmen auf eine große Mengen von Audiodaten solcher Instrumente trainiert. Für das Klavier erreicht die Anwendung eine Genauigkeit von bis zu 85%. Neben der Audioaufnahme bietet das Tool die Möglichkeit Audiodateien (MIDI,MP3,...) auf eine Cloudinfrastruktur hochzuladen und transkribieren zu lassen. Weiterhin kann das System Melodien über Künstliche Intelligenz komponieren.[57]
Die Tools *Songs2See*[58] oder *Audio Score Ultimate 8*[59] ermöglichen es eine Melodie zu transkribieren und an ein Keyboard-Layout anzupassen. Zur Optimierung des Ergebnisses wird dem Nutzer nach der Aufnahme die Möglichkeit gegeben die Tonlängen für die einzelnen Töne manuell anzupassen.
Die Anwendung *TwelveKeys*[60] wandelt zunächst eine Audiodatei in ein Audiospektrogramm um. Das entstehende Spektrogramm kann anschließend mit einem Keyboard-Layout synchronisiert werden. In meinem jetzigen Verständnis kann diese Anwendung nicht zur Generierung eines Notenblattes verwendet werden, sondern soll dem Musiker lediglich das Erlernen oder Analysieren einer Melodie ermöglichen. Auch die Software-Anwendung *Transcribe!*[61] dient lediglich zur Erkennung von Noten auf einem Keyboard, erzeugt allerdings kein Notenblatt. Die Techniken, die hier verwendet werden, sind allerdings dennoch von Interesse, da die eigentliche Transkription bereits durchgeführt wird. Es fehlt lediglich die Erstellung eines Notenblattes.
Bestimmte Anwendungen versuchen bereits intelligente Verfahren, wie neuronale Netze für die Transkription einzusetzen, um mit der Komplexität von Musik umzugehen. *Anthemscore*[62] verwendet neuronale Netze für die Transkription von Musik und ermöglicht dadurch Melodien bestehend aus mehreren Stimmen zu transkribieren. Dies vereinfacht weiterhin die Komposition von Musikstücken, da nicht für jede Melodie einzeln transkribiert werden muss.

Die bereits vorgestellten Ansätze für die Transkription von Musik beruhen auf einer Nachverarbeitung von Audiodateien. Für Echtzeitanwendungen oder der Einsatz auf eingebetteten und mobilen Systemen stellt das Laufzeitverhalten bei rechenintensiven Algorithmen ein Problem dar. Dadurch kann die Qualität der Transkription in Mitleidenschaft gezogen werden. Aufgrund dieser Tatsache setzen die vorgestellten Anwendungen auf eine Nachverarbeitung einer Audiodatei, in der die Audiodaten gespeichert sind. Dadurch sind die Anwendungen größtenteils zeitunabhängig und können rechenintensive Algorithmen verwenden.

Durch die hier getätigte Analyse der bereits bestehenden Lösungen sollte eine Gesamteindruck vermittelt werden. Es zeigt sich, dass die besten Lösungen über intelligente Verfahren wie neuronale Netze verfügen. [63] stellt eine Beispielarchitektur für die Transkription von Musik vor, wobei zur Tonhöhen-, Tonlängen-, Akkord- und Instrumentendetektion musikalische trainierte Modelle verwendet werden sollen. Diese Methoden sind größtenteils als Desktop- oder Cloudanwendungen verfügbar und können daher unabhängig von der Berechnungszeit existieren. Weiterhin besteht die Transkription aus der Nachverarbeitung der Audiosignale. Es findet keine Echtzeitverarbeitung statt.

In dieser Bachelorarbeit soll ein Ansatz entwickelt werden, der eine Transkription von Musik in Echtzeit ausgehend von Audiosignalen ermöglicht und ein Notenblatt der erkannten Melodie generiert. Dabei wird bewusst auf intelligente Verfahren wie Methoden des Maschinellen Lernens oder Neuronalen Netzen verzichtet, da ein robustes System durch solche Methoden jederzeit erweitert werden kann.

4 Implementation

4.1 Entwicklungsumgebung

Die Entwicklung der Pipeline wurde auf einem (*Aspire VN7-593G*)-Laptop mit dem Betriebssystem *Ubuntu Version 18.04.2 LTS Bionic Beaver* durchgeführt. Der Laptop stellt dabei die performante Version dar und wurde neben der Entwicklung auch zum Testen des Systems verwendet. Der Rechner besitzt einen *Intel(R) Core(TM) i7-7700HQ CPU @2.80GHz* Intelprozessor. Die Audioaufnahme fand dabei durch die interne Soundkarte (*HDA-Intel - HDA Intel PCH*) statt.
Weiterhin sollte das System auch auf einem eingebetteten System ausführbar sein. Dazu wurde ein *Raspberry Pi 3 Model B Plus Rev 1.3* mit einem *Cortex-A53*-Prozessor verwendet. Das System verfügt über einen Arbeitsspeicher von 1 Gigabyte. Der Raspberry Pi sollte dabei als untere Schranke für die Performanzevaluation dienen. Zur Aufnahme der Audiosignale wurde ein USB Mikrofon (*C-Media Electronics Inc. USB PnP Sound Device*) verwendet.

4.2 Pipeline

Für die Transkription von Musik wurde im Zuge dieser Bachelorarbeit die Pipeline 4.1 entwickelt. Die Signale der Audioquellen (Gesang, Pfeifen, Instrumente,...) werden als erstes über ein Mikrofon erfasst und von diesem in eine digitale Repräsentation umgewandelt (*Audioaufnahme*). Das erfasste Signal wird zunächst für die FFT aufbereitet und über das STFT-Verfahren sowohl frequenztechnisch, wie auch zeittechnisch aufgelöst (*Audiovorverarbeitung*). Die in der Bachelorarbeit entwickelten Algorithmen führen die entsprechend ermittelten Frequenzanteile in eine musikalische Notation über (*Transkription*).

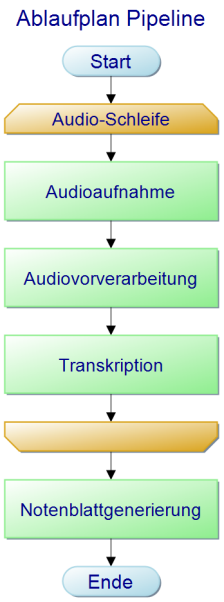


Abbildung 4.1: Pipeline zur Transkription von Musik.

Nach Ablauf der Pipeline generiert Lilypond aus der erkannten Melodie ein Notenblatt (*Notenblattgenerierung*). Die Abarbeitung der Pipeline kann dabei sowohl in Echtzeit-, wie auch über eine Audionachverarbeitung geschehen. Im Folgenden wird die Implementierung der einzelnen Komponenten der Pipeline genauer beschrieben.

Dabei kann die Granularität eines Pipelinedurchlaufs weiter verfeinert werden. Dies ist in Abbildung 4.2 ersichtlich. Zunächst nimmt das Mikrofon eine bestimmte Anzahl n an Messwerten auf und überführt ein zeit- und wertkontinuierliches Signal in eine zeit- und wertdiskrete digitale Darstellung. Diese Messwerte werden in eine FFT eingefüttert. Als Ausgabe wird ein Koeffizientenvektor c erhalten. Jede Frequency Bin c_k aus c kann in eine bestimmte Frequenz umgerechnet werden und gibt den Lautstärkeanteil der jeweiligen Einzelschwingung wieder. Der im Ausgangssignal enthaltene dominante Ton, trägt am meisten zum Gesamtsignal bei und weist daher die größte Amplitude auf. Der Wert in der entsprechenden Frequency Bin für die Einzelschwingung ist damit am größten. Über eine Maximumssuche kann die Frequency Bin mit dem größten Wert bestimmt werden und eine entsprechende Frequenz für die Frequency Bin ermittelt werden. Dadurch wurde die Frequenz der Einzelschwingung bestimmt, die den größten Anteil am Gesamtsignal trägt. Über die Tabelle 2.1 und die Formel 2.2 kann dieser Frequenzwert nun in einen Notenwert umgewandelt werden.

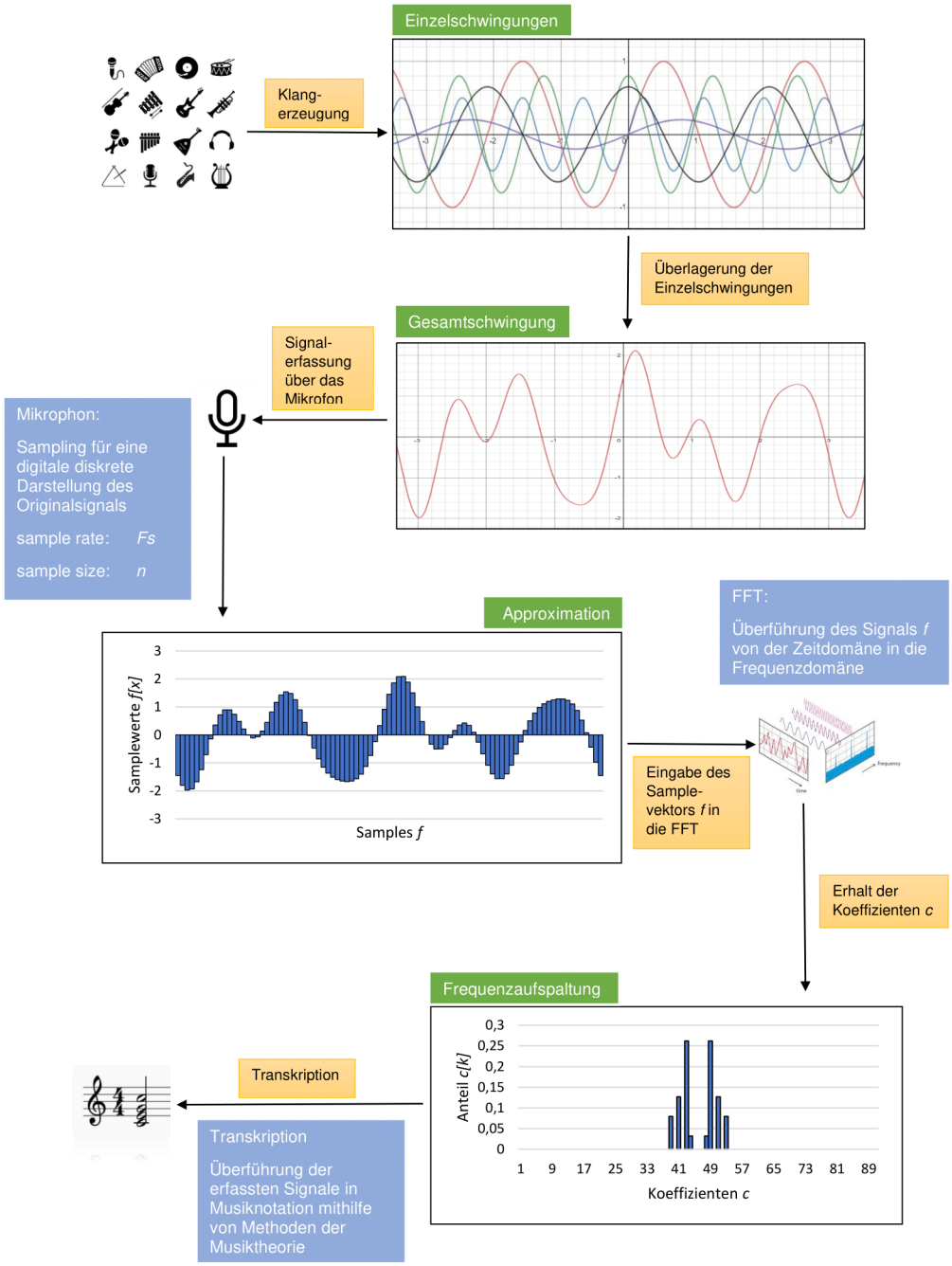


Abbildung 4.2: Einzelner Pipeline-Durchlauf.

4.2.1 Pipelinealgorithmus

Im Programmablaufplan 4.3 ist der Algorithmus für die gesamte Pipeline dargestellt. Für den Betrieb der Pipeline müssen sowohl die Sample-Rate *Fs*, die Sample-Größe n und die Schritt-Größe x für das STFT-Verfahren, sowie die Aufnahmezeit bekannt sein. Weiterhin gibt es den Sample-Vektor (*blauer Balken*), welcher die aufgenommenen Audiodaten in sich trägt. Die Pipeline wird so lange durchlaufen, bis die Aufnahmezeit überschritten wird.

Es wird zunächst der Sample-Vektor um x Messdaten vorangeschritten, sodass Platz für neue x Audiodaten entsteht. Gleichzeitig nimmt ein Buffer (*grüner Balken*) x Audiodaten über die Audioschnittstelle auf und schreibt diese an die frei gewordene Stelle im Sample-Vektor.
Anschließend wird eine Audiovorverarbeitung durchgeführt. Dabei wird eine Fensterfunktion auf die Audiodaten angewendet, sowie eine FFT durchgeführt. Weiterhin wird auch ein Frequenzbandpass durchgeführt, um für die Transkription irrelevante Frequenzen bereits zu Beginn herauszufiltern. Die Audiovorverarbeitung liefert den Koeffizientevektor, der in die Transkription gegeben wird.

Die Transkription liefert einen Notenwert. Falls dieser sich von dem zuletzt erkannten Notenwert unterscheidet, so handelt es sich um einen neuen Ton. Es kann die Tonlänge bestimmt werden und zusammen mit dem letzten Notenwert ein Notenwert in LilyPond-Notation generiert werden, welcher an die eigentliche Melodiezeichenkette angefügt wird.

Falls die Aufnahmezeit noch nicht überschritten ist, so startet die Pipeline in einen neuen Transkriptionsdurchlauf. Andererseits kann die Pipeline beendet werden und zusammen mit der Melodiezeichenkette kann über LilyPond ein Notenblatt generiert werden.

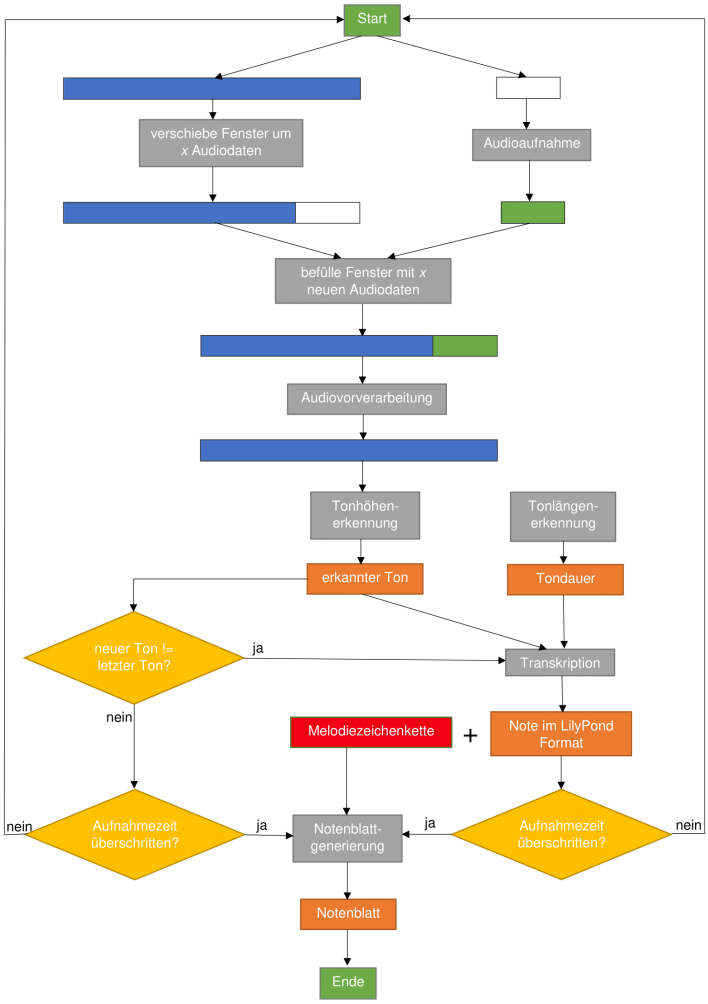


Abbildung 4.3: Pipeline-Algorithmus

Auf die einzelnen Komponenten der Pipeline wird im Folgenden genauer eingegangen.

4.2.2 Audioaufnahme

Bei der Audioaufnahme soll ein Audiosignal über ein Mikrofon eingefangen werden und eine Mögichkeit geboten werden diese Daten digital zu verarbeiten. Das Mikrofon tastet das Signal ab und bildet eine digitale Repräsentation des Signals und stellt damit die Audiorohdaten per PCM an der Audioschnittstelle einer entsprechenden Soundkarte dem Betriebssystem zur Verfügung.
Damit aus dem C-Programm auf die vom Mikrofon erfassten Audiosignale zugegriffen werden kann, muss eine Möglichkeit gefunden werden mit den Soundkarten

zu kommunizieren. Da sich beide Testsysteme (Laptop, Raspberry Pi) unter einem Linux-Betriebssystem befinden, kann dazu die *Advanced Linux Sound Architecture* (ALSA) herangezogen werden. Die Kommunikation zwischen dem C Programm und Schnittstelle der Soundkarte über ALSA ist in Abbildung 4.4 dargestellt.

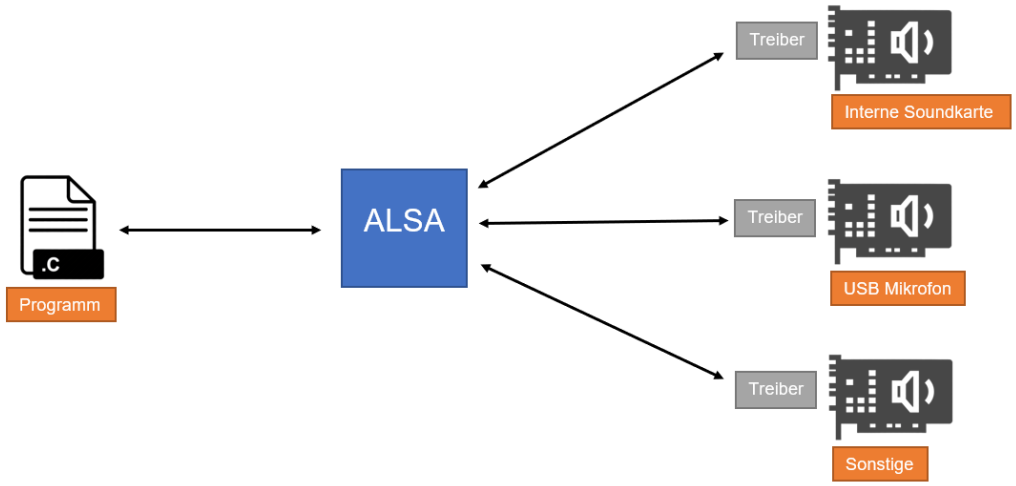


Abbildung 4.4: Darstellung der Interaktion zwischen dem C Programm und der Soundkartenschnittstelle über ALSA.

ALSA ermöglicht dem Programm mit Schnittstellen von Soundkarten zu interagieren. Dabei kommuniziert das Programm mit den Kernelmodulen (Treiber) der jeweiligen Soundkarte [64]. Damit kann das Programm aktiv Hardwareparameter wie die Samplingrate oder verwendete Anzahl an Kanälen für ein Mikrofon setzen. Bei der Ausführung der Pipeline auf dem Laptop werden die internen Mikrofone verwendet. Bei der Ausführung auf dem Raspberry Pi wird ein USB Mikrofon verwendet. Beide Mikrofone verwenden eine Samplingrate von $Fs = 44100$Hz. ALSA bietet weiterhin eine Möglichkeit die gesampleten Audiodaten gezielt von der Schnittstelle einer Soundkarte abzugreifen, in einem Buffer zu platzieren und dem Programm zur Verfügung zu stellen. Dieser Vorgang ist in Abbildung 4.5 dargestellt.

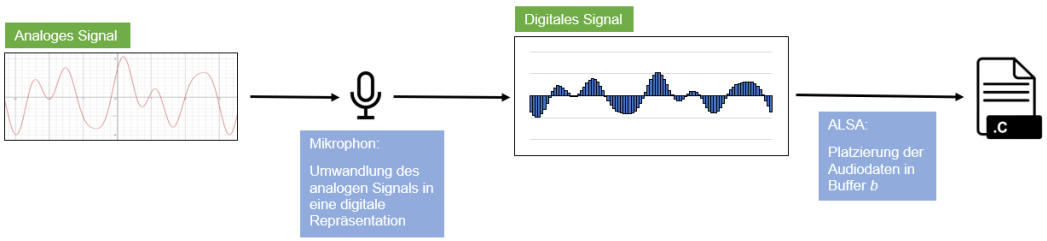


Abbildung 4.5: Audioaufnahme.

Als erstes wird über das Betriebssytem auf das ALSA Interface zugegriffen und dieses entsprechend konfiguriert. Dies ermöglicht es erst auf die Schnittstelle der Soundkarte aus dem Programm heraus zuzugreifen. Für diesen Zweck wurden bereits vorgefertigte Implementierungen verwendet.[65]
Anschließend wird ein Buffer definiert, welcher eine Größe entsprechend der Schrittgröße x aufweist. Der Buffer wird über die Interaktion mit ALSA mit x Audiomessdaten befüllt. Die neuen Daten werden dem restlichen Programm anschließend über den Buffer zur Verfügung gestellt. Über diesen Mechanismus kann das Programm regelmäßig Audiodaten von der Schnittstelle anfordern und für die Weiterverarbeitung verwenden.

4.2.3 Audiovorverarbeitung

Die Audiovorverarbeitung umfasst sämtliche Funktionen, die die aufgenommenen Daten für die eigentliche Transkription vorbereiten. Dazu gehört das Fensterverfahren, die Durchführung der FFT, eine Lautstärkenauswertung, sowie einem Frequenzbandpass. Ein zugehöriger Programmablaufplan ist in folgender Abbildung 4.7 dargestellt.

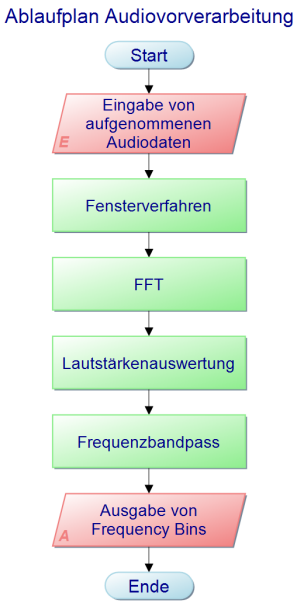


Abbildung 4.6: Programmablaufplan der Audiovorverarbeitung.

In der Audiovorverarbeitung werden die Daten in die benötigte Form für die Transkription gebracht und über eine Fouriertransformation von der Zeitdomäne in die Frequenzdomäne transferiert. Implementierungen zur Fouriertransformation wurden von [30] entnommen. Der für diese Funktionalität entwickelte Algorithmus ist in Al-

gorithmus **??** in Pseudocode aufgetragen.

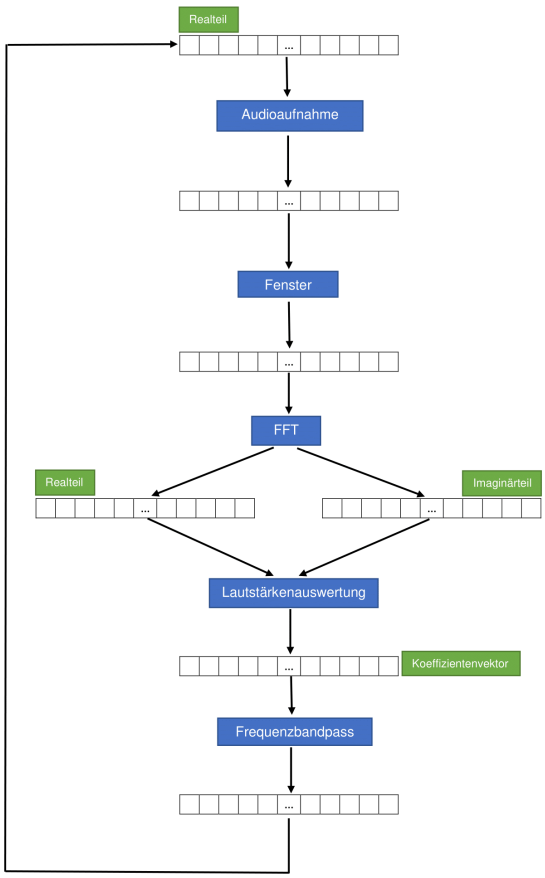


Abbildung 4.7: Audiovorverarbeitung

Es muss zunächst ein Fenster definiert werden, welches n Messwerte tragen kann. Dies wird über zwei Vektoren der Länge n ermöglicht, wobei einer für den Realteil und der andere für den Imaginärteil verantwortlich ist. In der Praxis stellen die über ein Mikrofon erfassten Audiodaten lediglich den Realteil dar, welcher in die FFT gegeben wird. Der imaginäre Anteil trägt dabei für jede Stelle den Wert 0. Er ist dennoch nötig, da der imaginäre Anteil über die FFT später befüllt wird.
Neue Audiodaten werden regelmäßig über ALSA angefordert, in einem Buffer zwischengespeichert und auf den realen Vektorteil übertragen. Es ist bereits eine feste Schrittgröße x definiert, welche spezifiziert, um wie viele Samples das Fenster (realer Anteil) fortbewegt werden muss, bevor die nächste FFT durchgeführt wird. Eine

kleinere Schrittgröße führt dazu, dass eine größere Anzahl von FFT-Berechnungen pro Zeiteinheit durchgeführt werden und damit eine bessere zeitliche Auflösung bei gleichbleibender Frequenzauflösung erreicht wird. Da die FFT bei gleicher Samplegröße n öfter angewendet wird, wird die Pipeline allerdings rechenintensiver.
Damit Diskontinuitäten in der Frequenzmessung minimiert werden können, muss eine Fensterfunktion, vor der Anwendung der FFT, auf die gesammelten Audiodaten angewendet werden. Die Fensterfunktionen wurden gemäß [34] entnommen. Damit das eigentliche Fenster auch nach der Ausführung der Fourier-Transformation die ursprünglichen Audiorohdaten enthält, wird zuvor eine Kopie des Fensters angelegt. Auf die Kopie wird zunächst eine Fensterfunktion angewendet, damit spektrale Streueffekte in der Frequenzdomäne gemindert werden. Dabei muss das Fensterverfahren lediglich auf den Realteil des Signals angewendet werden, da der Imaginärteil an allen Stellen den Wert 0 trägt. Anschließend werden sowohl der reale Vektor, wie auch der imaginäre Vektor in die FFT gegeben und dort mit den resultierenden Werten für Real- und Imaginärteil befüllt.
Über die Lautstärkenauswertung kann aus imaginärem und realen Vektor ein neuer Vektor gebildet werden, welcher die eigentlichen Werte der Frequency Bins enthält. Dieser Vektor enthält den Lautstärkeanteil der Einzelfrequenzen am Gesamtsignal. Dabei werden Real- und Imaginärteil über Formel 2.22 in einen einzigen Vektor umgewandelt.
Abschließend wird als Sicherheitsmaßnahme ein Frequenzbandpass angewendet, welcher jegliche für die Musik irrelevanten Frequenzen herausfiltert. Dabei werden in der Anwendung entsprechend 2.6 lediglich Frequenzen im Bereich $[100; 10000]$Hz zugelassen. Jede Bin im Amplitudenvektor, welche eine Frequenz beschreibt, die über oder unter diesem Bereich liegt, wird auf den Wert 0 gesetzt. Dadurch besitzt die jeweilige Einzelschwingung keinen Anteil am Gesamtsignal mehr und verschwindet damit aus der Betrachtung.
Nach der Audiovorverarbeitung liegt damit ein Vektor vor, welcher die Lautstärkeanteile der n Einzelfrequenzen enthält, mit derer das Gesamtsignal approximiert werden soll. Dieser Vektor wird im Folgenden weiterverarbeitet und transkribiert.

4.2.4 Transkription

An dieser Stelle wurden im Zuge der Bachelorarbeit eigene Algorithmen entworfen, um die Transkription ausgehend vom Koeffizientenvektor c zu realisieren. Dabei werden die transkribierten Notenwerte in LilyPond-Notation notiert. Im Folgenden wird beschrieben, wie anhand der Frequency Bins Töne, Pausen, Akkorde und Tonlängen erkannt werden können und diese zur Melodieerkennung verwendet werden können.

Tonerkennung

Bei der Tonerkennung ist es von Bedeutung den Notenwert einer für ein Signal bestimmten Frequenz zu identifizieren. Für diesen Zweck wurde der folgende Algorithmus entworfen.

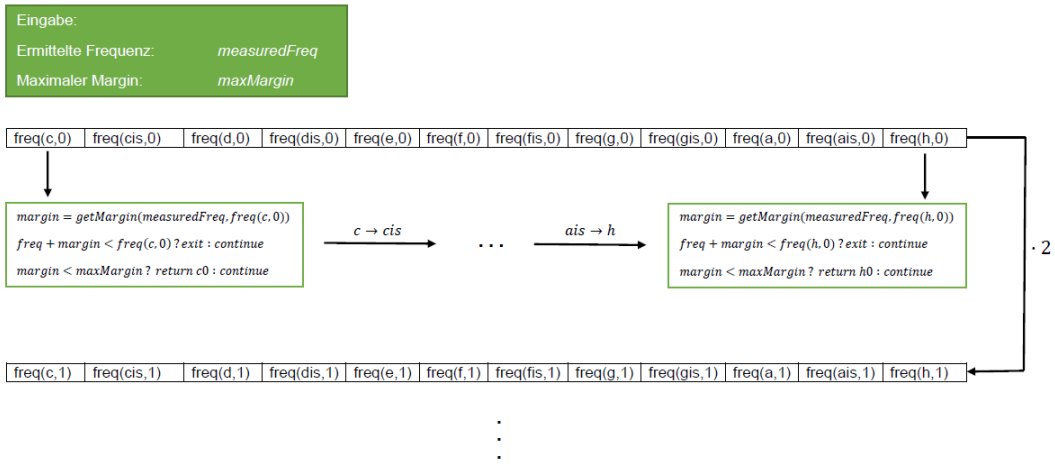


Abbildung 4.8: Transkription

Es wird angenommen, dass der zu erkennende Ton die dominante Frequenz im aufgenommenen Signal darstellt. Damit muss ihr Lautstärkenanteil maximal sein. Diese Erkenntnis wird ausgenutzt und die Frequency Bin im Lautstärkenvektor ermittelt, welche den maximalen Wert aufweist. Dies entspricht dem Lautstärkenanteil der jeweiligen Einzelschwingung. Um die maximale Frequency Bin zu bestimmen, genügt eine Maximumssuche über den Koeffizientenvektor c. Die ermittelte Frequency Bin kann über deren Index mithilfe der Formel 2.24 in eine Frequenz umgerechnet werden.
Damit wurde die dominante Frequenz f_k im Signal ermittelt. Diese Frequenz muss nun in einen Notenwert umgerechnet werden. Ausgehend von der Tabelle 2.1 wird ein Vektor erstellt, welcher die 12 Frequenzen der Grundtöne in Oktave 0 besitzt. Der Vektor repräsentiert damit eine Oktave. Zur Verringerung des Rechenaufwands könnte von einer höheren Oktave gestartet werden. Der Frequenzvektor wird nun iterativ durchlaufen und mit der ermittelten Frequenz $measuredFreq$ verglichen. Da $measuredFreq$ ausgehend von einer endlichen Anzahl an Messwerten errechnet wurde, wird $measuredFreq$ nie der exakten Frequenz des eigentlichen Tones entsprechen. Dazu wird ein Sicherheitsabstand *maxMargin* in der Einheit cent gewählt und über Formel 2.2 überprüft.
Es wird iterativ der Frequenzvektor *freq* durchlaufen. Es wird die Frequenzabweichung in cent zwischen der aktuell betrachteten Frequenz *freq(halfStep,octave)* und der gemessenen Frequenz *measuredFreq* berechnet. Anschließend wird überprüft, ob die *measuredFreq* zusammen mit dem Margin kleiner ist als die aktuell betrachtete Frequenz *freq(halfStep,octave)*. Falls ja, dann ist *measuredFreq* kleiner als alle kom-

menden Frequenzen in *freq* und die Berechnung kann abgebrochen werden. Damit entspricht die gemessenen Frequenz keinem Ton. Andererseits kann mit der Abarbeitung des Algorithmus fortgefahren werden.
Danach soll ermittelt werden, ob *measuredFreq* dem *freq(halfStep,octave)* zugeordneten Ton entspricht. Falls der berechnete Margin *margin* kleiner ist als *maxMargin*, dann ist diese Bedingung erfüllt und die Bearbeitung kann abgebrochen werden. Andererseits wurde der Ton noch nicht gefunden und es wird zum nächsten Halbton in der Oktave fortgeschritten. Für diesen werden erneut die gleichen Bedingungen getestet.
Wurde eine Oktave durchlaufen, dann wird die Tabelle *frequencyTable* herangezogen, um die nächste Oktave zu ermitteln. In der Praxis wird ein simplerer Mechanismus verwendet, um die nächst höhere Oktave zu ermitteln. Es wird die Kenntnis ausgenutzt, dass Notenwerte, die eine Oktave höher liegen eine exakt um den Faktor 2 größere Frequenz aufweisen. Die Werte im Vektor *freq* müssen also lediglich verdoppelt werden, um die Frequenzwerte der entsprechenden Notenwerte eine Oktave darüber zu bestimmen.

Pausenerkennung

Grundlegend könnten die Lautstärkenwerte im Koeffizienzvektor c herangezogen werden und eine Lautstärkenuntergrenze definiert werden. Falls alle Werte in jeder Bin unter diesem Wert liegt, dann erfüllt keine Frequency Bin die Lautstärkenvoraussetzung, um als dominante Frequenz gewertet zu werden. Damit würde eine Pause ermittelt werden. Im entwickelten System wird bislang keine Pausenerkennung umgesetzt.

Akkorderkennung

Die Akkorderkennung baut auf der Tonerkennung auf. Es muss allerdings angegeben werden, wie viele Töne y aus dem Gesamtsignal ermittelt werden sollen. Es müssen diejenigen Frequenzen ermittelt werden, welche die y höchsten Lautstärkenanteile am Gesamtsignal besitzen. Es werden dazu diejenigen Frequency Bins ermittelt, welche die y höchsten Werte aufweisen. Die ermittelten Frequency Bins werden anschließend über Formel 2.24 in Frequenzen umgerechnet. Diese Frequenzen müssen danach wie bereits besprochen in die entsprechenden Notenwerte umgewandelt werden.
Das System ermöglicht lediglich eine Akkorderkennung für einfrequentige Schwingungen. Für Klänge mit Obertönen kann das System keine Garantie bieten, welche der Frequency Bins als Ton gewertet wird. Daher wird das System die Akkorde nicht für Instrumente bestimmen können. Dazu ist die Komplexität zu hoch, als dass dies mithilfe der in diesem System entwickelten Algorithmen ermöglicht werden würde.

Tonlängenerkennung

Zur Tonlängenerkennung muss die Zeitdauer eines entsprechenden Tones ermittelt werden. Dazu kann die Dauer $\frac{x}{Fs}$ zwischen zwei FFT-Berechnungen herangezogen werden und diese mit der Anzahl der FFT-Berechnungen multipliziert werden, welche denselben Ton identifzieren. Dieses Verfahren bezieht allerdings nicht die FFT-Berechnungszeit und andere verzögerungsbehaftete Berechnungen mitein, welche in der Praxis auftreten. Als praktische Lösung kann ein Timer verwendet werden.

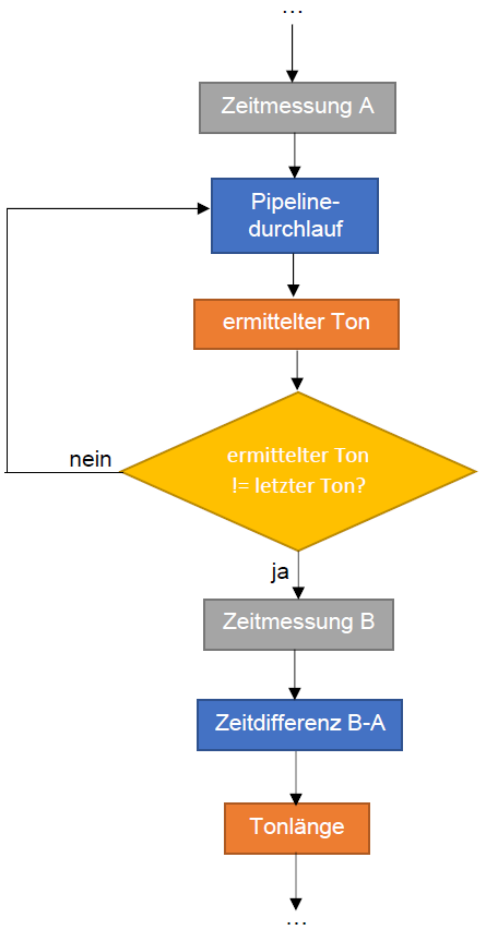


Abbildung 4.9: Tonlängendetektion

Wird ein Ton das erste Mal erkannt, so wird ein Timer gestartet. Nach jeder FFT wird anschließend erneut der Ton ausgewertet. Falls der neu ermittelte Ton mit dem Ton übereinstimmt, für welche der Timer gestartet wurde, dann wird weiterhin vom selben Ton ausgegangen. Falls der neu ermittelte Ton sich vom Ton des Timers unterscheidet, dann handelt es sich um einen neuen Ton. Der entsprechende Timer wird gestoppt und die Zeitdauer ermittelt. Aus der Zeitdauer und der erkann-

ten Tonhöhe wird anschließend ein Notenwert ermittelt, welcher neben der Tonhöhe auch die Tonlänge in sich trägt. In der Praxis werden zur Simulation eines Timers zwei Zeitwerte verwendet. Ein Zeitwert wird zu Beginn gesetzt. Wurde ein neuer Ton erkannt, so wird der zweite Zeitwert gesetzt. Aus der Differenz der beiden Zeitwerte kann die Zeitdauer des Tones ermittelt werden und diese in die entsprechende Notation umgewandelt werden.

Melodiekonstruktion

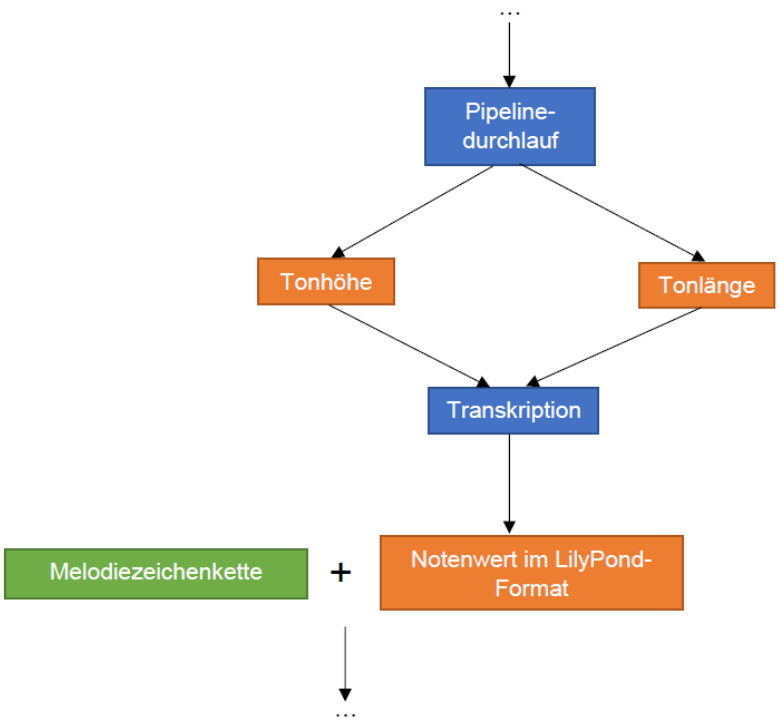


Abbildung 4.10: Melodiekonstruktion

Die erkannte Melodie wird zunächst in einer Zeichenkette festgehalten. Diese ist bereits in Lilypond-Notation formatiert und wird im Laufe der Pipeline um neue Notenwerte erweitert. Sobald ein neuer Notenwert erkannt wurde, wird aus seiner Dauer und Tonhöhe die entsprechende LilyPond-Notation generiert. Die formatierte Zeichenkette wird anschließend an die bereits bestehende Melodiezeichenkette angehängt. Nach Ablauf der Aufnahmezeit ist in der Melodiezeichenkette die gesamte erkannte Melodie als LilyPond-Format enthalten. Diese kann anschließend von LilyPond zur Erzeugung eines Notenblattes verwendet werden.

Taktvorgabe

Damit ein Musiker die Melodie an ein Tempo anpassen kann, wird eine Taktvorgabe benötigt. Diese erfüllt die Rolle eines Metronoms oder des Dirigenten. Im entwickelten System wurde daher eine Taktvorgabe implementiert.

Die Taktvorgabe gibt asynchron den Takt vor und kann über einen seperaten Thread implementiert werden. Der Thread wartet entsprechend der Tempoangabe eine konstante Zeit und gibt möglicherweise auditives, aber zumindest visuelles Feedback, dass ein neuer Taktschlag erfolgt ist. Über dieses Feedback kann sich der Benutzer des Systems orientieren, um zu wissen, wann der nächste Ton in der Melodie gespielt werden soll.
Das Tempo *temp* eines Musikstückes ist in der Einheit Schläge pro Minute *bpm* gegeben. Zur Taktvorgabe wird ein Metronom simuliert, wobei das Zeitintervall zwischen zwei Schlägen über folgende Formel berechnet werden kann:

$$\Delta t = \frac{60 \cdot 1000}{temp} ms \tag{4.1}$$

Diese Formel liefert die Zeitdauer für einen Schlag des Metronoms/Taktgebers.
Visuelles Feedback kann über die Taktzahl gegeben werden, wobei auditives Feedback die sinnvollere Variante ist. Im entwickelten System wird dazu eine Audiodatei verwendet, welche über das Dateisystem angesteuert werden kann. Die Dauer der Audiodatei muss dabei in das Metronomverhalten miteinbezogen werden. Dabei kann die Interaktion mit der Datei eine bestimmte Latenz besitzen, die in das Feedbackintervall miteingerechnet werden muss. Dennoch kann diese Interaktion zu einem Bottleneck für die Echtzeitfähigkeit des Systems führen. Eine weitere Alternative wäre innerhalb des Codes direkt über die Lautsprecher des technischen Systems eine Audioausgabe zu erzeugen.
Eine weitere Sache, die beachtet werden muss, ist die Tatsache, dass hörbares Feedback des Systems die Qualität der Transkription vermindern kann, da die Taktschläge das Audiosignal verunreinigen. Daher sollte die Taktoption lediglich mit Kopfhörern durchgeführt werden.

4.2.5 Notenblatt-Generierung

Die Erzeugung eines Notenblattes ausgehend von einer Zeichenkette, welche die erkannte Melodie enthält, ist über den Programmablaufplan 4.11 dargestellt.

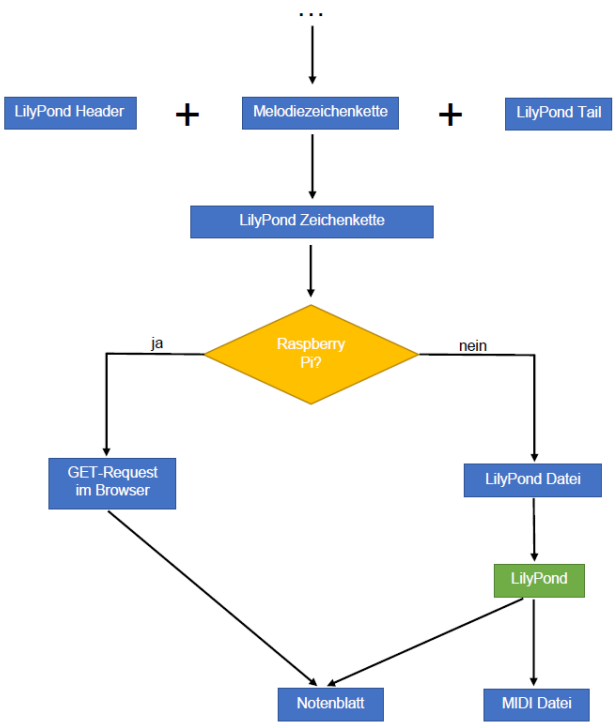


Abbildung 4.11: Notenblatterzeugung

Die Melodiezeichenkette befindet sich bereits in LilyPond-Notation. Zur Verarbeitung mit dem LilyPond-Framework müssen zunächst noch zusätzliche Daten (*LilyPond Header,LilyPond Tail*) hinzugefügt werden. Durch die Zusatzinformationen können Informationen auf dem finalen Notenblatt spezifiziert werden, sowie angegeben werden, ob neben einer PDF-Ausgabe auch die Erstellung einer MIDI-Datei erwünscht ist. Die daraus resultierende *LilyPond Zeichenkette* kann in einer Textdatei mit der Endung *.ly* gespeichert werden.
Das Betriebssystem ermöglicht dem Programm über eine Schnittstelle mit einer ausführbaren Datei des LilyPond-Frameworks zu interagieren. Das Programm kann dadurch über den Kommandozeilenbefehl *lilypond fileName* eine Lilypond Executable starten, die aus der Lilyponddatei ein Notenblatt in Form einer PDF generiert. Außerdem ist es möglich durch Angabe in der Lilyponddatei eine MIDI Datei zu erzeugen.[47]
Lilypond kann für alle gängigen Betriebssysteme (Windows,Mac,Ubuntu,...) installiert werden [66]. Da die Raspberry Pi Architektur eine ARM-Architektur darstellt, ist es also nicht möglich Lilypond auf dem Raspberry Pi auszuführen. Damit ist eine Notengenerierung auf dem Raspberry Pi nicht ohne Weiteres möglich. Dieses Defizit kann nun über zwei Möglichkeiten gelöst werden.
Zum einen kann die Zeichenkette in einer Textdatei mit der Endung *.ly* gespeichert werden und auf einen Lilypondfähigen Rechner übertragen werden (ssh, USB-Stick). Dieser kann anschließend über den Lilypond-Befehl die Datei in eine PDF und/oder MIDI-Datei umwandeln.

Weiterhin besteht die Möglichkeit die Melodie direkt auf dem Raspberry Pi über den Browser zu visualisieren. Dazu wurde ein Lilypond-Tool von Joshua Nettek entwickelt [67], welches im Web verfügbar ist. Dieses besteht aus einem Lilypond-Editor als Frontend und einer LilyPond-Render Maschine als Backend. Die Web-Anwendung ermöglicht es zudem eine Melodiezeichenkette über die URL automatisiert an das Backend weiterzugeben. Dieses rendert anschließend das Notenblatt und präsentiert es im Browser. Die zu visualisierende Melodie muss dabei in der URL als Zeichenkette url-kodiert weitergegeben werden: $https://www.hacklily.com.org/\#src=<melodyString>$. Für diese Funktionalität wurde ein zusätzliches Python-Skript geschrieben, welches die erkannte Melodie-Zeichenkette als Kommandozeilenargument erhält, diese URL-kodiert und anschließend einen Webbrowser mit der URL öffnet. Zur manuellen Weiterverarbeitung der erkannten Melodie kann entweder die Lilypond Datei in einem Lilypond basierten Musikeditor wie *Frescobaldi*[68] geladen werden. Für nicht lilypondkompatible Musikeditoren wie *MuseScore*[1] besteht diese Möglichkeit nicht. Allerdings bieten viele erhältliche Editoren eine Möglichkeit MIDI Dateien zu importieren. Damit ist also auch für nicht LilyPond-kompatible Editoren eine manuelle Weiterverarbeitung der Melodie möglich.

4.3 Betriebsmodi

Das Ziel der Bachelorarbeit ist es mitunter ein echtzeitfähiges System zur Transkription von Musik zu entwickeln. Damit die Qualität der Melodieerkennung gewährleistet werden kann, muss eine effiziente Implementierung der Pipeline 4.1 gefunden werden. Neben den theoretischen Werten wie Samplerate Fs, Samplegröße n etc., kann die Pipeline auch unterschiedlich auf dem tatsächlichen technischen System implementiert werden und damit versucht werden die Effizienz der Pipeline zu verbessern. In Folge dieser Überlegungen wurden drei verschiedene Versionen der Pipeline implementiert. Jede dieser Versionen besitzt unterschiedliche Eigenschaften und weist Vor- und Nachteile auf. Auf die Qualität der Implementierungen wird noch genauer in Kapitel 5 eingegangen und soll im Folgenden vorgestellt werden.

4.3.1 Sequentielle Implementierung

Diese Variante der Implementierung entspricht dem bereits vorgestellten Algorithmus 4.1. In jedem Schritt des STFT-Verfahrens werden als erstes Audiodaten angefordert und diese anschließend transkribiert. Die Verarbeitung läuft dabei vollkommen sequentiell ab, sodass der Zeitpunkt der nächsten Anforderung von Audiodaten von der Geschwindigkeit der Berechnung der Transkription abhängt.

Vorteile und Nachteile

Der Vorteil an dieser Variante ist, dass sie relativ einfach zu implementieren ist. Allerdings kann Qualität der Transkription verloren gehen, falls die Pipeline zu rechenintensiv für das ausführende technische System ist. Je länger das ausführende System für die FFT-Berechnungen benötigt, desto weniger Samples werden gesammelt und desto bruchstückhafter wird die Transkription. Da der Rechenaufwand für die FFT nicht linear wächst, leiden vor allem rechenschwache Systeme unter dieser Variante der Pipeline. Mehr zu diesem Thema findet sich im Kapitel 5.

4.3.2 Sequentielle Implementierung mit Audiodatenspeicherung

Mit dieser Version der Pipeline wird ein ähnlicher Ansatz verfolgt, den andere Transkriptionsapplikationen verwenden [57][62]. Die gesamte Melodie wird zunächst in einer Datei abgespeichert. Anschließend wird die Transkription anhand der gespeicherten Audiodaten durchgeführt. Im Folgenden ist der grundlegende Algorithmus für diese Variante niedergeschrieben.

Algorithmus

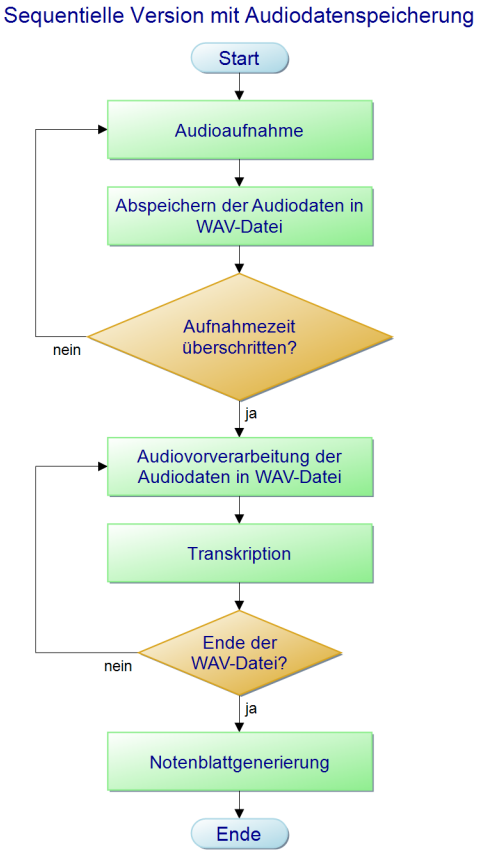


Abbildung 4.12: Sequentielle Version mit Speicherung der Audiodaten in einer WAV-Datei.

Im Diagramm 4.16 ist der Ablauf des Algorithmus dargestellt. Es werden in regelmäßigen Abständen Daten über ALSA angefordert und diese in einer WAV-Datei abgespeichert. Dies wird so lange fortgesetzt bis die Aufnahmezeit überschritten wird.
Anschließend folgt die eigentliche Verarbeitung der Audiodaten. Es wird nach der Aufnahme der Melodie Daten die Verarbeitung nach dem STFT-Verfahren durchgeführt. In jedem Transkriptionsschritt werden zunächst Daten aus der WAV-Datei gelesen und diese anschließend vorverarbeitet und die bereits kennengelernten Transkriptionsalgorithmen darauf angewendet. Es gilt, dass in jedem Iterationsschritt jeweils so viele Daten gelesen, wie die Schrittgröße x definiert wurde. Da die Verarbeitung in dieser Variante der Implementierung nicht in Echtzeit stattfindet, muss die Zeit mithilfe der Schrittgröße x und der Sample-Rate Fs festgehalten werden. Mit jedem Schritt werden x Audiodaten gelesen und eine Samplegröße n verarbeitet und damit schreitet das STFT-Verfahren in jedem Schritt um $\frac{x}{Fs}$ Sekunden voran. Die Verarbeitung wird so lange fortgeführt, bis das Ende der WAV-Datei erreicht

wurde und damit die Aufnahme beendet ist. Anschließend wird ein Notenblatt aus der erstellten Melodiezeichenkette generiert.
Zur Realisierung dieser Implementierung mussten zusätzliche Funktionen implementiert werden, welche das Lesen von und Schreiben auf eine WAV-Datei ermöglichten.[65]

Vor- und Nachteile

Diese Version der Pipeline bietet ein robustes zeitunabhängiges Transkriptionsverfahren ohne Qualitätsverluste bei der Transkription. Der Grund hierfür liegt darin, dass das Schreiben auf eine WAV-Datei unabhängig vom technischen System ungefähr gleich viel Zeit in Anspruch nimmt. Damit werden unabhängig von der Rechenleistung des Systems annähernd gleich viel Audiodaten abgespeichert und damit die Melodie in der gleichen Qualität digital repräsentiert. Die eigentliche Transkription der Daten muss die Daten lediglich aus der WAV-Datei lesen und kann dies unabhängig vom technischen Systme fehlerfrei durchführen. Es gehen nach der Aufnahme keine Daten mehr verloren.
Da das System zeitunabhängig ist, kann die Berarbeitung der einzelnen Messwerte auch eine rechenintensivere Audionachverarbeitung ausführen und damit eine bessere Qualität der Transkription zu ermöglichen. Es können neuronale Netze wie in [62] eingesetzt werden.
Andererseits kann die ständige Interaktion mit der WAV-Datei und damit mit dem Dateisystem zum Flaschenhals des Designs werden. Weiterhin ist mit dieser Implementierung offensichtlich keine Echtzeitverarbeitung möglich und damit nicht als zufriedenstellende Lösung für diese Bachelorarbeit anzusehen.

4.3.3 Parallele Implementierung

Um eine qualitativ hochwertige Echtzeitverarbeitung zu ermöglichen, muss eine Möglichkeit gefunden werden, damit zum einen schnell genug Audiodaten gesammelt werden können und andererseits gleichzeitig die Verwertung der Audiodaten vonstatten geht. Dies wird über eine parallele Ausführung der Audioaufnahme und Audioverarbeitung in unabhängigen Threads ermöglicht.

Posix Threads, Mutexe, Warteschlange

In der Implementierung werden Posix Threads (pThreads) verwendet. Damit ist es möglich die parallele Abarbeitung von Prozessen zu simulieren. Ein Thread muss vom Hauptthread gestartet werden (*start*) und kann den weiteren Programmablauf pausieren, bis der Thread beendet ist (*join*).[69] Weiterhin gibt es Mutexe, die einen Synchronisationsprozess zwischen mehreren Threads bei geteiltem Ressourcenzugriff darstellen.[70]

In der parallelen Version der Pipeline wurde ein Thread für die Audioaufnahme und ein Thread für die Audioverarbeitung kreiert. Damit soll ermöglicht werden, dass die Audioaufnahme und die Audioverarbeitung gleichzeitig stattfinden. Das grundlegende Modell der parallelen Verarbeitung ist in folgender Abbildung dargestellt:

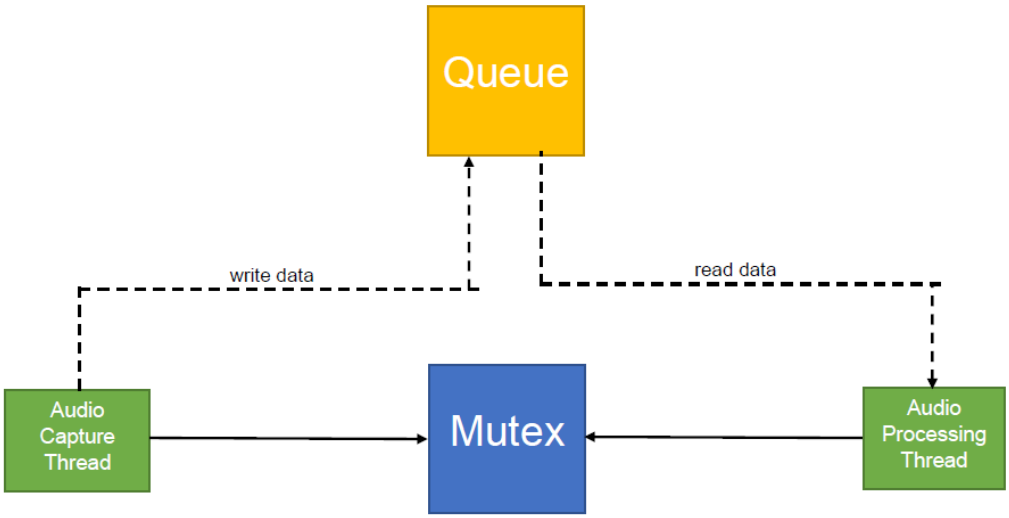


Abbildung 4.13: Thread Synchronisation bei geteiltem Datenzugriff auf Warteschlange.

Der Audioaufnahme-Thread muss die aufgenommenen Audiodaten an einem bestimmten Ort speichern und der Audioverarbeitungs-Thread liest die gespeicherten Daten von diesem Ort. Um den Umweg über eine Datei zu vermeiden, wird als interne Datenstruktur eine Warteschlange (Queue) zur Zwischenspeicherung der Audiodaten angelegt. Die beiden Threads greifen unabhängig voneinander auf die Warteschlange zu. Es muss also dafür gesorgt werden, dass die Daten in der Warteschlange konsistent bleiben. Dies wird ermöglicht, in dem die beiden Threads jeweils dem Mutex signalisieren, dass sie eine Operation durchführen möchten, welche von den anderen Threads nicht unterbrochen werden soll.

Schreibzugriff

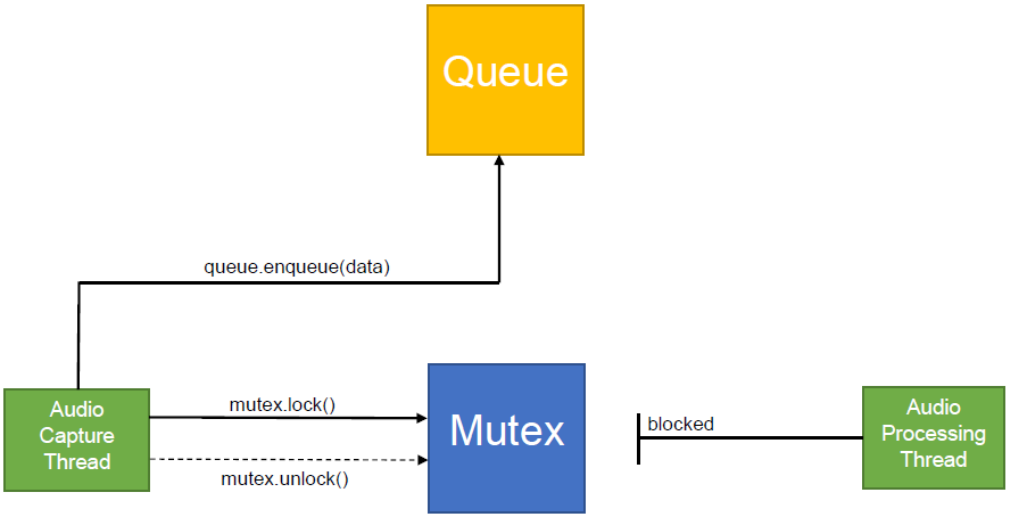


Abbildung 4.14: Schreibzugriff des Audioaufnahme-Threads auf die Warteschlange.

In Abbildung 4.14 ist der Schreibzugriff des Audioaufnahme-Threads auf die Warteschlange zu sehen. Der Thread signalisiert dem Mutex, dass dieser den Zugriff aller anderen Threads auf die geteilte Datenstruktur blockieren soll (*mutex.lock()*). Anschließend kann der Thread Daten der Warteschlange hinzufügen (*queue.enqueue(data)*). Sobald die Daten auf die Warteschlange geschrieben wurden, muss sofort die Datenstruktur für die anderen Threads freigegeben werden. Dies wird erreicht, indem der Thread dem Mutex signalisiert, dass die anderen Threads wieder befreit werden sollen (*mutex.unlock()*).

Lesezugriff

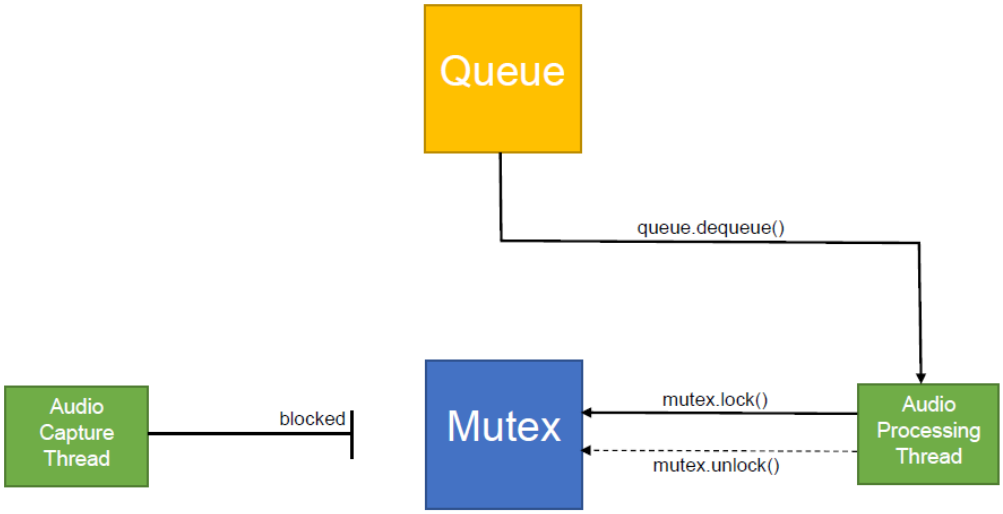
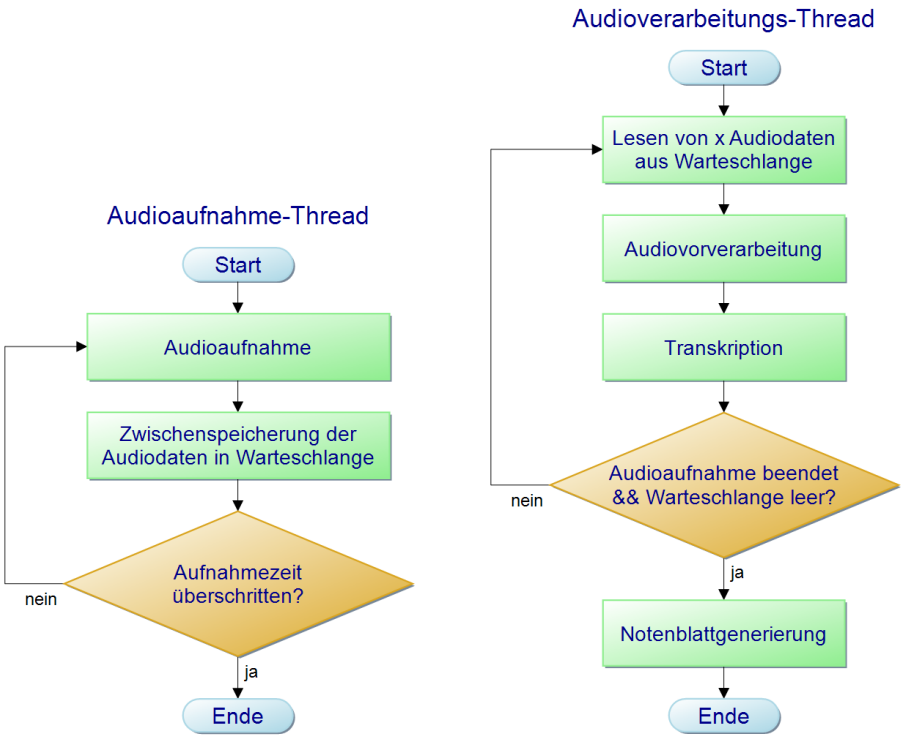


Abbildung 4.15: Lesezugriff des Audioverarbeitungs-Threads auf die Warteschlange.

In Abbildung 4.15 ist der Lesezugriff des Audioverarbeitungs-Threads auf die Warteschlange dargestellt. Bevor der Thread auf die Warteschlange zugreifen darf, muss er dem Mutex signalisieren, dass dieser die Ausführung aller weiteren Threads blockieren soll (*mutex.lock()*). Nachdem ein Element aus der Warteschlange gelesen wurde (*queue.dequeue()*), muss der Thread den Zugang zur Warteschlange sofort wieder freigeben (*mutex.unlock()*), um die restlichen Threads nicht in deren Ausführung zu behindern.

Algorithmus

In Abbildung 4.16 ist der Programmablauf sowohl für den Audioaufnahme-Thread, wie auch für den Audioverarbeitungs-Thread dargestellt.



(a) Audioaufnahme-Thread **(b)** Audioverarbeitungs-Thread

Abbildung 4.16: Parallele Ausführung der Audioaufnahme (a) und Audioverarbeitung (b) in zwei seperaten Threads.

Es werden zwei Threads, ein Mutex und eine Warteschlange als gemeinsame Datenstruktur angelegt. Ein Thread ist dafür zuständig Audiodaten über die ALSA-Schnittstelle anzufragen und anschließend in der Warteschlange abzuspeichern. Dieser Thread nimmt so lange Daten auf und schreibt sie in die Warteschlange, bis die festgelegte Aufnahmezeit überschritten wird. (Abb. 4.16a)
Ein weiterer Thread ist für die eigentliche Audioverarbeitung und Transkription der in der Warteschlange zwischengespeicherten Audiodaten zuständig. Dieser Thread liest so lange Daten aus der Warteschlange, bis diese leer ist und der Audioaufnahme-Thread keine Daten mehr aufnimmt. Abschließend kann im Audioverarbeitungs-Thread ein Notenblatt aus der erkannten Melodie generiert werden.(Abb. 4.16b)

Vorteile und Nachteile

Da beide Threads parallel arbeiten, sind sie prinzipiell unabhängig voneinander. Dadurch muss die Aufnahme der Audiodaten (Audioaufnahme-Thread) nicht in jedem Durchlauf auf langwierige Berechnungen der Audioverarbeitung und Trankskription warten (Audioverarbeitungs-Thread). Die Daten können im Audioaufnahme-Thread ohne Verzögerungen aufgenommen und in der Warteschlange abgelegt werden. Dabei

entfällt die Zeit für die Speicherung im Dateisystem. Es wird also ermöglicht, dass Audiodaten ohne größere Verluste aufgenommen werden können. Dies wird die Qualität der Transkription verbessern, da kaum Daten verloren gehen. Weiterhin ist die Berechnungszeit der Audioverarbeitung und Transkription irrelevant für die Qualität der Transkription. Der Audioverarbeitungs-Thread kann im Prinzip beliebig lange für die Ausführung benötigen, ohne dass Qualitätseinbußen in der erkannten Melodie zu verzeichnen sind.
Durch diesen Ansatz ist es also möglich die maximale Qualität für die Transkription zu erreichen und gleichzeitig eine performante Möglichkeit bietet ein echtzeitfähiges System auf einem eingebetteten System zu implementieren. Auch wenn die Qualität maximal ist, so kann die Transkription je nach technischen System dennoch beliebig lange für die Transkription benötigen. Durch die parallele Verarbeitung wird damit ein Kompromiss zwischen Echtzeitverarbeitung und Qualität der Transkription ermöglicht und stellt deshalb die beste Option für die Implementierung der Pipeline dar.

4.4 Benchmarking

In Bezug auf das Benchmarking müssen vor allem zwei Dinge getestet werden. Dies ist einerseits die Qualität des Ergebnisses der Transkription in Bezug auf die Korrektheit. Da das System allerdings auch echtzeitfähig sein soll, muss die Performanz der Pipeline in Bezug auf die drei Betriebsmodi betrachtet werden, aber auch ein Vergleich zwischen Standard- und Minimalsystem durchgeführt werden. Es wurden jeweils automatisierte Tests entwickelt, die mehrere Größen verändern und versuchen den Definitionsbereich der zu testenden Größe möglichst breit abzudecken.

4.4.1 Betrachtung der Qualität

Die Qualität der Transkription kann einerseits in Bezug auf das eigentliche Erkennen der richtigen Töne und Akkorde betrachtet werden. Andererseits kann die Qualität der Melodieerkennung untersucht werden, die neben der Tonhöhe auch die Tonlängenerkennung zur Detektion von Rhythmen durchführt. Weiterhin kann das Benchmarking neben reinen Frequenzen für Töne auch gespielte Töne von Instrumenten beinhalten. Damit kann die Robustheit des Systems betrachtet werden. Im Folgenden wird die Implementierung des Qualitätsbenchmarking, sowie Qualitätsmaße angesprochen. Dabei werden die ermittelten Ergebnisse eines jeden Benchmarkingdurchlaufs in einer CSV-Datei gespeichert, sodass diese von Datenverarbeitungsprogrammen wie *Excel* visualisiert werden können.

Tonerkennung

Für das Ton-Benchmarking werden iterativ die Frequenzen der einzelnen Töne oktavenweise ermittelt. Anschließend wird über ein Python-Skript die Frequenz als Kommandozeilenargument übergeben und von diesem für eine bestimmte Zeitspanne in einem seperaten Thread gespielt. Parallel dazu wird die normale Pipeline ausgeführt. Der Ablauf des Benchmarkings ist in Abbildung 4.17 verdeutlicht.
Die Pipeline liefert in jedem Durchlauf eine gemessene Frequenz. Bei Vergleich mit der theoretisch erwarteten Frequenz, welche aktuell vom Python-Skript gespielt wird, kann eine Frequenzabweichung in der Cent-Skala errechnet werden (Formel 2.2). Sobald die Abweichung kleiner als ein festgelegter Cent-Margin ist, so wurde der richtige Ton erkannt und es kann zum nächsten Ton fortgeschritten werden. Dazu wird erneut dasselbe Schema durchlaufen. Falls nicht die richtige Frequenz erkannt wird, dann beträgt die Abweichung einen größeren Cent-Betrag als der maximale Margin und das Python-Skript beendet die Pipeline nach Ablauf der Spielzeit. Anschließend wird auch an dieser Stelle zum nächsten Ton fortgeschritten.
Um die Tonerkennung zu verbessern, muss eine bessere Frequenzauflösung geschaffen werden und damit die Samplegröße n erhöht werden. Weiterhin muss eine bestimmte Obergrenze für die Tonerkennung herrschen, da ansonsten die Frequenzen zum Teil gefährlich für das menschliche Ohr werden können, aufgrund der in den Schwingungen steckenden Intensitäten [71][72]. Für die Tonerkennung wurden alle Töne bis einschließlich Oktave 6 betrachtet.

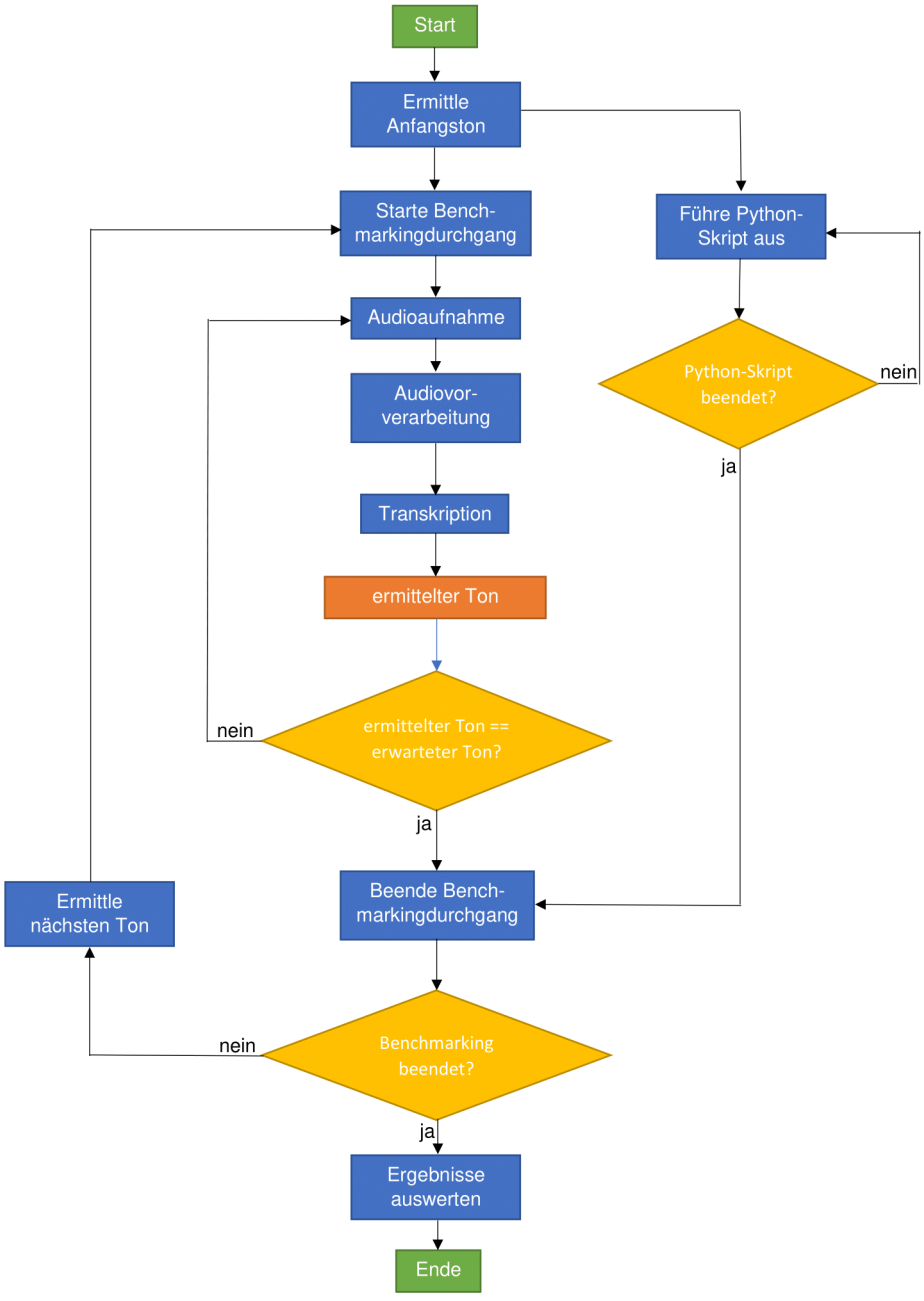


Abbildung 4.17: Ablaufdiagramm des Ton-Benchmarkings.

Akkorderkennung

Für die Akkorderkennung wird das Konzept der Tonerkennung in Abbildung 4.17 erweitert. Anstelle einzelner Töne werden Akkorde erzeugt und miteinander verglichen. Es wird dabei nicht iterativ vorgegangen, sondern z Frequenzen zu Tönen entsprechend zufällig berechnet. Dabei beschreibt z die Anzahl der Töne pro Akkord. Diese Anzahl wird beim Benchmarking ausgehend von 2 Tönen pro Akkord inkrementell erhöht. Beim Akkord-Benchmarking wurde die Akkorderkennung auf maximal 5 Töne pro Akkord getestet.
Die ermittelten Frequenzen werden auch an dieser Stelle an ein Python-Skript weitergegeben, welches diese gleichzeitig für eine bestimmte Zeitdauer als Sinusschwingungen akustisch ausgibt.

Melodieerkennung

Für die Melodieerkennung gestaltet sich das Benchmarking weit komplizierter. Neben der Tonhöhe, muss auch die Tonlänge betrachtet werden. Für die Melodieerkennung wurde ein automatisierter Test entwickelt. Dabei werden einerseits einfache zufällig generierte Melodiezeichenketten erzeugt. Weiterhin können die Melodien auch aus einer Textdatei stammen. Das Benchmarking wird für jede der drei entwickelten Pipeline-Varianten durchgeführt und bewertet.

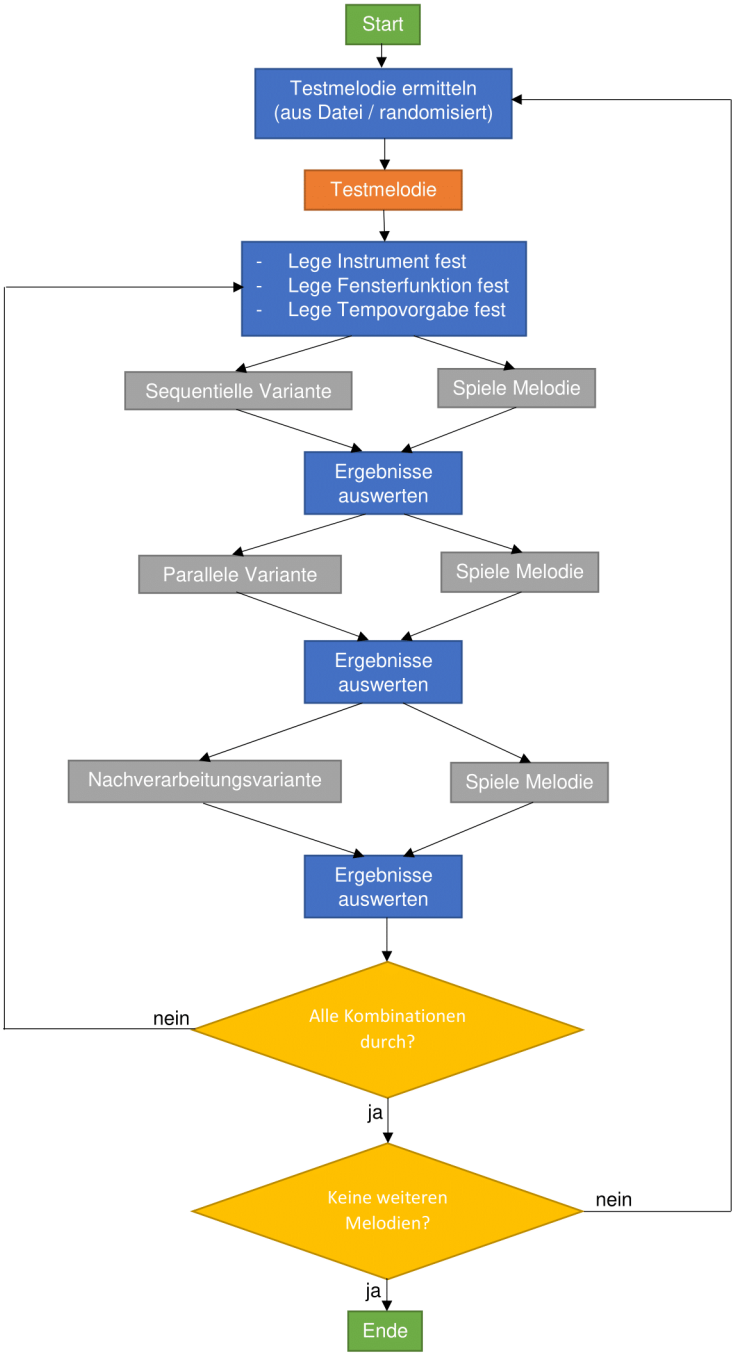


Abbildung 4.18: Ablaufdiagramm des Melodie-Benchmarkings.

Die Testmelodien müssen dabei im LilyPond-Format vorliegen. Dabei beschreibt jede Zeile der Datei eine eigenständige Melodie. Das Programm liest die Melodie zeilenweise aus und generiert daraus eine MIDI-Datei. Beim zufälligen Benchmarking wird randomisiert eine einfache Melodie im LilyPond-Format generiert und daraus eine MIDI-Datei generiert. Die Töne der erzeugten Melodien liegen dabei lediglich in den Oktaven 3,4 und 5. Dieses Tonintervall ist sinnvoll, da der Frequenzbereich von vielen Instrumenten dem entspricht.[73]

Die abgespeicherte Datei kann in einem seperaten Thread abgespielt werden und gleichzeitig die Pipeline in einem bestimmten Betriebsmodus ausgeführt werden. Durch diese Möglichkeiten ist es einerseits möglich das System für einen großen Definitionsbereich zu testen, weiterhin besteht aber auch die Möglichkeit die Melodien manuell anzupassen. Zum Beispiel sind bei der zufälligen Methode in seiner aktuellen Version keine punktierten Notenwerte, Pausen oder Bindebögen möglich. Dafür können aber in einer Textdatei entsprechende Melodien eingetragen werden.
Bei der Melodieerkennung ist auch das Zeitverhalten von Interesse. Daher können unterschiedliche Tempi unterschiedliche Auswirkungen auf die Transkription haben. Daher wird in einer Schleife für jede Melodie unterschiedliche Tempovorgaben getestet. Die Ergebnisse des Benchmarkings werden dadurch aussagekräftiger.

Im Gegensatz zum Ton- und Akkordbenchmarking, bei denen lediglich die richtigen Töne identifiziert werden und das zeitliche Verhalten irrelevant ist, ist beim Melodiebenchmarking mit zeitlich verändernden Signalen zu rechnen. Die Qualität der Transkription hängt also auch davon ab, wie gut spektrale Streueffekte gemindert werden können. Die Qualität der Transkription muss damit neben dem Tempo auch in Bezug auf die verwendete Fensterfunktion betrachtet werden. Daher wird in einer weiteren Schleife für jede Melodie verschiedene Fensterfunktionen betrachtet.
Anschließend muss die erkannte Melodie mit der Testmelodie verglichen werden. Dabei wird tonweise durch die Testmelodie geschritten. Jeder Ton besitzt eine bestimmte Tonlänge. Es wird anschließend so lange die erkannte Melodie tonweise durchschritten, bis die Tonlänge der Testmelodie überschritten wird. Anschließend kann der nächste Ton in der Testmelodiezeichenkette verwendet werden. Damit ist sichergestellt, dass das Benchmarking weniger anfällig für Transkriptionsfehler ist. Die Testmelodie und die erkannte Melodie bleiben damit synchronisiert.

Als Qualitätsmaße dienen damit zum einen wie viele Töne zum richtigen Zeitpunkt korrekt erkannt wurden. Dazu kann ein einfacher Zähler verwendet werden. Aus den korrekt identifizierten Tönen und den insgesamt in der Melodie existierenden Tönen kann eine Erfolgsquote für die Melodieerkennung errechnet werden. Weiterhin muss die Tonhöhenerkennung und die Tonlängenerkennung überprüft werden. Dabei wird für jeden erkannten Ton die Frequenzdifferenz in cent zum aktuell betrachteten Ton in der Testmelodie berechnet (Formel 2.2) und am Ende der Ausführung wird über die Anzahl der Töne gemittelt. Weiterhin kann die Notenlängenabweichung zwischen erkanntem Ton und aktuell betrachteten Ton der Testmelodie bestimmt werden. Anschließend wird auch an dieser Stelle ein Mittelwert gebildet.
Da das System letztendlich zur Transkription von Musik unabhängig vom Instru-

ment verwendet werden soll, können die Tests weiter ausgeweitet werden. Dazu kann in der LilyPond-Datei das gewünschte MIDI-Instrument spezifiziert werden, welches die Melodie beim Benchmarking spielen soll. LilyPond bietet eine Auswahl von 128 MIDI-Instrumenten [74]. Die Namen dieser Instrumente können in einer Textdatei abgespeichert werden iterativ ausgelesen werden. Dadurch können die entwickelten Tests für verschiedene Instrumente erweitert werden. Damit wäre eine robuste Möglichkeit gegeben das System effizient für mehrere Instrumente zu testen.

Das finale System ist nicht in der Lage eine Transkription für beliebige Instrumente durchzuführen, da es dazu dem exakten Klangprofil des Instruments bedarf. Für das Benchmarking musste daher ein relativ einfrequentiges MIDI-Instrument verwendet werden. Dazu wurde das Instrument *voice oohs* gewählt.
Die Ergebnisse des Qualitätsbenchmarkings sind in Kapitel 5 festgehalten.

4.4.2 Betrachtung der Performanz

Die Pipeline kann grob in die Abschnitte *Audioaufnahme*, *Audiovorverarbeitung*, *Transkription* und *Notenblattgenerierung* aufgeteilt werden. Dabei ist letzteres unabhängig von den Betriebsmodi und nicht vergleichbar bezüglich unterschiedlichen ausführenden Systemen. Es interessieren damit lediglich die ersten drei Module. Weiterhin ist bekannt, dass die FFT mit zunehmender Fenstergröße nicht linear rechenintensiver wird. Das Benchmarking-Modell wurde so implementiert, dass von jedem der drei Teile die Ausführungszeit gemessen wurde. Zusätzlich wurden die FFT-Ausführungszeit betrachtet, sowie die allgemeine Ausführungszeit des jeweiligen Betriebsmodus.

Damit die jeweiligen Zeiten bestimmt werden konnten, wurden Timer in die eigentlichen Betriebsmodi eingefügt, an der richtigen Stelle gestartet und an der richtigen Stelle gestoppt. Die Zeitdifferenz wurde anschließend in einer CSV-Datei gespeichert und kann in Programmen wie Excel ausgewertet werden.

Zur Betrachtung der Performanz sind weiterhin wichtig unterschiedliche Schrittgrößen x und unterschiedliche Samplegrößen n zu betrachten. Dies wurde mithilfe zweier Schleifen implementiert. Die äußere verdoppelt dabei immer n und die innere verdoppelt jeweils x, bis $x == n$ gilt.

Da die Pipeline während der Aufnahmezeit mehrmals durchlaufen wird, bietet es sich an die Ausführungszeiten in jedem Durchlauf zu messen und auf zu summieren. Später kann der Mittelwert der Ausführungszeiten ermittelt werden. Dazu wird zusätzlich ein Zähler *counter* implementiert, der inkrementiert wird, sobald ein Pipelinedurchlauf beendet wurde. Die summierten Ausführungszeiten der einzelnen Pipeline-Teile können durch den Zähler *counter* geteilt werden und damit die gemittelten Ausführungszeiten der Codeabschnitte berechnet werden. Damit liefert das Ergebnis einen stabileren Wert.

Weiterhin kann über *counter* die Verlustrate bei einem bestimmten n und x berechnet werden. Bei einer bestimmten Ausführungszeit t kann die theoretische Obergrenze der Anzahl Durchläufe in der Pipeline berechnet werden. Diese entspricht der Annahme, dass alle Teile bis auf die Audioaufnahme keine Ausführungszeit benötigen. Lediglich durch die Aufnahme von x Audiodaten wird die Zeit $\frac{x}{Fs}$ Sekunden vorangeschritten bei einer Samplerate Fs. Die theoretische Obergrenze der Anzahl der Pipeline-Durchläufe ist gegeben durch:

$$maxIterations = \frac{t}{\frac{x}{Fs}} \tag{4.2}$$

Damit wird die maximale Anzahl an Pipeline-Durchläufen bei einer Ausführungszeit t, einer Schrittgröße x und einer Samplerate Fs berechnet. Der Zähler *counter* beschreibt die tatsächliche Anzahl der Pipelinedurchläufe eines bestimmten Betriebsmodus.

Bei den Varianten kann es vorkommen, dass $counter < maxIterations$ ist. Der Grund für diese Tatsache ist, dass neben der Audioaufnahme auch die Audiovorverarbeitung, sowie die Transkription eine bestimmte Berechnungszeit benötigen. Dadurch können in derselben Aufnahmezeit t nicht gleich viele Pipelinedurchläufe durchgeführt werden und damit weniger Audiodaten aufgenommen werden. Dies entspricht einem Verlust von Audiodaten und kann damit die Qualität der Transkription schmälern. Die Verlustrate r kann wie folgt berechnet werden:

$$r = \frac{maxIterations - counter}{maxIterations} = 1 - \frac{counter}{maxIterations} \tag{4.3}$$

Durch diese Vorkehrungen können Performancbetrachtungen in Bezug auf die verschiedenen Betriebsmodi bei unterschiedlichen n und x durchgeführt werden. Weiterhin bietet es eine Möglichkeit das System auf verschiedenen technischen Systemen zu beurteilen und damit Aussagen über die Skalierbarkeit des Systems getroffen werden.
Die eigentlichen Betrachtungen werden im Kapitel 5 fortgeführt.

4.5 Zusätzliche Entwicklungen

Im Laufe der Bachelorarbeit wurden verschiedene grafische Hilfselemente entwickelt. Dies wurde vor allem diesbezüglich durchgeführt, dass bestimmte Dinge getestet werden mussten, oder bestimmte Fragenstellungen geklärt werden mussten. Jede dieser Entwicklungen wurde über Python mit dem *PyQt5*-Framework [75] realisiert. Im Folgenden werden die entstandenen Hilfselemente vorgestellt.

Da ein Hauptaugenmerk auf der Audioverarbeitung der akustischen Signale lag, war es von großer Bedeutung zu überprüfen, ob Komponenten wie die FFT oder Zugriffe auf WAV Dateien korrekt funktionierten. Um die Funktionsweise dieser Komponen-

ten zu prüfen, wurde mithilfe von Python ein Audiospektrogramm aus den aufgenommenen Daten erstellt. Dazu gibt es die Möglichkeit das entsprechende Signal, welches in einer WAV Datei aufgezeichnet wurde, synchron zum Audiospektrogram abzuspielen. Damit ist es möglich ein Signal mit Auftrennung in seine Komponenten zu betrachten. Dieses Tool kann auch dazu verwendet werden, die Klangprofile eines Instruments zu bestimmen.

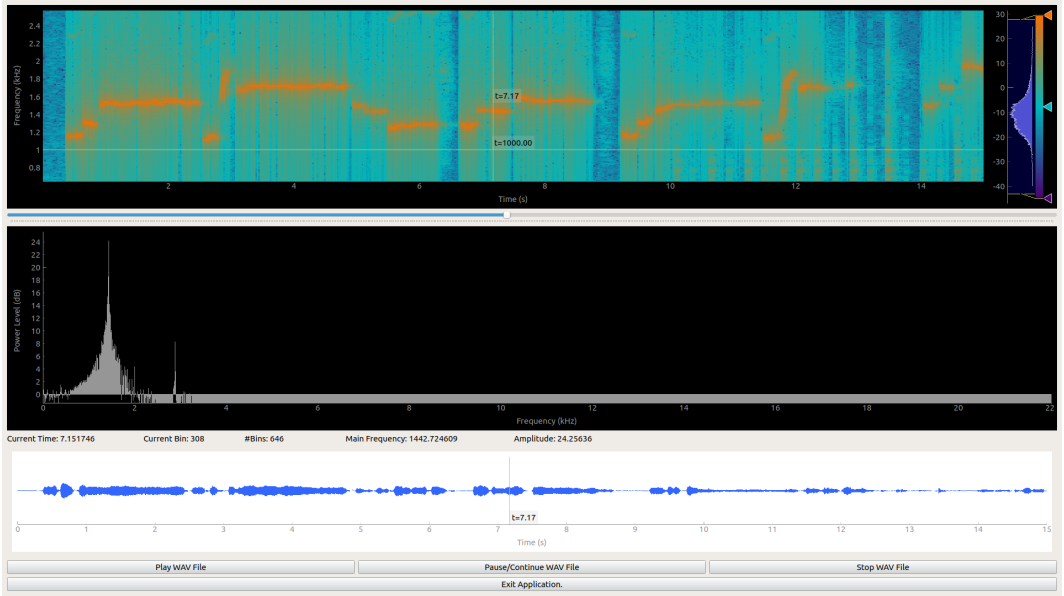


Abbildung 4.19: Audio-Spektrogramm.

Weiterhin war es wichtig zu überprüfen, ob die Ton- und Akkorderkennung auf korrekte Weise arbeiten. Dazu wurde ein Tool geschrieben, welches es ermöglicht eine bestimmte Anzahl an Frequenzen gleichzeitig zu erzeugen. Damit ist es möglich die Ton- und Akkorderkennung manuell zu überprüfen. Dieses Tool gibt es einerseits als automatisiertes Skript, dem die Frequenzen der Töne als Kommandozeilenargument übergeben werden können. Andererseits gibt es die Möglichkeit die Frequenzen manuell in einer GUI zu verändern. Das Tool kann bei der Ton- und Akkorderkennung eingesetzt werden.

Abbildung 4.20: Frequenzgenerator zur Erzeugung von Tönen und Akkorden.

Als letztes zusätzliches Werkzeug wurde eine grafische Oberfläche programmiert, um das durch die Pipeline generierte Notenblatt dem Nutzer zu präsentieren und gleichzeitig die erkannte Melodie über eine aus einer MIDI Datei erzeugten WAV Datei abzuspielen.

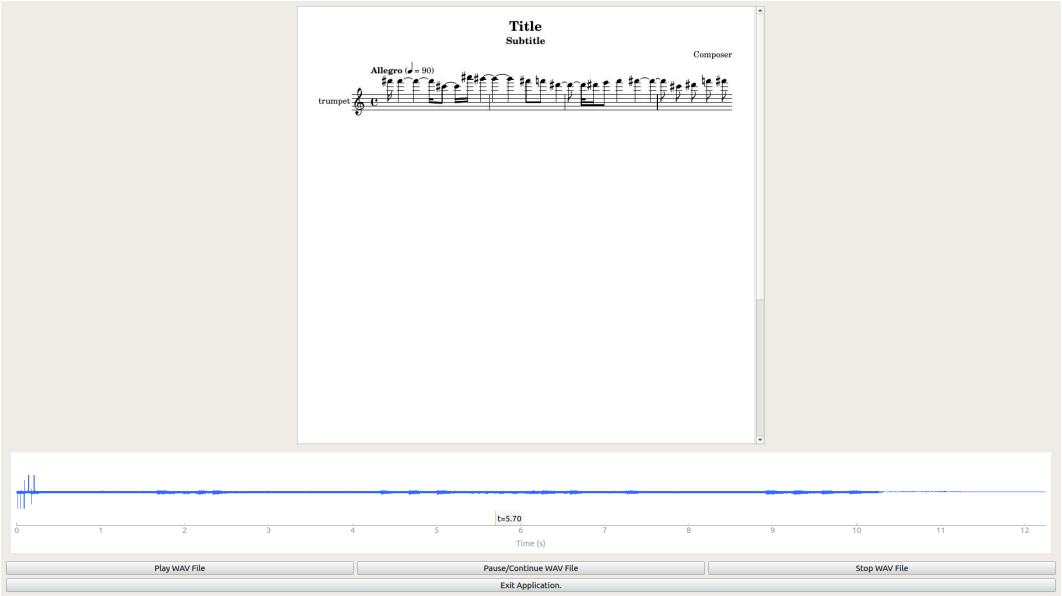


Abbildung 4.21: Notenblatt-Demonstrator.

Da ARM-Systeme keine LilyPond-Installation besitzen [66], ist das Tool 4.21 nicht für den Raspberry Pi verwendbar, da der Raspberry Pi ohne LilyPond keine Notenblatt generieren kann. Es wurde ein weiteres Python-Skript geschrieben, welches allerdings HackLily [67] eine URL über den Browser aufruft und dort die Melodie visualisiert.

All diese Entwicklungen haben letztendlich zu der Entwicklung des finalen Systems beigetragen und sind deswegen an dieser Stelle aufgeführt.

4.6 Weitere Bemerkungen

Das System ist größtenteils in C programmiert. Dabei wurden zur Berechnung der FFT auf bereits vorgefertigte Implementationen zurückgegriffen [30]. Zur Interaktion mit den Audioschnittstellen des jeweiligen Rechners wurden Codeausschnitte von [65] verwendet. Die Anbindung von grafischen Elementen wurde über Python vorgenommen, wobei das *PYQT5*-Framework [75] zur Erstellung von grafischen Elementen herangezogen wurde. Die Erzeugung des Notenblattes wird über das *LilyPond*-Framework realisiert. Weiterhin kann über *Doxygen* eine Dokumentation aus dem Code generiert werden.

5 Ergebnisse und Diskussion

5.1 Methodik

Das Thema der Bachelorarbeit lautet die *Entwicklung eines echtzeitfähigen Systems zur Transkription von Musik*. Ausgehend von diesem Titel können mehrere Eigenschaften abgeleitet werden, die das System erfüllen muss und wonach getestet werden soll. Dies ist zum einen die Tatsache, dass eine echtzeitfähige Pipeline zur Transkription von Musik entwickelt werden sollte. Zur Beurteilung des Ergebnisses müssen damit performanztechnische Gesichtspunkte betrachtet werden. Diese sind über Timer und Iterationszähler in einem automatisierten Test implementiert worden. Die einzelnen Module der Pipeline müssen laufzeitmäßig betrachtet werden. Weiterhin muss das Verhalten auf unterschiedlich leistungsstarken technischen System untersucht werden. Während der Bachelorarbeit wurde als Standardsystem der Laptop aus 4.1 verwendet und als Minimalsystem ein *Raspberry Pi 3 Model B*, welches als Eingebettetes System angesehen werden kann. Die Performanzbetrachtungen beider technischer Systeme werden in diesem Kapitel gegenübergestellt.

Da die Pipeline eine Transkription von Musik durchführt, muss neben den Performanzbetrachtungen auch die Qualität der Transkription erfasst werden. Die Tonhöhenerkennung bietet zusammen mit der Tonlängenerkennung die Möglichkeit eine Melodie zu transkribieren. Weiterhin ermöglicht die Tonhöhenerkennung eine Ton- und Akkordidentifizierung ausgehend von einem akustischen Signal. Damit die Qualität der Transkription im Hinblick auf die Ton-, Akkord- und Melodieerkennung beurteilt werden kann, so muss das transkribierte Ergebnis der Pipeline mit einem Referenzwert verglichen werden. Dazu wurden mehrere zusätzliche Funktionen und Skripte implementiert, die automatisierte Tests darstellen.

Im Laufe der Bachelorarbeit wurden drei verschiedene Varianten der Pipeline entwickelt, wobei jede unterschiedliche Qualitäten bezüglich Performanz und Qualität der Transkription liefert. Die Performanz- und Qualitätsbetrachtungen wurden dabei für jede Variante einzeln vorgenommen und werden miteinander verglichen.
Für diesen Teil der Bachelorarbeit werden zunächst theoretische Betrachtungen zu den einzelnen performanz- und qualitätskritischen Größen durchgeführt. Anschließend wird auf die Ergebnisse der automatisierten Tests bezüglich Ton-, Akkord- und Melodieerkennung, sowie dem Zeitverhalten eingegangen.

5.2 Theoretische Betrachtungen

In der Pipeline gibt es einige Parameter, die performanz- sowie qualitätskritisch für die Transkription von Musik sind. Dies sind unter anderem das Tempo, die Samplegröße, die Schrittgröße und die Fensterfunktion. Für die Samplingrate wird eine konstante Größe angenommen, im allgemeinen Fall beträgt sie 44100 Hz.
Wird die Anzahl der Messwerte n verändert, so kann bei gleichbleibender Samplingrate Fs eine unterschiedliche Frequenzauflösung erreicht werden. Gleichzeitig gilt, dass eine größere Samplegröße n zu einem deutlich größeren Rechenaufwand für das ausführende technische System aufgrund der FFT führt. Bei einer Samplingrate $Fs = 44100\text{Hz}$, gilt außerdem, dass die FFT bei größerer Samplegröße n einen immer größeren Zeitbereich abdeckt. Es gilt also eine Samplegröße n zu finden, welche eine ausreichende Frequenzauflösung bietet, gleichzeitig jedoch nicht zu rechenintensiv für die zu verarbeitenden Systeme wird. Wird die Tabelle 2.1 betrachtet, so zeigt sich ein weiteres Merkmal: Je höher die Oktave ist, desto weiter liegen die einzelnen Frequenzen zwischen zwei Halbtönen auseinander.

Oktave	Δf [Hz]	n_{min}	n_{tats}	t [ms]	*tempo* [bpm]
0	1.32	33419	(2^{16})	1486.08	10
1	2.64	16710	2^{15}	743.04	20
2	5.28	8355	2^{14}	371.52	40
3	10.56	4178	2^{13}	185.76	80
4	21.11	2089	2^{12}	92.88	161
5	42.23	1045	2^{11}	46.44	322
6	84.46	523	2^{10}	23.22	645
7	168.91	262	2^{9}	11.61	1291
8	337.83	131	2^{8}	5.80	2583
9	675.66	66	2^{7}	2.90	5167
10	1351.32	33	2^{6}	1.45	10335

Tabelle 5.1: Durchschnittliche Frequenzauflösung Δf, minimale Samplinggröße n_{min}, tatsächliche Samplinggröße n_{tats}, Zeitintervall t der FFT, maximales Tempo bei einer 16-tel Notenauflösung für die entsprechende Oktave und einer Samplingrate $Fs = 44100\text{Hz}$

In Tabelle 5.1 sind die durchscnittlichen Frequenzunterschiede zwischen zwei Halbtönen pro Oktave aufgetragen. Da Halbtöne in der Musik für die Notation die kleinste zu unterscheidende Einheit darstellen, genügt es eine Frequenzauflösung entsprechend der Tabelle zu erreichen. Es ist zu erkennen, dass die Frequenzabstände sich von Oktave zu Oktave verdoppeln. Weiterhin wurde über das Frequenzintervall pro

Bin die minimal benötigte Samplegröße $n = \lceil \frac{Fs}{\Delta f} \rceil$ bestimmt, um die benötigte Frequenzauflösung pro Oktave zu erreichen. Da die eigentliche FFT lediglich für Samplegrößen mit einer Potenz von 2 ausgelegt ist, muss die jeweils nächst größere Zweierpotenz verwendet werden. Für Oktave 0 müsste also mit dem Radix-2-Algorithmus $n = 2^{16} = 65536$ Samples verwendet werden.

Weiterhin wurde berechnet wie groß das Zeitintervall der FFT für die entsprechende Samplegröße n ist: $t = \frac{n}{Fs}$. Damit zeigt sich auch, dass die FFT für eine größere Samplegröße n zwar immer einen kleineren Frequenzbereich pro Bin abbildet und damit die Frequenzauflösung gesteigert wird. Gleichzeitig bedeutet dies jedoch auch, dass ein immer größerer Zeitbereich in der FFT betrachtet wird. Dadurch nimmt die zeitliche Auflösung bei steigender Frequenzauflösung ab. Dies ist für die Musik insofern von Bedeutung, da Noten eine unterschiedliche Dauer aufweisen können. Die Frequenzen müssen also in einer bestimmten Zeit ausgewertet werden. Diese Tatsache fügt eine praktische Obergrenze für die Samplegröße n hinzu und damit eine Grenze für die Frequenzauflösung.

Ein schnelles Lied fängt bei einem Tempo von ungefähr 120 Schlägen pro Minute an. Bei einem $\frac{4}{4}$-Takt, bedeutet dies, dass 120 Viertel pro Minute das Tempo vorgeben. Für jede Viertel wird also eine Zeit von 0,5 Sekunden in Anspruch benötigt. Um ein Lied auf Viertel aufzulösen, muss also maximal alle 0,5 Sekunden eine FFT ausgewertet werden. Dies würde laut Tabelle die Samplegröße also auf 2^{14} begrenzen. Soll eine Achtelauflösung erreicht werden, dann muss die FFT alle 0,25s berechnet werden. In den meisten Liedern ist mindestens eine Zeitauflösung auf 16-tel Notenwerte nötig. Damit muss eine FFT alle 0,125s ausgewertet werden. Dies begrenzt die Sampelgröße n auf maximal 2^{12} Samples. Dies bedeutet, dass erst ab Oktave 4 die Töne mit der FFT zuverlässig aufgelöst werden können. Weiterhin muss ein Sicherheitspuffer für die Berechnungszeit der FFT selbst miteinbezogen werden. Diese kann je nach Leistungsfähigkeit des Systems unterschiedlich ausfallen. In der Tabelle 5.1 sind in der letzten Spalte jeweils die maximalen Tempi angegeben, um Töne in der entsprechenden Oktave für eine 16-tel Auflösung gerade noch auflösen zu können.
Es zeigt sich, dass besonders für die niedrigen Frequenzen das Tempo drastisch reduziert werden muss. Diese Tempowerte werden über die Formel $tempo = \frac{60}{\frac{t}{1000}} \cdot \frac{1}{4}$ berechnet. Der Faktor $\frac{1}{4}$ kommt dadurch zustande, dass eine 16-tel Note einem Viertel von einem Viertel entspricht. Die Tempoangabe *tempo* ist dabei in Schläge pro Minute angegeben, wobei ein Schlag in den meisten Fällen einer Viertel entspricht.

Um diese Metriken zu verbessern, können mehrere Dinge verändert werden: Zum einen kann das Tempo (Schläge pro Minute) reduziert werden. Damit würden für die entsprechenden Tonlängen mehr Zeit zur Verfügung bekommen. Dies würde wiederum bedeuten, dass eine größere Samplegröße n verwendet werden könnte und damit die Frequenzauflösung für einen größeren Notenbereich abzudecken. Weiterhin kann die Minimalauflösung von 16-tel auf 8-tel gesetzt werden. Auch wäre erneut mehr Zeit für die FFT übrig. Wie in der Tabelle 5.1 anschaulich demonstriert, liegt die

tatsächlich verwendete Samplegröße weit über der minimal benötigten Samplegröße. Der Grund hierfür liegt an der Voraussetzung des Radix-2-Algorithmus von Zweierpotenzen [26]. Dies könnte durch einen anderen FFT-Algorithmus gelöst werden, der diese Bedingung nicht erfüllen muss.

5.2.1 Performanz- und Qualitätskritische Parameter

Mithilfe der theoretisch gewonnenen Erkenntnisse werden im Folgenden sinnvolle Werte für die performanz- und qualitätskritischen Größen in der Pipeline festgelegt:

- **Samplerate *Fs*:** Die Samplerate Fs sollte auf 44100 Hz festgelegt werden. Damit kann der auditive Bereich des Menschen effizient abgedeckt werden. Das System sollte auch für andere Größen funktionieren, allerdings wird die Audioaufnahme langsamer und schneller. Die Performanzbetrachtungen dieser Bachelorarbeit beziehen sich immer auf eine Samplerate von $Fs = 44100$ Hz.
- **Samplegröße *n*:** Für die Ton- und Akkorderkennung ist ein möglichst hohe Samplegröße n zu empfehlen, da dadurch die Frequenzauflösung zunimmt. Für die Ton- und Akkorderkennung wurde beim Benchmarking eine Samplegröße von $n = 8192$ Samples gewählt. Die FFT kann damit Frequenzen bis auf $\frac{44100}{8192} = 5,38$ Hz auflösen. Damit konnten Töne von Oktave 3 aufwärts effizient identifiziert werden. Falls zusätzlich Zeitverhalten gefordert ist, wie zum Beispiel bei der Melodieerkennun, sollte ein niedrigere Samplegröße n gewählt werden. Im Benchmarking wurde eine Samplegröße von $n = 4096$ verwendet.
- **Schrittgröße *x*:** Über die Schrittgröße kann die Anzahl der FFT-Berechnungen gesteuert werden. Bei einer zu geringen Schrittgröße x kann es zu erheblichen Performanzeinbußen kommen. Für das Benchmarking wurde eine Schrittgröße von $x = 1024$ Samples verwendet. Damit wird alle $\frac{1024}{44100} = 0,023$ Sekunden eine FFT durchgeführt. Bei einem Tempo von 120 Schlägen pro Minute, muss die Pipeline alle $\frac{60}{120} \cdot \frac{1}{4} = 0,125$ Sekunden durchlaufen werden. Bei einer Schrittgröße von $x = 1024$ kann damit auf einem leistungsfähigen System ausreichend zeitlich aufgelöst werden, falls die eigentlichen Berechnungen nicht länger als $0,125 - 0,023 = 0,102$ Sekunden benötigt.
- **Tempo *tempo*:** Die meisten Lieder besitzen eine Tempoangabe unter 120 Schläge pro Minute.
- **Notenwertauflösung:** Die Pipeline sollte eine Notenwertauflösung auf 16-tel Ebene aufweisen, damit auch feinere Nuancen in den Melodien erkannt werden können.
- **Margin *margin*:** Für das Benchmarking wurde ein Margin von 40 cent festgelegt. Damit sollten Frequenzen, die ziemlich exakt zwischen zwei Notenwerten liegen vernachlässigt werden. Weiterhin wird dadurch die Notenerkennung robuster, da die exakte Frequenz mit einer endlichen Anzahl an Samples nicht bestimmt werden kann.

5.3 Benchmarking

Um ein vollständiges Benchmarking durchführen zu können, müssen sämtliche performanz- und qualitätsrelevanten Größen betrachtet werden. Diese sind die Samplegröße n, die Schrittgröße x, das Tempo *tempo*, die Aufnahmedauer t und die Fensterfunktion f_{win}. Es wird dabei von einer konstanten Samplerate $Fs = 44100$Hz ausgegangen. Dabei beschreiben n und x vor allem performanzrelevante Größen. Die Größen *tempo* und f_{win} sind qualitätsrelevante Größen.

5.3.1 Qualität der Transkription

In diesem Kapitel soll die Qualität der Pipeline in Bezug auf die Transkription beurteilt werden. Dazu wird zunächst betrachtet, wie gut das System einzelne Töne erkennen kann. Anschließend wird die Betrachtung auf unterschiedliche Akkorde erweitert und schließlich wird die Qualität der erkannten Melodie untersucht, bei der neben der Tonhöhe auch die Notenlänge eine Rolle spielt. Die Ergebnisse der Ton- und Akkorderkennung beruhen auf der sequentiellen Variante der Pipeline.

Tonerkennung

In Tabelle 5.2 sind die Ergebnisse des Ton-Benchmarkings aufgetragen. Damit die bestmögliche Qualität erzielt wird, muss die Frequenzauflösung maximiert werden. Daher wurde für das Noten-Benchmarking eine Samplegröße und Schrittgröße von $n = x = 8192$ gewählt. Es wurde für jede Oktave deren Frequenzbereich notiert, sowie die durchschnittliche Abweichung zwischen der theoretischen und gemessenen Frequenz in der Cent-Skala aufgetragen. Weiterhin wurde vermerkt wie viele Töne in der jeweiligen Oktave korrekt erkannt wurden. Beim Benchmarking wurde die Notenerkennung für alle Töne in den ersten sieben Oktaven durchgeführt. Außerdem wurde als Fensterfunktion beispielhaft das Gauß-Fenster verwendet und der erlaubte Cent-Margin wurde auf 40 cent festgelegt.

octave	Frequenzintervall Hz	Frequenzabweichung [cent]	korrekt erkannt	Erfolgsrate %
0	[16,35 ; 30,87]	805,4564	0	0
1	[32,70 ; 61,74]	377,3083	0	0
2	[65,41 ; 123,47]	303,3857	8	66,67
3	[130,81 ; 246,94]	29,1111	10	83,33
4	[261,63 ; 493,88]	16,0438	12	100
5	[523,25 ; 987,77]	6,2663	12	100
6	[1046,50 ; 1975,53]	3,8802	12	100

Tabelle 5.2: Ergebnisse des Ton-Benchmarkings.

Tabelle 5.2 zeigt, dass die Notenerkennung für höhere Oktaven eine größere Erfolgschance bietet. Bei niedrigen Frequenzen kann der Ton entweder zu leise sein [76] oder aber die Frequenzauflösung ist zu gering und damit die Samplegröße n zu gering. Mit einer Samplegröße von $n = 8192$ Samples ist es möglich die einzelnen Frequenzen auf eine Genauigkeit von $\Delta f = \frac{Fs}{n} = \frac{44100}{8192} = 5,38\text{Hz}$ aufzulösen. Laut Tabelle 5.1 ist eine effiziente Tonauflösung deshalb erst ab Oktave 3 möglich. Dies bestätigt sich auch anhand der Tabelle 5.2. Die durchschnittliche Abweichung liegt bei ungefähr 30 cent und es konnten 10 von 12 Töne in der Oktave erkannt werden. Damit liegt die durchschnittliche Abweichung unter der festgelegten Cent-Margin von 40 cent. Für niedrigere Oktaven ist die Notenerkennung zunehmend schlechter. Dies trifft vor allem auf die ersten zwei Oktaven zu. In diesen ist die Auflösung der FFT zu ungenau und es werden keine Töne richtig erkannt. Für größere Oktaven, zeichnet sich das exakte Gegenteil ab. Die Tonerkennungsrate liegt bei 100% und damit werden 12 von 12 Töne korrekt erkannt. Die durchschnittliche Abweichung wird zunehmend kleiner.

Akkorderkennung

Auch für die Akkorderkennung soll eine möglichst gute Frequenzauflösung ermöglicht werden. Die zeitliche Auflösung spielt dabei keine Rolle. Die Samplegröße und Schrittgröße wurden erneut auf $n = x = 8192$ Samples festgelegt. Auch für das Akkord-Benchmarking wurde ein Cent-Margin von 40 cent und das Gauß-Fenster als Fensterfunktion gewählt. Es wurden für jede Anzahl an Tönen pro Akkord 15 Akkorde zufällig generiert. Das Benchmarking hat die Akkorderkennung für Akkorde zwischen 2 und 5 Tönen gleichzeitig getestet. Dabei wurden die einzelnen Töne zufällig ausgewählt, wobei lediglich Töne aus den Oktaven 3,4 und 5 gewählt wurden. Eine größere Tonspanne wäre unrealistisch für ein Akkord.

Töne pro Akkord	Abweichung [cent]	Anzahl Ausreißer	Abweichung ohne Ausreißer [cent]	korrekt erkannt	Anzahl Töne	Erfolgsrate %
2	48,92	2	18,55	27	30	90,00
3	79,56	3	15,55	41	45	91,11
4	77,54	5	17,84	54	60	90,00
5	40,85	2	19,26	67	75	89,33

Tabelle 5.3: Ergebnisse des Akkord-Benchmarkings für jeweils 15 zufällig generierte Akkorde.

Es zeigt sich unabhängig von der Anzahl der Töne pro Akkord, dass die Akkorderkennung eine Erfolgsrate von um die 90% besitzt. Ungenauigkeiten können durch unterschiedliche wahrgenommenen Lautstärken für unterschiedliche Tonhöhen entstehen [76]. Weiterhin können Fenstereffekte das Ergebnis verschlechtern.
Die Akkorderkennung wird vor allem dahingegend gemindert, dass es einzelne Ausreißer bei der Erkennung geben kann. Diese ziehen die durchschnittliche Abweichung zwischen gemessenen Frequenzen und theoretisch erwarteten Frequenzen so weit nach oben, dass diese über dem maximalen Cent-Margin von 40 cent liegt. Als Ausreißer sind in diesem Beispiel jene gerechnet, welche eine Abweichung von mehr als 100 cent aufweisen und damit mehr als ein Halbton auseinander liegen. Werden diese Ausreißer jeweils für die Messungen herausgerechnet, so ergibt sich eine durchschnittliche Abweichung von weniger als 20 cent, unabhängig von der Anzahl Töne im Akkord. Damit liegt die Abweichung unter der festgesetzten Schranke von 40 cent.
In Tabelle 5.3 zeigt sich, dass die Pipeline eine relativ robuste Lösung für die Akkorderkennung ist. Es sei angemerkt, dass die Ergebnisse sich lediglich auf reine Sinusschwingungen beziehen. Für unterschiedliche Instrumente ist die Akkorderkennung um ein Vielfaches komplexer, da neben der Hauptfrequenz des Tones auch noch sämtliche Oberfrequenzen mitschwingen, die für jedes Instrument verschieden sind.

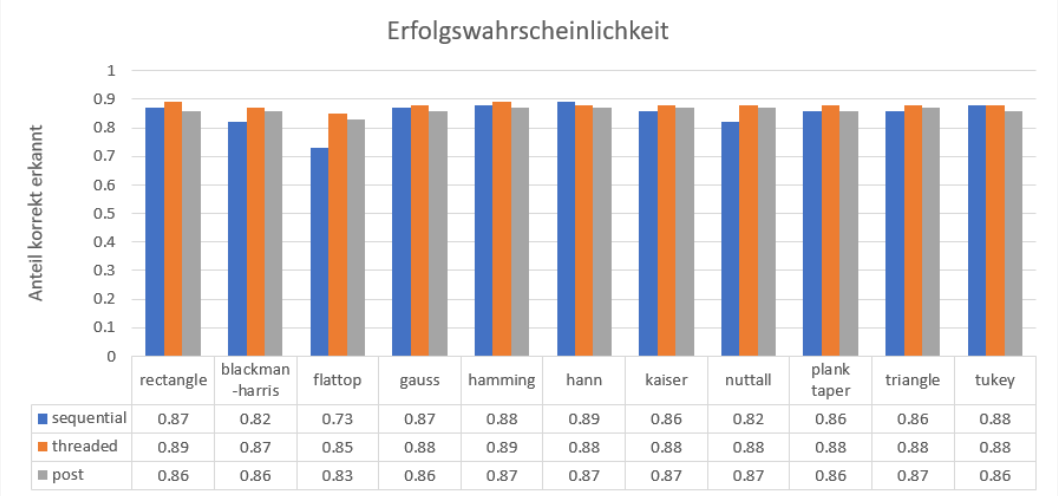
Melodieerkennung

Aus performanztechnischen Gründen wurde für das Melodiebenchmarking eine Samplegröße von $n = 4096$ Samples verwendet. Weiterhin wurde eine Schrittgröße von $x = 1024$ Samples gewählt. Alle $\frac{1024}{44100} = 0,023$ Sekunden wird damit eine Transkription durchgeführt. Als MIDI-Instrument wurde *voice oohs* verwendet. Weiterhin wurde das Benchmarking auf fünf zufällig generierte Melodien, sowie auf fünf in einer Textdatei abgespeicherten manuell erstellten Melodien angewendet.

Das Benchmarking wurde für jede Melodie mit elf Fensterfunktionen getestet. Als

Fensterfunktionen wurde das Rechtecksfenster (*rectangle*), das Blackman-Harris Fenster (*blackman-harris*), das Flat Top Fenster (*flattop*), das Gauß Fenster (*gauss*), das Hamming Fenster (*hamming*), das Hann Fenster (*hann*), das Kaiser Fenster (*kaiser*), das Nuttall Fenster (*nuttall*), das Planck-Taper Fenster (*planck-taper*), das Dreiecksfenster (*triangle*), sowie das Tukey Fenster (*tukey*) Fenster verwendet [34]. Für jede Melodie und Fensterfunktion wurden weiterhin Testdurchläufe mit 60, 90 und 120 Schlägen pro Minute als Tempoangabe durchgeführt. Diese Einstellungen wurden jeweils unter gleichen Bedingungen für alle drei Pipeline-Varianten angewendet.

In Abbildung 5.1 ist die Erfolgswahrscheinlichkeit der Melodieerkennung in Abhängigkeit von elf Fensterfunktionen und den drei Ausführungsvarianten der Pipeline aufgetragen.



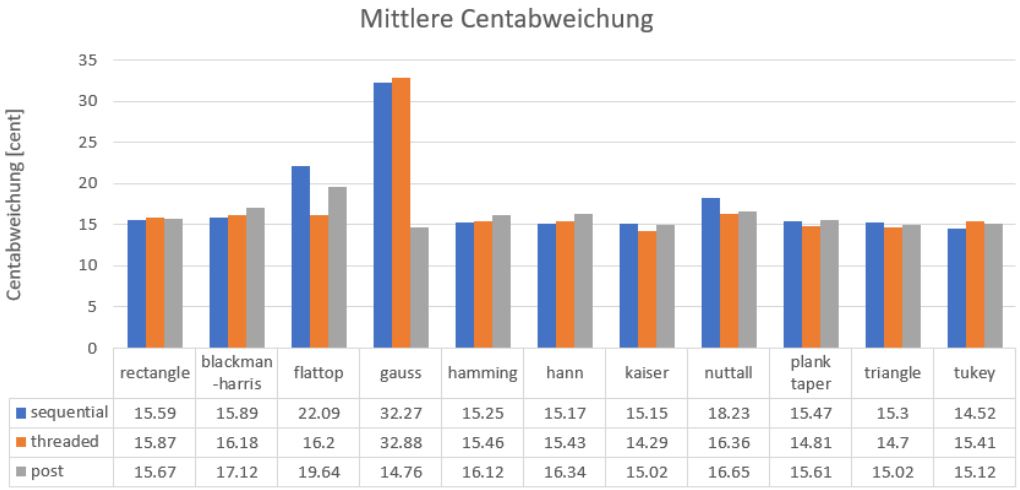
	rectangle	blackman -harris	flattop	gauss	hamming	hann	kaiser	nuttall	plank taper	triangle	tukey
sequential	0.87	0.82	0.73	0.87	0.88	0.89	0.86	0.82	0.86	0.86	0.88
threaded	0.89	0.87	0.85	0.88	0.89	0.88	0.88	0.88	0.88	0.88	0.88
post	0.86	0.86	0.83	0.86	0.87	0.87	0.87	0.87	0.86	0.87	0.86

Abbildung 5.1: Erfolgswahrscheinlichkeit für die Melodieerkennung für verschiedene Fensterfunktionen unterschieden nach den drei Varianten.

Abbildung 5.1 zeigt, dass keine der Fensterfunktionen die Transkriptionsgenauigkeit zu sehr verschlechtern. Die Erfolgswahrscheinlichkeit liegt bei allen Fensterfunktionen zwischen 80% und 90%. Auch das Rechteckfenster *rectangle* erreicht eine sehr gute Transkriptionsqualität, wobei im Prinzip kein Fenster auf die Audiodaten angewendet wird. Die Fensterfunktionen *blackman-harris*, *flattop* und *nuttall* weisen die schlechtesten Transkriptionsergebnisse auf. Die Wahrscheinlichkeiten könnten für andere Melodien, größere Tonsprünge, oder kleinere Schrittgrößen anders ausfallen.

Eine Melodie ist grundlegend aus Tonhöhe und Tonlänge aufgebaut. Die Qualität der Melodieerkennung kann anhand dieser zwei Größen beurteilt werden. Es müssen im Laufe des Benchmarkings Tonhöhen- und Tonlängenunterschiede zwischen ermittelter Melodie und Testmelodie bestimmt werden. Für die Tonhöhenerkennung werden Frequenzunterschiede in der Cent-Skala festgehalten, damit unterschiedliche Töne aus unterschiedlichen Oktaven miteinander vergleichbar sind. Beim Tonlän-

genvergleich genügt die Differenz in Millisekunden fest zu halten.
Für die folgenden Betrachtungen wurden lediglich Melodien miteinbezogen, welche Töne ab Oktave 4 enthalten, da die Auflösung für darunter liegende Oktaven aufgrund von 5.1 nicht garantiert werden kann. Als maximaler Cent-Margin wurden 40 cent festgelegt. Damit entsprechen sich zwei Frequenzen tonweise lediglich einander, falls diese weniger als 40 cent auseinander liegen. Im Folgenden sind die gemittelten Frequenzunterschiede in der Cent-Skala für elf Fensterfunktionen und den drei Pipeline-Varianten aufgetragen.

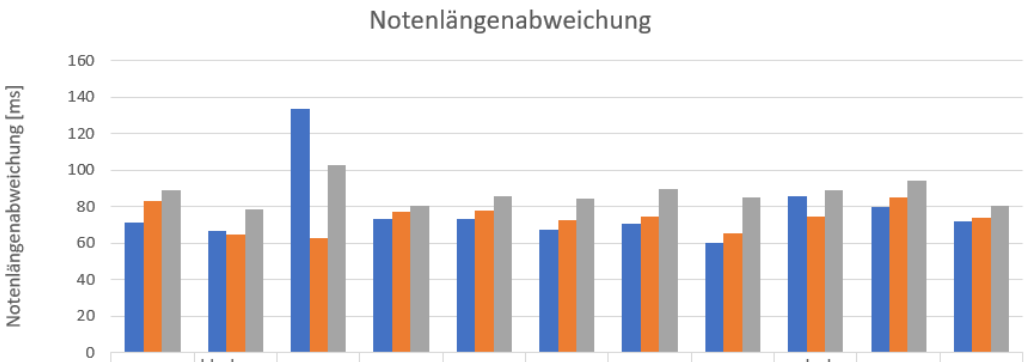


	rectangle	blackman -harris	flattop	gauss	hamming	hann	kaiser	nuttall	plank taper	triangle	tukey
sequential	15.59	15.89	22.09	32.27	15.25	15.17	15.15	18.23	15.47	15.3	14.52
threaded	15.87	16.18	16.2	32.88	15.46	15.43	14.29	16.36	14.81	14.7	15.41
post	15.67	17.12	19.64	14.76	16.12	16.34	15.02	16.65	15.61	15.02	15.12

Abbildung 5.2: Gemittelte Centabweichung in cent.

Abbildung 5.2 zeigt erneut ein größtenteils einheitliches Verhalten, welches unabhängig von den Fensterfunktionen ist. Für die Fensterfunktionen *flattop*, *nuttall* und *gauss* ergeben sich größere Frequenzunterschiede als bei den anderen Fensterfunktionen. Auch an dieser Stelle schneidet die Fensterfunktion *rectangle* äußerst positiv ab. Weiterhin fällt für alle Fensterfunktionen allerdings auf, dass die gemittelte Centabweichung unter dem maximalen Margin von 40 cent liegt. Damit erkennen die Fensterfunktionen im Allgemeinen die Melodien einigermaßen gleich gut.

Als weitere kritische Größe gilt die Notenlängendifferenz aus erkannter Melodie und der Testmelodie. Die Differenz wird in Millisekunden festgehalten und ist gemittelt in Abhängigkeit von elf Fensterfunktionen und der drei Pipeline-Varianten in folgender Abbildung zu sehen:



	rectangle	blackman -harris	flattop	gauss	hamming	hann	kaiser	nuttall	plank taper	triangle	tukey
sequential	71.33	66.76	133.94	73.5	73.22	67.39	70.33	60.33	85.98	79.61	71.59
threaded	82.92	64.54	62.94	76.85	77.5	72.72	74.52	65.46	74.44	85.27	73.93
post	88.85	78.45	102.44	80.44	85.95	84.2	89.43	84.88	89.07	94.13	80.64

Abbildung 5.3: Gemittelte Notenlängendifferenz in ms.

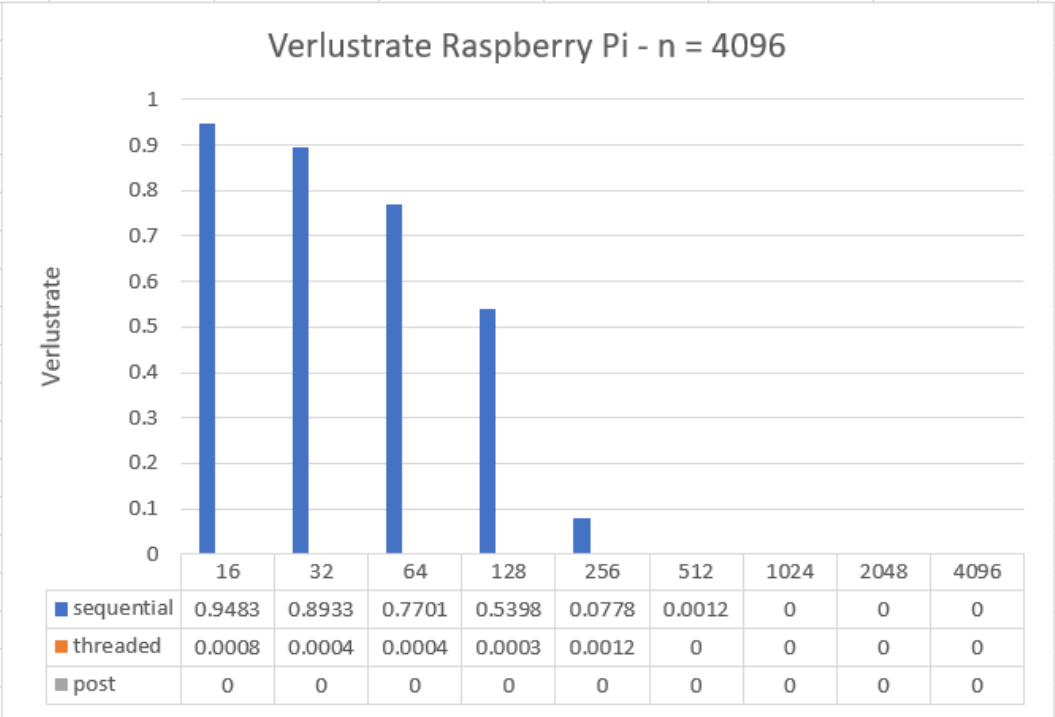
Abbildung 5.3 zeigt erneut, dass die verschiedenen Fensterfunktionen auch notenlängentechnisch einheitlich transkribieren. Dabei spielt erneut die Fensterfunktion *flattop* eine Ausnahme dar, da die Notenlänge der transkribierten Töne an dieser Stelle zum Teil über 100 Millisekunden von den Notenlängen der Töne der Testmelodie abweichen. Auch die Fensterfunktionen *plank-taper* und *triangle* weisen schlechtere Ergebnisse als die restlichen Funktionen auf. Ein besonders gutes Verhalten bieten die Fensterfunktionen *blackman-harris* und *nuttall*. Für diese Metrik befindet sich die Fensterfunktion *rectangle* nicht bei den besten Fensterfunktionen. Es zeigt sich, dass die sequentielle Verarbeitung mit Zwischenspeicherung *post* für alle Fensterfunktionen bis auf *flattop* zeitlich schlechter auflöst, als die anderen beiden Varianten.
Bei einem Tempo von 120 Schlägen pro Minute benötigt eine 16-tel Note 0,125 Sekunden. Für diese Größe würden alle, bis auf die Fensterfunktion *flattop* eine genügende zeitliche Auflösung für die STFT erreichen.

Es zeigt sich damit, dass für eine $n = 4096$ und $x = 1024$ eine ausreichende zeitliche und tontechnische Auflösung erreicht wird, unabhängig von der verwendeten Fensterfunktion. Dabei weisen vor allem die Fensterfunktion *gauss* und *flattop* eine schlechtere Tontechnische Auflösung als die anderen Funktionen. Auch bei der Notenlängendetektion ist von *flattop* abzuraten, da diese die schlechtesten Ergebnisse liefert. Das Rechteckfenster liefert erstaunlicherweise sehr gute Ergebnisse. Dies mag angesichts kleinerer Schrittgrößen x und Samplegrößen n unterschiedlich ausfallen. Weiterhin schneiden die drei Pipeline-Varianten größtenteils einheitlich für die Fensterfunktionen ab. Dies entspricht auch den theoretischen Erwartungen.

5.3.2 Betrachtung der Performanz

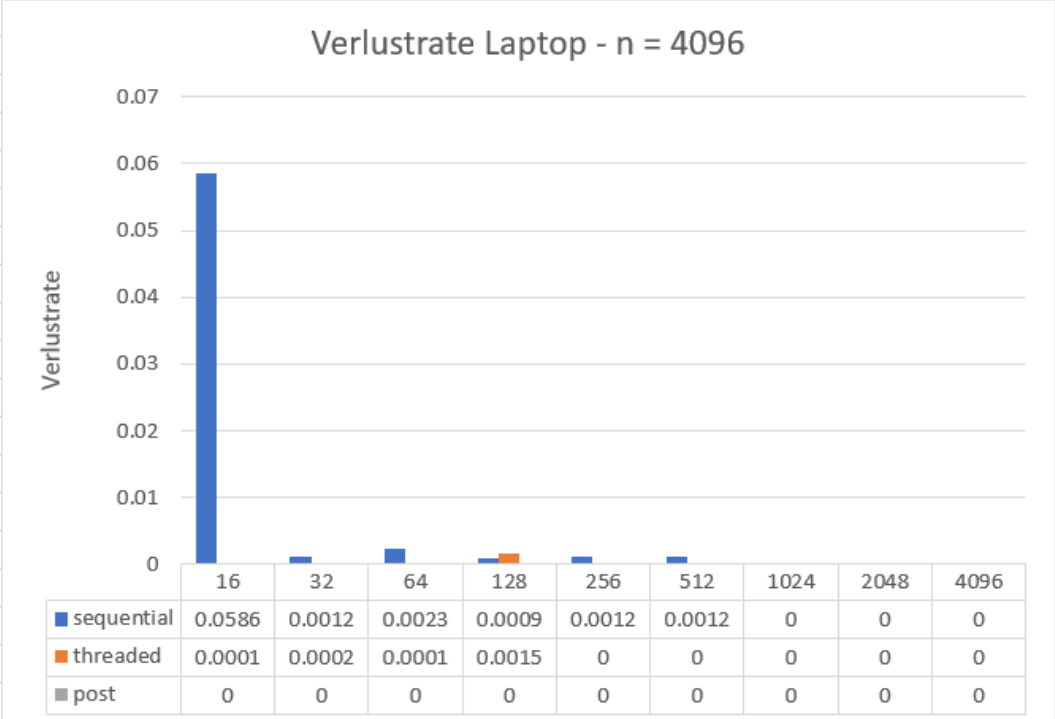
Zur Beurteilung der Performanz des Systems wurde ein automatisierter Test geschrieben, welcher jede der Betriebsmodi durchläuft und in jedem Durchgang die Samplegröße n und die Schrittgröße x anpasst. Es wurden die Zeiten für die jeweiligen Ausführungsteile der Pipeline gemessen, sowie die Anzahl der Durchläufe durch die Pipeline. Die Ergebnisse, die im Folgenden angeführt werden, beziehen sich alle auf eine Aufnahmezeit von 10 Sekunden. Als Samplegröße soll ihr beispielhaft die Werte für $n = 4096$ Messdaten verwendet werden. Der Grund hierfür ist, dass man mit 4096 Bins eine Frequenzauflösung von $\Delta f = \frac{n}{Fs} = \frac{4096}{44100} = 10,77\text{Hz}$ erreicht, welches zur Auflösung der meisten Töne genügt.

Mithilfe der gemessenen Anzahl an Durchläufen durch die Pipeline kann zusammen mit der theoretisch maximal zu erreichenden Anzahl an Durchläufen die Verlustrate berechnet werden. Dabei kann zwischen unterschiedlichen Betriebsmodi (sequential, threaded und post) unterschieden werden. *sequential* betrachtet den normalen Betriebsmodus mit sequentieller Ausführung. *threaded* untersucht die Variante mit paralleler Verarbeitung und *post* beschreibt das Zeitverhalten von der Variante mit Speicherung der Audiodaten in einer WAV-Datei. Abbildung 5.4 beschreibt die Verlustrate von zwei unterschiedlichen Systemen (4.1) in Bezug auf die drei Betriebsmodi und in Abhängigkeit von verschiedenen Schrittgrößen x.



	16	32	64	128	256	512	1024	2048	4096
sequential	0.9483	0.8933	0.7701	0.5398	0.0778	0.0012	0	0	0
threaded	0.0008	0.0004	0.0004	0.0003	0.0012	0	0	0	0
post	0	0	0	0	0	0	0	0	0

(a) Verlustrate Raspberry Pi für verschiedene Schrittgrößen x



	16	32	64	128	256	512	1024	2048	4096
sequential	0.0586	0.0012	0.0023	0.0009	0.0012	0.0012	0	0	0
threaded	0.0001	0.0002	0.0001	0.0015	0	0	0	0	0
post	0	0	0	0	0	0	0	0	0

(b) Verlustrate des Laptops für verschiedene Schrittgrößen x

Abbildung 5.4: Verlustraten von Minimal- und Standardsystem bei einer Samplegröße $n = 4096$.

Es zeigt sich, dass je kleiner die Schrittgröße x ist, desto größer ist die Verlustrate. Dies ist damit zu erklären, dass alle $\frac{x}{Fs}$ Sekunden eine FFT-Berechnung durchgeführt wird. Eine kleinere Schrittgröße x bedeutet damit, dass eine höhere Anzahl an FFT-Berechnungen durchgeführt werden müssen. Diese sind sehr zeitaufwendig und können den Ablauf der Pipeline soweit behindern, dass es zu großen Verlusten in Bezug auf augenommene Audiosamples kommt. Je größer die Schrittgröße, desto verlustfreier ist die Pipeline.

Die sequentielle Variante *sequential* ist laufzeittechnisch sehr anfällig. Dies liegt daran, dass die Audioaufnahme von Audiodaten jeweils auf die Berechnung der FFT des vorigen Pipelineschrittes warten muss. Damit werden weniger Audiodaten aufgenommen und es kommt zu erheblichen Verlusten. Die beiden anderen Varianten *threaded* und *post* sind weitestgehend robust gegenüber Verlusten, da die Audioaufnahme nicht auf die restliche Verarbeitung der Pipeline warten muss. Die Variante *threaded* besitzt bei niedrigen Schrittgrößen x eine vernachlässigbare Verlustrate, die vor allem dadurch bedingt ist, dass es zu kurzen Wartezeiten kommen kann, falls beide Threads gleichzeitig auf die Warteschlange zugreifen wollen. Die Variante *post* weist unabhängig von der Schrittgröße x keinerlei Verluste auf. Dies entspricht der Annahme, dass die Audioaufnahme vollkommen unabhängig und getrennt von der eigentlichen Verarbeitung stattfindet. Die Abspeicherung der Audiodaten ist schnell genug, sodass es zu keinen unnötigen Wartezeiten vonseiten der Audioaufnahme kommt. Die Variante *post* bietet damit eine robuste verlustfreie Variante zur Verarbeitung von Audiodaten. Dies kommt allerdings mit dem Preis, dass keine Echtzeit geboten werden kann.

Wird das Verhalten der beiden Systeme Laptop und Raspberry Pi verglichen, so fällt auf, dass der Laptop eine viel niedrigere Verlustrate aufweist, als der Raspberry Pi. Dies lässt sich mit der Rechenleistung der beiden technischen Systeme begründen. Bei einer Schrittgröße von $x = 16$ Samples, kommt es beim Raspberry Pi zu einer Verlustrate von über 90%. Beim Laptop beträgt die Verlustrate lediglich um die 6%. Damit eine einigermaßen robuste verlustfreie Verarbeitung garantiert werden kann, sollte beim Raspberry Pi eine Schrittgröße größer als $x = 512$ Samples gewählt werden. Für den Laptop sollte die Schrittgröße größer als $x = 32$ sein.
Diese Werte gelten für eine Samplegröße von $n = 4096$ Samples und einer Aufnahmezeit von $t = 10$ Sekunden. Die Ergebnisse sollten aufgrund der relativen Berechnung der Werte unabhängig von der Aufnahmezeit sein. Bei Variation von Schrittgrößen n kann entsprechend analog vorgegangen werden.

Ein weitere Faktor, der betrachtet werden muss, ist der zeitliche Mehraufwand (*overhead*) der einzelnen Betriebsmodi. Der Overhead wurde mithilfe der folgenden Formel berechnet:

$$o = \frac{overallRunTime}{recordingTime} \tag{5.1}$$

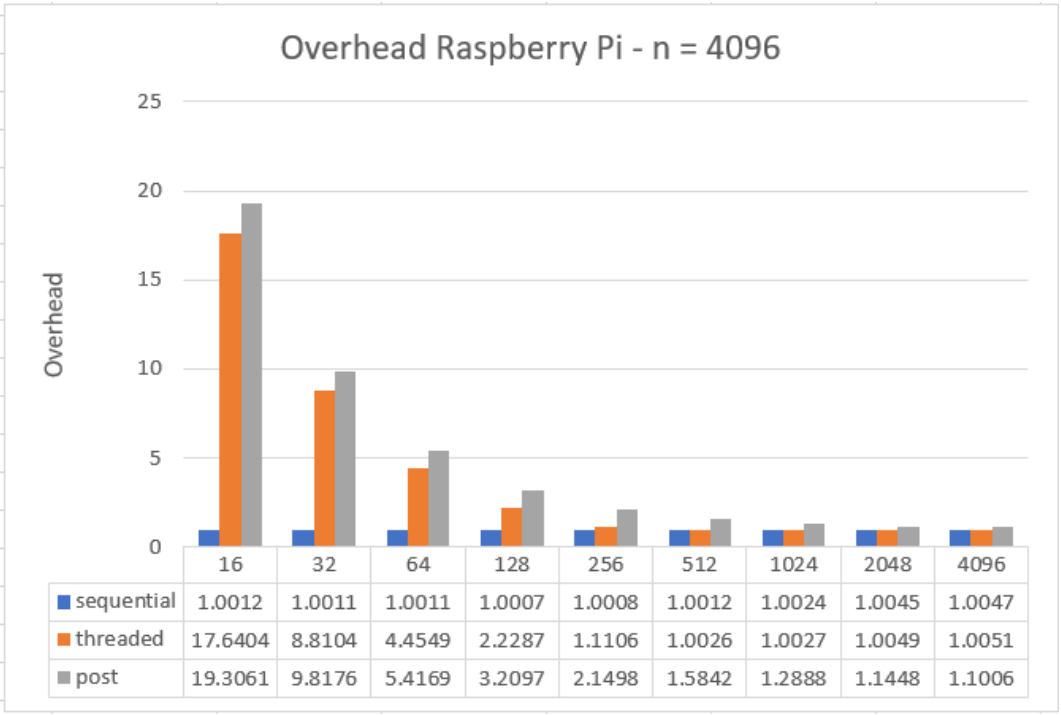
Dieser Faktor beschreibt wie viel größer die Laufzeit des jeweiligen Betriebsmodus ($overallRunTime$) im Vergleich zur Aufnahmezeit ($recordingTime$) ist. In Abbildung 5.5 sind die Werte für o für unterschiedliche Schrittgrößen x bei einer Samplegröße

$n = 4096$ für zwei unterschiedlicen Systeme aufgetragen.

In Abbildung 5.5 zeigt sich, dass die sequentielle Version die einzige tatsächliche echtzeitfähige Variante der Pipeline darstellt. Dies kommt allerdings mit dem Preis, dass die Variante *sequential* die größte Verlustrate der drei Betriebsmodi aufweist. Die Werte für *sequential* beschreiben nicht exakt einen Faktor von $o = 1$, da die Berechnungszeit des letzten Pipelinedurchgangs miteinbezogen werden muss.

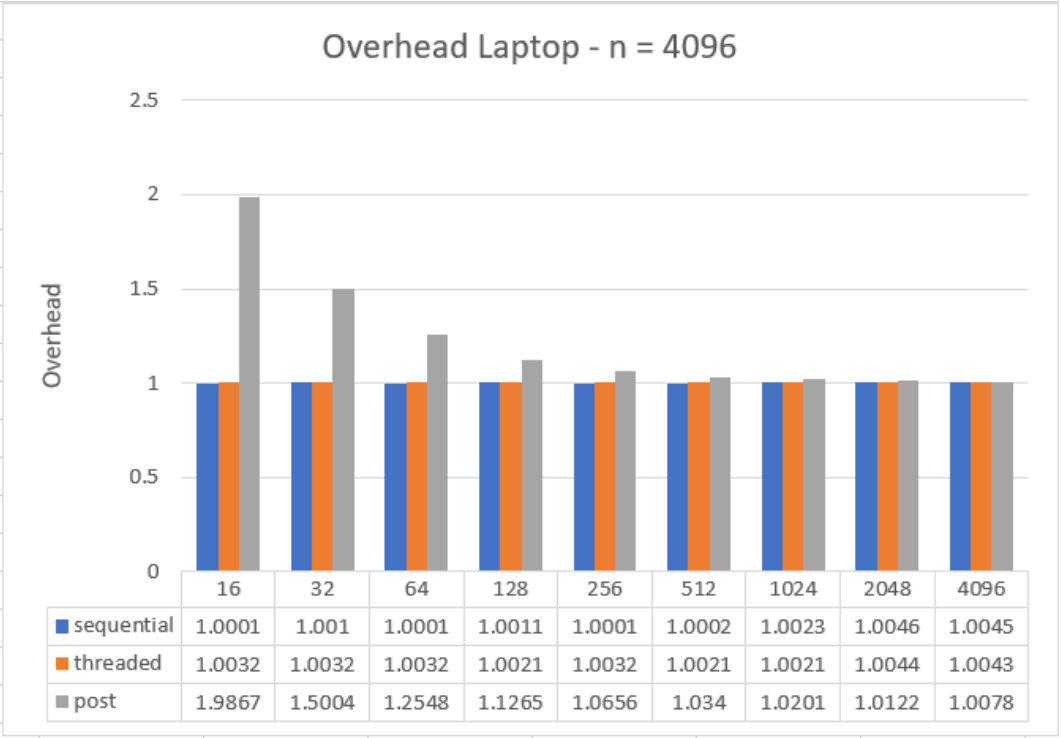
Bei der Variante *post* zeigt sich, dass die Speicherung der Audiodaten in der WAV-Datei mit anschließender Verarbeitung der Audiodaten große Einbußen in Bezug auf die Laufzeit hat. Es gehen zwar keine Daten verloren, allerdings hat diese Variante nichts mit einem echtzeitfähigen System zu tun. Beim Raspberry Pi ist die Laufzeit bei einer Schrittgröße von $x = 16$ bis um den Faktor 20 größer als die eigentliche Aufnahmezeit. Beim Laptop liegt der Faktor lediglich bei 2.

Der Mittelweg zwischen *sequential* und *post*, bietet die Variante *threaded*. Dieses Verfahren ist keine wirkliche Echtzeitverarbeitung, da die Verarbeitung auf der Warteschlange operiert. Für den Laptop errechnet sich allerdings ein Faktor, der praktisch 1 ist und damit einer Echtzeitverarbeitung entspricht. Für den Raspberry Pi zeigt sich allerdings deutlich, dass die Audioverarbeitung nicht immer dem Befüllen der Warteschlange hinterherkommt. Bei einer Schrittgröße von $x = 16$ liegt der Faktor der Variante *threaded* bei $o = 17,5$. Die Schrittgröße x sollte laut Abbildung 5.5a eine minimale Größe von $x = 512$ Samples betragen.



	16	32	64	128	256	512	1024	2048	4096
sequential	1.0012	1.0011	1.0011	1.0007	1.0008	1.0012	1.0024	1.0045	1.0047
threaded	17.6404	8.8104	4.4549	2.2287	1.1106	1.0026	1.0027	1.0049	1.0051
post	19.3061	9.8176	5.4169	3.2097	2.1498	1.5842	1.2888	1.1448	1.1006

(a) Overhead des Raspberry Pi für verschiedene Schrittgrößen x



	16	32	64	128	256	512	1024	2048	4096
sequential	1.0001	1.001	1.0001	1.0011	1.0001	1.0002	1.0023	1.0046	1.0045
threaded	1.0032	1.0032	1.0032	1.0021	1.0032	1.0021	1.0021	1.0044	1.0043
post	1.9867	1.5004	1.2548	1.1265	1.0656	1.034	1.0201	1.0122	1.0078

(b) Overhead des Laptops für verschiedene Schrittgrößen x

Abbildung 5.5: Overhead von Minimal- und Standardsystem bei einer Samplegröße $n = 4096$.

In Abbildung 5.4 und Abbildung 5.5 wurde von einer Sampelgröße $n = 4096$ ausgegangen. Die Pipeline besteht aus den Teilen Audioaufnahme, Audiovorverarbeitung, Transkription und Notenblattgenerierung. Diese Teile tragen unterschiedliche Anteile an der Gesamtlaufzeit. Dabei zeigt sich vor allem, dass die Audiovorverarbeitung einen Großteil an der Laufzeit ausmacht. Insbesondere die darin enthaltene FFT trägt zu einem Großteil für den Overhead bei der Verarbeitung von *sequential* bei. Im Folgenden sind die Laufzeitanteile der FFT-Berechnungen für den Laptop und den Raspberry Pi gemittelt über alle drei Betriebsmodi für unterschiedliche Sampelgrößen n aufgetragen.

Samplegröße n	Anteil FFT Raspberry Pi	Anteil FFT Laptop
128	0,3331	0,2142
256	0,3739	0,2364
512	0,4067	0,2607
1024	0,4414	0,2743
2048	0,4728	0,2988
4096	0,5137	0,3174
8192	0,5486	0,3403

Tabelle 5.4: Gemittelter Laufzeitanteil der FFT-Berechnung für verschiedene Samplegrößen n.

In Tabelle 5.4 wird verdeutlicht, dass je größer die Samplegröße n ist, desto größer ist auch der Anteil der FFT-Berechnung an der Gesamtlaufzeit. Dies deckt sich mit den Laufzeitbetrachtungen der FFT im Grundlagenteil. Es zeigt sich auch, dass der Anteil der FFT beim Laptop einen viel kleineren Anteil an der Gesamtlaufzeit ausmachen. Die Tabelle 5.4 bestätigt, dass die FFT einen Großteil der Laufzeit ausmacht.

5.4 Robustheit

Das entwickelte System ist dazu in der Lage beliebige Größen zur Basis 2 für die Samplegröße n und die Schrittgröße x zu wählen. Damit kann die Performanz und die Qualität frei gewählt werden. Dadurch kann das System auch für Eingebettete Systeme angepasst werden.
Das System kann durch seinen modularen Aufbau in automatisierte Tests integriert werden un dadurch verschiedene Größen getestet werden. Weiterhin kann das System mit verschiedenen Mikrofonen verwendet werden. Damit bietet die Pipeline eine robuste Lösung für bestimmte performanztechnische Merkmale. Auch können unterschiedliche Fensterfunktionen verwendet werden.
Allerdings bietet das System keine Qualitätsgarantie für die Transkription von un-

terschiedlichen Instrumenten. Die Komplexität ist für solche Einsätze zu hoch für die entwickelte Pipeline. Das System funktioniert für einfrequentige Signale.

5.5 Skalierbarkeit

Das System verfügt über drei Betriebsmodi 4.3. Dabei kann vor allem die Variante *post*, um intelligentere und rechenintensivere Algorithmen erweitert werden, da diese Variante unabhängig von Zeit eine maximale Qualität der Transkription bietet. Weiterhin ist die Pipeline in grob vier Module aufgeteilt: Audioaufnahme, Audiovorverarbeitung, Transkription und Notenblattgenerierung. Durch diesen modularen Aufbau kann das System relativ einfach skaliert werden und auf die eigenen Bedürfnisse zugeschnitten werden. Die Pipeline kann an für sich um weitere Module erweitert werden, wobei für diese Fälle die Betriebsmodi angepasst werden können.

Weiterhin ist durch die verschiedenen Benchmarking-Tests eine skalierbare Möglichkeit geboten das System für unterschiedliche Instrumente, Tempi, unterschiedliche Melodien, Töne und Akkorde effizient zu testen (4.4).
Aktuell basiert die FFT-Berechnung auf dem Radix-2-Algorithmus [26] und damit müssen die Samplegröße n und die Schrittgröße x Größen zur Basis 2 aufweisen. Durch den Bluestein-Algorithmus könnten die beiden Größen weiter angepasst werden [29].

5.6 Limitierungen des Systems

An dieser Stelle soll es um die Einschränkungen des aktuellen Systems gehen. Besonders die Komplexität der Transkription von mehreren Instrumenten soll genauer untersucht werden.

5.6.1 Klangfarbe von Instrumenten

Der aktuelle Stand des Systems lässt maximal eine Melodieerkennung mithilfe von klaren relativ einfrequentigen Signalen zu. Das System bietet keine Möglichkeit die Transkription auf Instrumente anzuwenden. Im Folgenden wird auf die Gründe dieser Beobachtung eingegangen.

Wird auf unterschiedlichen Instrumenten ein Ton gespielt, so handelt es sich um den an für sich selben Ton, allerdings hören sich die Instrumente unterschiedlich an. Dieses unterschiedliche Auftreten von Instrumenten wird als Klangfarbe eines Instrumentes bezeichnet.[10]
In Abbildung 5.6 ist das unterschiedliche Klangverhalten von vier verschiedenen Instrumenten bezüglich des Tones $C4$ aufgetragen [77].

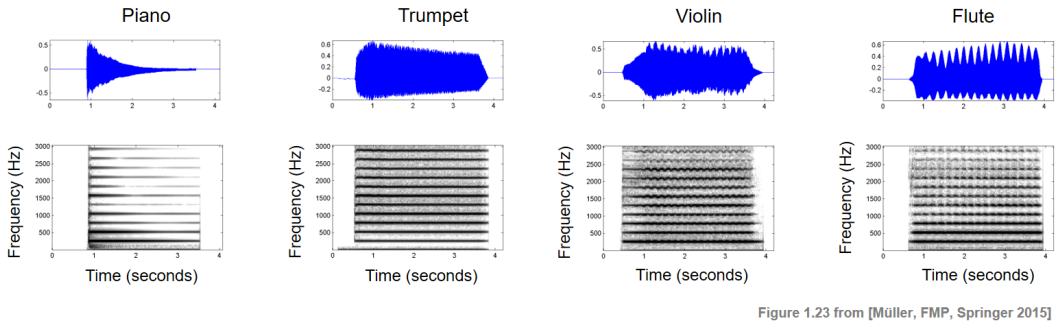


Abbildung 5.6: Klangfarben von unterschiedlichen Instrumenten für den gleichen Ton.[77]

Die unterschiedliche Klangfarbe lässt sich durch weitere Schwingungen erklären, die neben der Hauptfrequenz im Signal mitschwingen und von Instrument zu Instrument verschieden sind. Diese überlagern sich und ergeben den entsprechenden Klang des Instruments.[77] Tonanteile, die mit höherer Frequenz schwingen, werden als Obertöne bezeichnet. Eine Untermenge von Obertönen sind die Harmonischen, die eine ganzzahlig vielfache Frequenz der Grundfrequenz aufweisen. Bei Instrumenten sind meistens Harmonische enthalten. Harmonische Frequenzen werden generell als angenehm für das menschliche Ohr empfunden. Schwingungen, die mit niedrigerer Frequenz als der Grundton schwingen, werden als Untertöne bezeichnet. Diese spielen allerdings nur eine untergeordnete Rolle für diese Arbeit, da sie in den meisten Instrumenten nicht auftreten.[78]

ASDR

Eine weitere Begründung für das unterschiedliche Auftreten der Instrumente ist der sogenannte Amplitude Envelope oder ADSR (Attack,Decay,Sustain,Release) Envelope [79]. Ein Beispiel-ASDR Envelope kann in Abbildung 5.7 gefunden werden [80]. In der Attack-Phase wird der Ton aufgebaut und die Amplitude des Signals steigt auf ihren maximalen Wert. Anschließend folgt die Decay Phase, in der die Amplitude auf einen konstanten Wert sinkt, der dann in Sustainphase gehalten wird. Am Ende eines Tones sinkt die Amplitude wieder auf 0. Der ADSR Envelope beschreibt damit eine Note für ein bestimmtes Instrument über einen bestimmten Zeitraum. Die Harmonischen und Obertöne zusammen mit dem Amplitudenenvelope geben dem Instrument also eine bestimmte Klangfarbe. Unterschiedliche Instrumente besitzen unterschiedliche ASDR-Envelopes (Abb. 5.6).

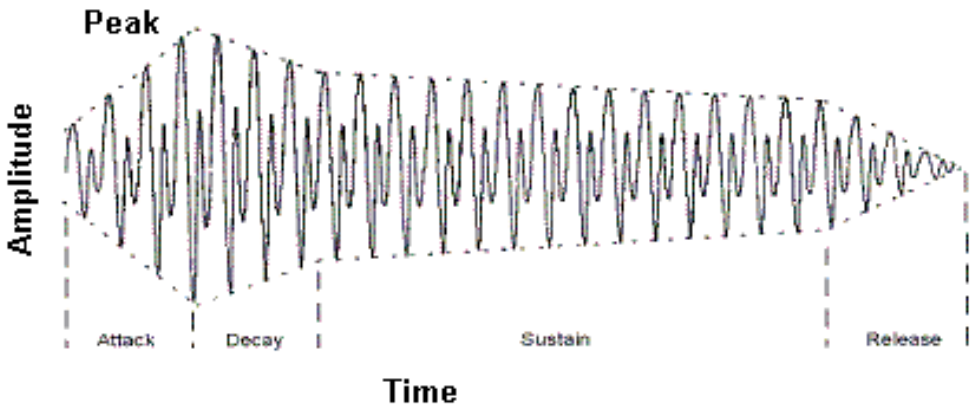


Abbildung 5.7: ASDR Envelope. [80]

Damit unterschiedliche Instrumente unterschieden werden können, ist es nötig eine Möglichkeit zu finden ein Signal im Hinblick auf die Harmonischen, Oberschwingungen einer Grundfrequenz , sowie auf das ADSR Verhalten zu analysieren. Auf dies wird in der Implementation genauer eingegangen.

Bauform der Instrumente

Ein Hauptgrund für das unterschiedliche Klangverhalten von unterschiedlichen Instrumenten ist die Bauform des Instruments. Um die Komplexität hinter der Instrumentenerkennung zu verstehen, soll die Bauform der Trompete genauer betrachtet werden.

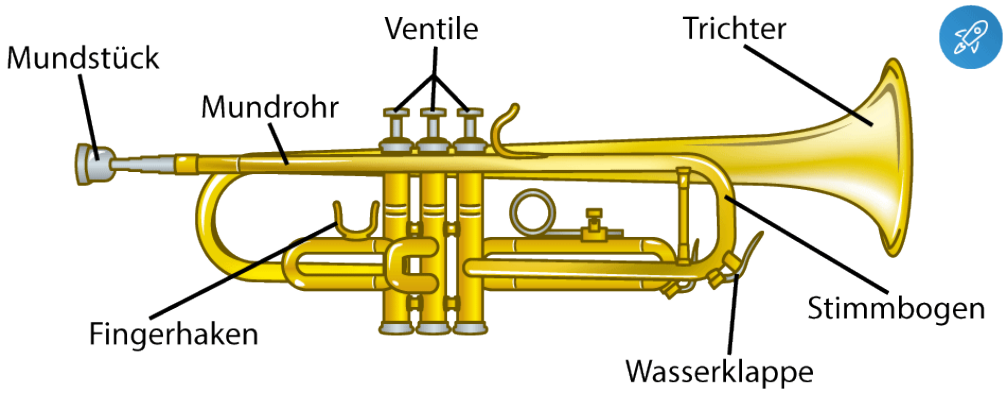


Abbildung 5.8: Aufbau Trompete.[81]

Eine Trompete besteht grundsätzlich aus einem Mundstück (Mouthpiece), einem verbogenen Schallrohr als eigentlichem Körper (Tube) und einem Schallbecher (Bell).

Aus dem Schallbecher treten die für die Trompete charakteristischen Töne aus. Der Körper der Trompete produziert an für sich lediglich ungerade Harmonische Töne. Es ist jedoch erwünscht eine ganze harmonische Sequenz zu erzeugen. Dies wird über das Mundstück und den Schallbecher ermöglicht, welche resonante Frequenzen in Richtung von harmonischen Frequenzen zwingt. Dabei treten zwei Effekte auf.

Zum einen zwingt der Schallbecher die tieferen Resonanzen frequenzmäßig in Richtung höherer Frequenzen. Gleichzeitig eliminiert er dabei die Grundfrequenz und fügt einen Phantomton, auch Pedalton genannt, zum charakteristischen Spektrum hinzu. Dieser Effekt wird generell auch *Bell Effect* genannt. Weiterhin forciert das Mundstück die hohen Resonanzen in tiefere Frequenzbereiche. Dieser Effekt wird auch als *Mouthpiece Effect* bezeichnet.[82]

Ähnliche Effekte treten bei unterschiedlichen Instrumenten auf und machen die automatisierte Transkription von Musik zu einem deutlich komplexeren Unterfangen. Es müssten für jedes Instrument das Klangprofil analysiert und die eigentliche Pipeline integriert werden.

Lautstärken-Erkennung

Der Charakter eines Musikstückes ist oft durch die Dynamik, also die Lautstärkenunterschiede gegeben. Damit das entwickelte System in der Lage ist Lautstärkenunterschiede und Dynamik in einer Melodie zu erkennen, müsste bekannt sein, für welche Lautstärke ein Notenwert als laut beziehungsweise leise gilt.
Die wahrgenommene Lautstärke eines Tones hängt neben dem Schalldruck auch von der Tonhöhe ab [76]. Es kann damit auch für die Lautstärke ein charakteristisches Spektrum aufgestellt werden:

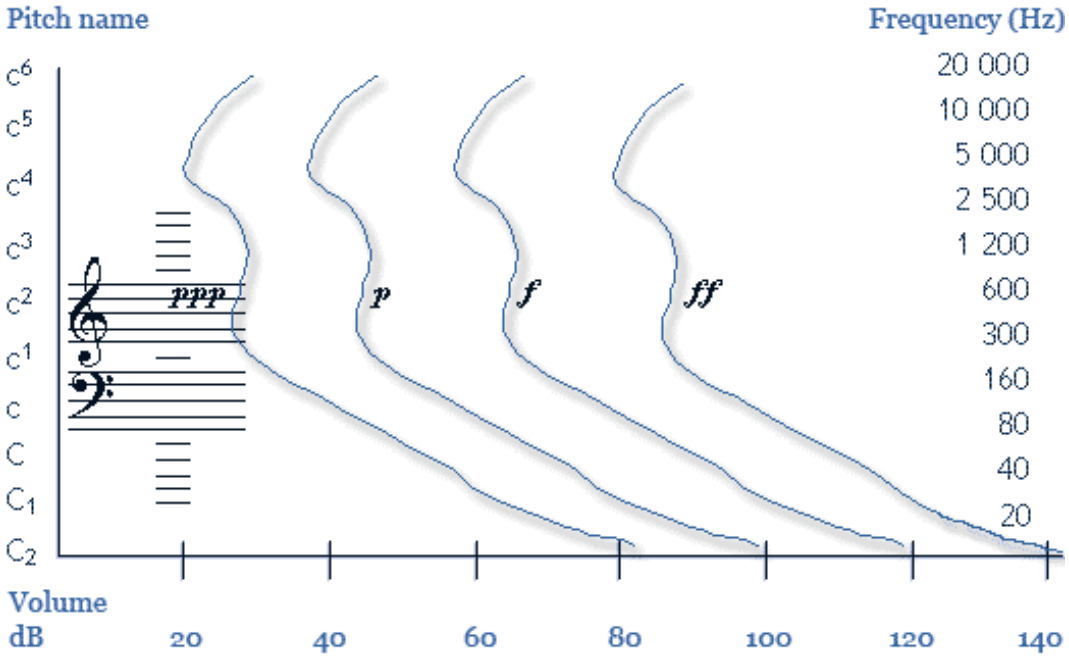


Abbildung 5.9: Lautstärkenprofil.[76]

Abbildung 5.9 zeigt die *Fletcher-Munson* Kurve. Diese beschreibt den Schalldruck über das Frequenzspektrum, wobei die wahrgenommene Lautstärke von gehaltenen Tönen festgehalten wird.[83]
Es zeigt sich, dass für niedrigere Frequenzen eine größere Lautstärke aufgewendet werden muss, um einen Ton zu erzeugen, welcher den gleichen Schalldruck wie ein höherer Ton mit niedrigerer Lautstärke aufweist.

5.6.2 Grenzen der Fourier-Transformation

Sowohl DFT, FFT, wie auch STFT sind Fourier-Transformationen und versuchen damit Signale über Sinus- und Kosinusfunktionen zu approximieren. Diese wiederum sind periodische Signale. Aufgrund dieser Periodizität fehlt es den Funktionen an zeitlicher Lokalität. Bei der STFT wird versucht die Zeitlokalität durch ein sich bewegendes Fenster zu erreichen.

Damit die Zeitauflösung steigt, muss die Fenstergröße verkleinert werden. Das heißt es wird die Samplegröße n verringert. Dies wiederum bedeutet bei gleich bleibender Samplingrate eine Verringerung der Frequenzauflösung, da zunehmend mehr Frequenzen in einer Bin abgebildet werden. Wird das Fenster vergrößert, dann nimmt zwar die Frequenzauflösung zu, allerdings betrachtet man auch ein immer größer werdendes Zeitintervall für jede FFT. Selbst bei einem sich in kleinen Schritten fortbewegendes Fenster um die Zeitgenauigkeit bei guter Frequenzauflösung zu verbessern, kommt man an die Grenzen der Fourieranalyse.

Abrupte Änderungen werden frequenztechnisch allerdings immer noch durch Werte in anderen Bins von einem früheren Zeitpunkt verunreinigt. Diese Dualität wird auch als die Heisenberg'sche Unschärferelation bezeichnet [84]. Es ist damit unmöglich mit dem Fourierverfahren eine Frequenz exakt zu einem bestimmten Zeitpunkt zu bestimmen. Soll die Frequenz genau aufgelöst werden, dann strebt die Fenstergröße gegen unendlich und die Zeitauflösung wird schlecht. Soll die Zeitauflösung exakt sein, dann wird das Fenster lediglich ein Sample enthalten. Es würden sich alle Frequenzen eines Signals zu diesem Zeitpunkt in einer Bin befinden. Damit ist die Frequenzauflösung unendlich ungenau.

Mithilfe der Wavelet-Theorie ist es möglich diese Grenzen zu umgehen. Die detektierten Signale werden in der Wavelet-Theorie mithilfe von *kurzlebigen* Wellen, so genannten Wavelets approximiert, die nur über eine kurze Zeitspanne bestehen bleiben. Damit soll die Zeitauflösung im Vergleich zu der Fouriertransformation verbessert werden.[85]

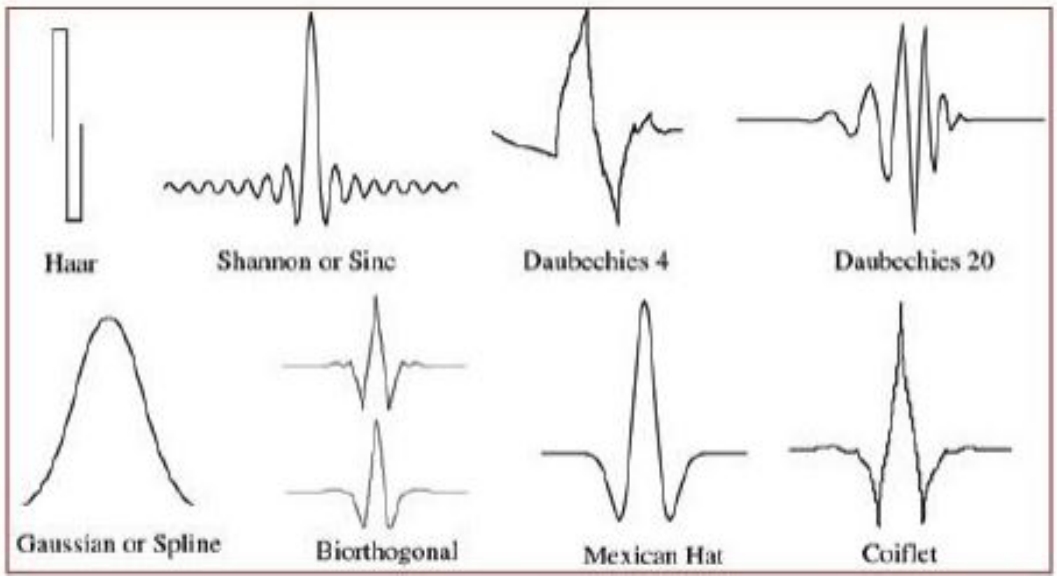


Abbildung 5.10: Wavelets.[85]

In Abbildung 5.10 sind verschiedene Wavelets aufgetragen. Um diese an unterschiedliche Frequenzen anzupassen, werden diese entsprechend zeitlich gestaucht oder gestreckt. Dadurch kann die Wavelet-Theorie gut auf Frequenzänderungen reagieren. Im Prinzip soll ein Mittel gefunden werden, mit dem bei niedrigen Frequenzen eine hohe Frequenzauflösung und bei hohen Frequenzen eine hohe Zeitauflösung erreicht wird.[85]

6 Zusammenfassung und Ausblick

Dieses Kapitel fasst die Erkenntnisse und Entwicklungen der Bachelorarbeit zusammen. Am Schluss wird ein Ausblick angeführt, welcher auf potentielle Erweiterungen der Pipeline für die Zukunft eingeht.

6.1 Zusammenfassung

In der Bachelorarbeit wurde damit ein rudimentäres System zur Transkription von Musik entwickelt. Dabei lag das Hauptaugenmerk auf zwei Gesichtspunkten. Einerseits sollte eine echtzeitfähiges System geschaffen werden. Die entwickelte Pipeline musste damit bestimmte performanzkritische Voraussetzungen erfüllen. Andererseits nimmt das System eine Transkription von Musik vor. Dadurch muss eine bestimmte Qualität der Transkription gewährleistet werden. Im Zuge dieser beiden Gesichtspunkte wurden drei verschiedene Varianten der Pipeline implementiert.

Eine Variante umfasst eine sequentielle Abarbeitung von Audioaufnahme und Audioverarbeitung. Die sequentielle Version stellt eine echtzeitfähige Möglichkeit dar Musik zu transkribieren. Gleichzeitig kommt es allerdings, je nach Wahl von bestimmten performanzkritischen Größen (Samplegröße n, Schrittgröße x), zu erheblichen Verlusten in der Qualität der Transkription.

Die zweite Variante ermöglicht eine verlustfreie Transkription von Musik. Dies wird ermöglicht, indem die gesamte Melodie zunächst in einer Audiodatei abgespeichert wird. Erst nach der Audioaufnahme werden die Audiodaten transkribiert. Bei dieser Version ist die Transkription unabhängig von der Zeit und verlustfrei. Gleichzeitig handelt es sich hierbei jedoch um keine echtzeitfähige Implementierung der Pipeline.

Die dritte Variante versucht eine Mischung aus Qualität der zweiten Variante und der Performanz der ersten Variante zu erreichen. Dabei wird die Audioaufnahme und die Audioverarbeitung parallel in eigenen Threads ausgeführt, wobei die beiden Threads die Audiodaten in einer gemeinsamen Datenstruktur verwalten. Die Audioverarbeitung kann in Echtzeit stattfinden und gleichzeitig wird eine hohe Qualität der Transkription garantiert (Abb. 5.4). Allerdings kann auch für diese Variante die Performanz für kleine Schrittgrößen x und große Samplegrößen x sehr schlecht ausfallen (Abb. 5.5a).

Zur Beurteilung der verschiedenen Varianten in Bezug auf Qualität und Performanz

des Systems wurden automatisierte Tests entwickelt. Weiterhin wurde die Robustheit der Pipeline auf verschiedenen technischen Systemen getestet. (Kap. 5)
Laufzeittechnisch muss sich bei einem normalen technischen Setup kaum Gedanken über die Performanz gemacht werden (Abb. 5.5b). Auf diesen Systemen beschreibt die Pipeline eine echtzeitfähige Lösung. Auf einem minimalen System, wie einem eingebetteten System, kann es allerdings zu Performanz-Engpässen kommen (Abb. 5.5a). Weiterhin beschreibt die Pipeline lediglich eine Basislösung für die Transkription von Musik. Es können diverse Parameter angepasst werden. So muss eine ausgewogene Mischung aus Samplegröße n, Schrittgröße x und der richtigen Fensterfunktion f_{win} bestimmt werden. Die gewählten Werte müssen neben Qualitäts-, auch Performanz-Betrachtungen in die Entscheidung miteinbeziehen.

Es zeigt sich, dass die entwickelten Methoden ein erster Ansatzpunkt für eine Transkription von Musik darstellen. Die Ton- und Akkorderkennung sind für einfrequente Schwingungen relativ robust und liefern akzeptable Ergebnisse (Kap. 5). Die Melodieerkennung könnte allerdings von einer intelligenteren Verarbeitung profitieren. Es fehlt die Möglichkeit Pausen effizient zu detektieren. Weiterhin ist das System lediglich für klare Signale tauglich. Für verschiedene Instrumente liefert eine Transkription nicht garantiert zufriedenstellende Ergebnisse. An dieser Stelle müsste das System auf individuelle Instrumente anpassbar sein und deren Klangprofil in die entwickelten Algorithmen einbezogen werden.

Letztendlich konnte das System seine Anforderungen ausreichend erfüllen und bietet Methoden unterschiedliche Töne, Akkorde und einfache Melodien zu erkennen. Das finale System bietet zudem die Möglichkeit erkannte Melodien als Notenblatt zu generieren. Durch den modularen Aufbau ist einerseits ein relativ robustes, sowie ein skalierbares und adaptives System geschaffen worden, welches auf eigene Bedürfnisse zugeschnitten und erweitert werden kann.

6.2 Ausblick

Die entwickelte Pipeline bietet eine Basisversion für ein Musik-Transkriptions Programm. Durch den modularen Aufbau der Pipeline kann die entwickelte Pipeline beliebig erweitert werden. Es sollen nun auf Punkte eingegangen werden, die noch nicht durch das aktuelle System abgedeckt werden.

Um ein besseres Nutzerverhalten zu ermöglichen, wäre die Entwicklung einer grafischen Oberfläche zu empfehlen. Dabei könnte die Pipeline direkt in die GUI integriert werden. Es wurden bereits erste grafische Elemente an das System angebunden, die vor allem zur Visualisierung der Ergebnisse dienten. Dabei wäre auch zu überlegen, ob möglicherweise die Pipeline in einer höheren Programmiersprache als C implementiert werden soll. Dabei könnten vor allem Sprachen in Betracht gezogen werden, die Frameworks zur Erstellung von grafischen Oberflächen aufweisen. Das finale System verwendet bereits grafische Komponenten, die in Python geschrieben wurden

und das *PyQt5*-Framework [75] verwenden. Python bietet zahlreiche Möglichkeiten zur Audioverarbeitung und kann auf einem Raspberry Pi ausgeführt werden [86]. Falls auf eine interpretierte Sprache wie Python [87] aus Performanzgründen verzichtet werden soll, besteht auch die Möglichkeit eine kompilierte Sprache wie C++ zur Erstellung von grafischen Oberflächen zu verwenden. C++ verfügt über das *Qt*-Framework [88], welches die Grundlage für das *PyQt*-Framework in Python darstellt.

Durch eine bessere Lautstärkenerkennung könnte der Melodieerkennung Dynamik hinzugefügt werden (Abb. 5.9). Es könnten laute von leisen Tönen unterschieden werden. LilyPond bietet hierfür Möglichkeiten Lautstärke in der Zeichenkette zu vermerken. Dadurch können Lautstärken- und Dynamiksymbole dem Notenblatt hinzugefügt werden. Weiterhin würde durch eine verbesserte Lautstärkenerkennung auch die Möglichkeit geschaffen werden, Pausen in einer Melodie zu erkennen und diese auf dem Notenblatt zu notieren. Dabei müsste eine Untergrenze für die Lautstärke definiert werden, ab der ein Audiosignal noch als Ton wahr genommen wird.

Damit die Transkription von Musik auf mehrere Instrumente ausgeweitet werden kann, muss das Klangprofil eines jeden Instrumentes bestimmt und anschließend in die Algorithmen der Pipeline integriert werden. Dabei muss nach 5.6.1 unweigerlich auch die Bauform eines Instrumentes berücksichtigt werden. Das System müsste auf verschiedene Instrumente konfigurierbar sein.

Eine Eigenschaft, die in der aktuellen Version der Pipeline noch gar nicht berücksichtigt wird, ist die Artikulation von Tönen. Damit wird generell verstanden, ob Notenwerte hart oder weich angespielt werden. Hier kann durch die Analyse des Klangprofils das ASDR-Verhalten 5.7 eines Instrumentes bestimmt werden. Unterschiedliche ASDR-Verhalten führen zu einer unterschiedlichen Artikulation eines gespielten Tones.

Es wird deutlich, dass die Komplexität eines solchen Systems die für diese Bachelorarbeit entwickelten Algorithmen bei Weitem übersteigt. Das Klangprofil von Instrumenten zu bestimmen und danach in die Pipeline zu integrieren weist eine hohe Komplexität auf. Zur Lösung solcher Hindernisse könnten intelligentere Verfahren in die Pipeline integriert werden. Dabei handelt es sich um Algorithmen für das maschinelle Lernen oder Neuronale Netze. Weiterhin müssen genug Audiodaten existieren, welche die Aufnahmen von unterschiedlichen Instrumenten für diverse Töne enthalten. Die Algorithmen müssten die Gewichtung von Oberschwingungen und Grundschwingung untereinander identifizieren können und das ASDR-Verhalten der Instrumente erkennen. Durch diese intelligenten Verfahren könnte auch eine Transkription von Musik für mehrere Instrumente aus einem Audiosignal ermöglicht werden.

Literatur

[1] T. T. Reviews, *Best Music Notation Software of 2019*, Zugegriffen am 14. August 2019. Adresse: https://www.toptenreviews.com/best-music-notation-software.

[2] Wikipedia, *Liste von musikalischen Symbolen*, Zugegriffen am: 17. Jun. 2019, Juni 2019. Adresse: https://de.wikipedia.org/wiki/Liste_von_musikalischen_Symbolen.

[3] T. P. Classroom, *Pitch and Frequency*, Zugegriffen am: 24. Jun. 2019. Adresse: https://www.physicsclassroom.com/class/sound/Lesson-2/Pitch-and-Frequency.

[4] Wikipedia, *Auditive Wahrnehmung*, Zugegriffen am 13. August 2019. Adresse: https://de.wikipedia.org/wiki/Auditive_Wahrnehmung.

[5] ——, *Kammerton*, Zugegriffen am 13. August 2019. Adresse: https://de.wikipedia.org/wiki/Kammerton.

[6] *Equations for the Frequency Table*, Zugegriffen am: 24. Jun. 2019. Adresse: https://pages.mtu.edu/~suits/NoteFreqCalcs.html.

[7] sengpielaudio, *Conversion of Intervals*, Zugegriffen am 31. Juli 2019. Adresse: http://www.sengpielaudio.com/calculator-centsratio.html.

[8] UNSW, *What is a decibel?*, Zugegriffen am 13. August 2019. Adresse: http://www.animations.physics.unsw.edu.au/jw/dB.htm.

[9] Wikipedia, *Bel (Einheit)*, Zugegriffen am 13. August 2019. Adresse: https://de.wikipedia.org/wiki/Bel_(Einheit).

[10] U. Essays, *Why Do the Same Notes Sound Different in Instruments?*, Zugegriffen am 13. August 2019. Adresse: https://www.ukessays.com/essays/physics/notes-sound-instruments-8936.php.

[11] Wikipedia, *Overtone*, Zugegriffen am 13. August 2019. Adresse: https://en.wikipedia.org/wiki/Overtone.

[12] ——, *Nyquist-Shannon-Abtasttheorem*, Zugegriffen am 14. August 2019. Adresse: https://de.wikipedia.org/wiki/Nyquist-Shannon-Abtasttheorem.

[13] Academic, *Nyquist-Theorem*, Zugegriffen am 14. August 2019. Adresse: https://deacademic.com/dic.nsf/dewiki/1034731.

[14] E. Weitz, 2018.

[15] texexec, *Fourierreihen*, Zugegriffen am 13. August 2019. Adresse: https://resources.mpi-inf.mpg.de/departments/d1/teaching/ss10/MFI2/kap43.pdf.

[16] U. Wien, *Fourierreihen*, Zugegriffen am 13. August 2019. Adresse: https://www.mathe-online.at/mathint/fourier/i.html.

[17] WhatIs, *Fourier analysis*, Zugegriffen am 13. August 2019. Adresse: https://whatis.techtarget.com/definition/Fourier-analysis.

[18] Wikipedia, *Orthonormalbasis*, Zugegriffen am 14. August 2019. Adresse: https://de.wikipedia.org/wiki/Orthonormalbasis#targetText=Eine%20Orthonormalbasis%20(ONB)%20oder%20ein,H%C3%BClle%20dicht%20im%20Vektorraum%20liegt..

[19] Wikiversity, *Vektorraum/K/Skalarprodukt/Orthogonale Projektion/Einführung/Textabschnitt*, Zugegriffen am 13. August 2019. Adresse: https://de.wikiversity.org/wiki/Vektorraum/K/Skalarprodukt/Orthogonale_Projektion/Einf%C3%BChrung/Textabschnitt.

[20] U. Freiburg, *Orthogonale Funktionsräume*, Zugegriffen am 14. August 2019. Adresse: https://www.qc.uni-freiburg.de/files/orthogonale_funktionenraeume_und_der_laplaceoperator.pdf.

[21] Wikipedia, *Trigonometrisches Polynom*, Zugegriffen am 13. August 2019. Adresse: https://de.wikipedia.org/wiki/Trigonometrisches_Polynom.

[22] S. Wörner, *Fast Fourier Transform*, Zugegriffen am 13. August 2019. Adresse: http://pages.di.unipi.it/gemignani/woerner.pdf.

[23] Wikipedia, *Eulersche Formel*, Zugegriffen am 13. August 2019. Adresse: https://de.wikipedia.org/wiki/Eulersche_Formel.

[24] H. Verlag, *Einheitswurzeln und Kreisteilungspolynome*, Zugegriffen am 13. August 2019. Adresse: http://www.heldermann-verlag.de/Mathe-Kabinett/Kreisteilungspolynome.pdf.

[25] L. slides by Kevin Wayne, *Divide and Conquer II*, Zugegriffen am 13. August 2019. Adresse: https://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/05DivideAndConquerII.pdf.

[26] Wikipedia, *Cooley-Tukey FFT algorithm*, Zugegriffen am 14. August 2019. Adresse: https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm.

[27] R. Lyons, *Computing FFT Twiddle Factors*, Zugegriffen am 13. August 2019. Adresse: https://www.dsprelated.com/showcode/232.php.

[28] T. H. Cormen, C. E. Leiserson, R. L. Rivest und C. Stein, 2017.

[29] Wikipedia, *Bluestein-FFT-Algorithmus*, Zugegriffen am 14. August 2019. Adresse: https://de.wikipedia.org/wiki/Bluestein-FFT-Algorithmus.

[30] P. Nayuki, *Free small FFT in multiple languages*, Zugegriffen am 14. August 2019. Adresse: https://www.nayuki.io/page/free-small-fft-in-multiple-languages.

[31] G. Waves, *How to interpret FFT results – obtaining magnitude and phase information*, Zugegriffen am 14. August 2019. Adresse: https://www.gaussianwaves.com/2015/11/interpreting-fft-results-obtaining-magnitude-and-phase-information/.

[32] M. Gasior und J. Gonzalez, *IMPROVING FFT FREQUENCY MEASUREMENT RESOLUTION BY PARABOLIC AND GAUSSIAN INTERPOLATION*, Feb. 2004.

[33] N. Instruments, *Schnelle Fourier-Transformation (FFT) und Fensterfunktion*, Zugegriffen am 31. Juli 2019. Adresse: https://www.ni.com/de-de/innovations/white-papers/06/understanding-ffts-and-windowing.html.

[34] Wikipedia, *Window Function*, Zugegriffen am 13. August 2019. Adresse: https://en.wikipedia.org/wiki/Window_function.

[35] S. Direct, *Short Time Fourier Transform*, Zugegriffen am 14. August 2019. Adresse: https://www.sciencedirect.com/topics/engineering/short-time-fourier-transform.

[36] Wikipedia, *Short Time Fourier Transform*, Zugegriffen am 14. August 2019. Adresse: https://en.wikipedia.org/wiki/Short-time_Fourier_transform#targetText=From%20Wikipedia%2C%20the%20free%20encyclopedia,as%20it%20changes%20over%20time..

[37] ——, *Pulse-Code-Modulation*, Zugegriffen am: 17. Jun. 2019, März 2019. Adresse: https://de.wikipedia.org/wiki/Puls-Code-Modulation.

[38] A. Jacobi, *Klang und kurzen Ladezeiten [Podcast]*, Zugegriffen am: 24. Jun. 2019. Adresse: https://www.sprechersprecher.de/blog/wav-vs-mp3-ladezeiten-und-audioqualitaet.

[39] Truelogic, *Parsing a WAV file in C*, Zugegriffen am: 17. Jun. 2019, Sep. 2015. Adresse: http://truelogic.org/wordpress/2015/09/04/parsing-a-wav-file-in-c/.

[40] C. S. Sapp, *WAV PCM soundfile format*, Zugegriffen am: 17. Jun. 2019, Dez. 1997. Adresse: http://soundfile.sapp.org/doc/WaveFormat/.

[41] ——, *RIFF WAVE*, Zugegriffen am: 17. Jun. 2019, Juli 2019. Adresse: https://de.wikipedia.org/wiki/RIFF_WAVE.

[42] unknown, *Digital Audio - Creating a WAV (RIFF) file*, Zugegriffen am: 17. Jun. 2019. Adresse: http://www.topherlee.com/software/pcm-tut-wavformat.html.

[43] Wikipedia, *Resource Interchange File Format*, Zugegriffen am: 17. Jun. 2019, Jan. 2018. Adresse: https://de.wikipedia.org/wiki/Resource_Interchange_File_Format.

[44] D. O. M. Guide, *MIDI Guide*, Zugegriffen am: 17. Jun. 2019, Mai 2019. Adresse: http://unseretollepage.de/hosted/midiguide/kap01.html#einseins.

[45] MuseScore, *MIDI Import*, Zugegriffen am 14. August 2019. Adresse: https://musescore.org/de/handbook/midi-import.

[46] L. Cons, K. Berry, O. Bachmann u. a., *LilyPond*, Zugegriffen am: 17. Jun. 2019, Juni 2019. Adresse: http://lilypond.org/index.de.html.

[47] Lilypond, *Handbücher für LilyPond 2.18.2*, Zugegriffen am 10. Juli 2019. Adresse: `http://lilypond.org/manuals.de.html`.

[48] L. development team, *Learning Manual*, Zugegriffen am 03. August 2019. Adresse: `http://lilypond.org/doc/v2.19/Documentation/learning.pdf`.

[49] ABRSM, *The statistics: part 1*, Zugegriffen am: 29. Jun. 2019. Adresse: `https://gb.abrsm.org/en/making-music/4-the-statistics`.

[50] Wikipedia, *Digitale Signalverarbeitung*, Zugegriffen am: 29. Jun. 2019. Adresse: `https://de.wikipedia.org/wiki/Digitale_Signalverarbeitung`.

[51] seewave, *A very short introduction to sound analysis for those who like elephant trumpet calls or other wildlife sound*, Zugegriffen am 14. August 2019. Adresse: `https://cran.r-project.org/web/packages/seewave/vignettes/seewave_analysis.pdf`.

[52] L. Lu, H.-J. Zhang und H. Jiang, *Content Analysis for Audio Classification and Segmentation*, Okt. 2002.

[53] H. Purwins, B. Li, T. Virtanen, J. Schlüter, S.-y. Chang und T. Sainath, *Deep Learning for Audio Signal Processing*, Mai 2019.

[54] MusicTech, *Six of the best audio analysis tools*, Zugegriffen am 14. August 2019. Adresse: `https://www.musictech.net/guides/buyers-guide/6-best-audio-analysis-tools/`.

[55] Audacity, *Audacity*, Zugegriffen am 14. August 2019. Adresse: `https://www.audacity.de/`.

[56] TechInMusicEd, *NotateMe and PhotoScore Updates*, Zugegriffen am 14. August 2019. Adresse: `https://techinmusiced.wordpress.com/2017/03/21/notateme-and-photoscore-updates/`.

[57] M. Scanner, *Melody Scanner*, Zugegriffen am 14. August 2019. Adresse: `https://melodyscanner.com/`.

[58] Songs2See, *Songs2See Editor*, Zugegriffen am 14. August 2019. Adresse: `https://www.songs2see.com/de/songs2see_produkte/editor/`.

[59] AudioScore, *AudioScore Ultimate*, Zugegriffen am 14. August 2019. Adresse: `https://www.avid.com/de/products/audioscore-ultimate`.

[60] N. Software, *TwelveKeys Music Transcription Software*, Zugegriffen am 14. August 2019. Adresse: `https://www.nch.com.au/twelvekeys/index.html`.

[61] seventhstring, *Transcribe!*, Zugegriffen am 14. August 2019. Adresse: `https://www.seventhstring.com/xscribe/overview.html`.

[62] AnthemScore, *AnthemScore*, Zugegriffen am 14. August 2019. Adresse: `https://www.lunaverus.com/`.

[63] E. Benetos, S. Dixon, D. Giannoulis, H. Kirchhoff und A. Klapuri, „Automatic music transcription: Challenges and future directions“, *Journal of Intelligent Information Systems*, Jg. 41, Dez. 2013. DOI: `10.1007/s10844-013-0258-3`.

[64] U. Users, *ALSA*, Zugegriffen am 01. August 2019. Adresse: `https://wiki.ubuntuusers.de/ALSA/`.

[65] A. Inan, *quick and dirty spectrum analyzer*, Zugegriffen am 14. August 2019. Adresse: https://github.com/xdsopl/spectrum.

[66] Nabble, *Instructions to install lilypond onto Raspberry Pi*, Zugegriffen am 14. August 2019. Adresse: http://lilypond.1069038.n5.nabble.com/Instructions-to-install-lilypond-onto-Raspberry-Pi-td216154.html.

[67] Github, *Hacklily*, Zugegriffen am 14. August 2019. Adresse: https://github.com/hacklily/hacklily.

[68] Wikipedia, *Frescobaldi (Software)*, Zugegriffen am 14. August 2019. Adresse: https://de.wikipedia.org/wiki/Frescobaldi_(Software).

[69] ——, *POSIX Threads*, Zugegriffen am 06. August 2019. Adresse: https://en.wikipedia.org/wiki/POSIX_Threads.

[70] ——, *Lock (computer science)*, Zugegriffen am 06. August 2019. Adresse: https://en.wikipedia.org/wiki/Lock_(computer_science).

[71] H. Hearing, *Understanding high-frequency hearing loss*, Zugegriffen am 14. August 2019. Adresse: https://www.healthyhearing.com/report/52448-Understanding-high-frequency-hearing-loss.

[72] Wikipedia, *Equal-loudness Contour*, Zugegriffen am 14. August 2019. Adresse: https://en.wikipedia.org/wiki/Equal-loudness_contour.

[73] Zytrax, *Frequency Ranges*, Zugegriffen am: 24. Jun. 2019. Adresse: http://www.zytrax.com/tech/audio/audio.html.

[74] LilyPond, *MIDI-Instrumente*, Zugegriffen am 13. August 2019. Adresse: http://lilypond.org/doc/v2.19/Documentation/notation/midi-instruments.

[75] R. Computing, *What is PyQt?*, Zugegriffen am 14. August 2019. Adresse: https://www.riverbankcomputing.com/software/pyqt/intro.

[76] T. B. of Acoustics, *Hertz, cent and decibel*, Zugegriffen am 13. August 2019. Adresse: http://www2.siba.fi/akustiikka/?id=38&la=en.

[77] A. Erlangen, *Timbre*, Zugegriffen am: 24. Jun. 2019. Adresse: https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S3_Timbre.html.

[78] Zytrax, *Timbre*, Zugegriffen am: 24. Jun. 2019. Adresse: http://www.zytrax.com/tech/audio/sound.html.

[79] Wikipedia, *ASDR*, Zugegriffen am 14. August 2019. Adresse: https://de.wikipedia.org/wiki/ADSR.

[80] Zytrax, *Tech Stuff - Sound Primer*, Zugegriffen am 14. August 2019. Adresse: http://www.zytrax.com/tech/audio/sound.html.

[81] Einsteiger.org, *Alles rund um die Trompete*, Zugegriffen am 14. August 2019. Adresse: https://www.einsteiger.org/trompete/.

[82] Hyperphysics, *Producing a harmonic sequence of notes with a trumpet*. Adresse: http://hyperphysics.phy-astr.gsu.edu/hbase/Music/brassa.html#c1.

[83] Wikipedia, *Equal-Loudness Contour*, Zugegriffen am 14. August 2019. Adresse: https://en.wikipedia.org/wiki/Equal-loudness_contour.

[84] ——, *Wavelet*, Zugegriffen am 14. August 2019. Adresse: https://de.wikipedia.org/wiki/Wavelet.

[85] G. Dallas, *Wavelets 4 Dummies: Signal Processing, Fourier Transforms and Heisenberg*, Zugegriffen am 14. August 2019. Adresse: https://georgemdallas.wordpress.com/2014/05/14/wavelets-4-dummies-signal-processing-fourier-transforms-and-heisenberg/.

[86] M. Portal, *Audio Processing in Python Part I: Sampling, Nyquist, and the Fast Fourier Transform*, Zugegriffen am 14. August 2019. Adresse: https://makersportal.com/blog/2018/9/13/audio-processing-in-python-part-i-sampling-and-the-fast-fourier-transform.

[87] N. Batchelder, *Is Python interpreted or compiled? Yes.* Zugegriffen am 14. August 2019. Adresse: https://nedbatchelder.com/blog/201803/is_python_interpreted_or_compiled_yes.html.

[88] Qt, *Qt for Beginners*, Zugegriffen am 14. August 2019. Adresse: https://wiki.qt.io/Qt_for_Beginners.

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich diese schriftliche Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe.

Ort, Datum Unterschrift