

Time Scale Modification

Nachimplementation bekannter Verfahren zur
Time-Scale Modifikation in JavaScript und
deren Auswertung

Roman Käslin

Hochschule Luzern - Technik und Architektur

Dozent: Prof. Dr. Thomas Hunziker

Experte: Prof. Dr. Eng. Armin Taghipour

31. Mai 2024

*Eine Arbeit für das Modul PAIND im Bachelorstudium
Elektrotechnik und Informationstechnologie*

Zusammenfassung

In dieser Arbeit wurden verschiedene bekannte Verfahren zur Time-Scale Modification untersucht. Diese Verfahren wurden in JavaScript implementiert, auf einer Weboberfläche dargestellt und anschliessend ausgewertet. Es wurde gezeigt, warum diese Verfahren gerechtfertigt sind, wie sie funktionieren und welche Entscheidungen bei der Implementierung getroffen wurden. Abschliessend wurde analysiert, wie sich die verschiedenen Methoden auf das Audiosignal auswirken und wie gut sie die Tonhöhe und die Zeitstruktur bewahren. Da keine dieser Verfahren eine perfekte Time-Scale Modification ohne hörbare Artefakte ermöglicht, wurden diese Artefakte aufgezeigt und analysiert. Dadurch konnte ermittelt werden, welche Methode für welchen Anwendungsfall am besten geeignet ist, welche Artefakte sie erzeugt und durch welche Tricks sich diese optimieren lassen.

Keywords— Audio, Signal Processing, JavaScript, Web

Inhaltsverzeichnis

1 Einleitung	1
2 Problemstellung	2
2.1 Upsampling	2
2.2 Downsampling	4
2.3 Aufgabenstellung	4
3 Methodik	6
3.1 Tools	6
3.2 Analyse	7
4 Time-Scale Modification	8
4.1 Overlap-Add (OLA)	8
4.2 Waveform Similarity Overlap-Add (WSOLA)	13
4.3 Phase Vocoder (PV)	14
4.4 Phase Vocoder mit Phase Locking (PVPL)	17
4.5 Harmonic-Percussive Source Separation (HPSS)	18
5 Implementation	19
5.1 File Handling	19
5.2 Signalverarbeitung	21
5.3 User Interface	22
5.4 Probleme	23
6 Auswertung	26
6.1 Perfect Reconstruction	26
6.2 Artefakte	28
6.3 Ausblick	31
6.4 Rückblick	32

Abbildungsverzeichnis

1	Abgetasteter Sinus mit einer Frequenz von 440 Hz über zwei Perioden	2
2	Gestreckter Sinus durch Zero Padding	3
3	Gestreckter Sinus mit Interpolation	4
4	Gestauchter Sinus durch Downsampling	5
5	Aufteilung des Signals in Frames	9
6	Überlagerung der Frames	10
7	Artefakte bei der Synthese von Frames	11
8	Hanning Fenster angewendet auf einen Frame	12
9	Überlagerung der Frames nach Anwendung der Window Funktion	12
10	Addition der Frames nach Anwendung der Window Funktion	13
11	Spectrogram nach der STFT	16
12	Datenflussdiagramm der Implementierung	19
13	Visualisierung der Web Audio API Nodes	23
14	Visualisierung der Perfect Reconstruction für OLA	27
15	Visualisierung der Perfect Reconstruction für PV	27
16	Gestrecktes Referenzsignal mit OLA	29
17	Verkürztes Referenzsignal mit OLA	30
18	Gestrecktes Referenzsignal mit PV	30
19	Gestrecktes Referenzsignal mit PV	31

1 Einleitung

Es gibt zahlreiche praktische Anwendungsfälle, in denen es erwünscht ist, die Wiedergabegeschwindigkeit eines Audioträcks anzupassen, sei es durch Verlangsamten oder Beschleunigen. Ein Beispiel hierfür wäre die Synchronisierung eines Videos in Zeitlupe mit der zugrunde liegenden Audio. Auf den ersten Blick scheint die Lösung einfach zu sein. Durch Interpolation oder Dezimierung der Samples sollte es möglich sein, den gewünschten Effekt zu erzielen. Bis zu einem gewissen Grad ist die Annahme auch korrekt. Allerdings liegt das Problem darin, dass dabei ein bekanntes Phänomen auftritt. Obwohl das Signal in der Zeitdomäne verändert wird, hat dies auch Auswirkungen auf die Frequenzen. Genauer gesagt verändern sich die Tonhöhe (Pitch) und die Klangfarbe (Timbre) des Signals. Das bedeutet, dass ein gestrecktes oder gestauchtes Signal nicht nur schneller oder langsamer klingt, sondern auch seine charakteristischen klanglichen Eigenschaften verändert werden.

Um mit diesem Phänomen umzugehen, gibt es verschiedene etablierte Verfahren, die unter dem Begriff Time-Scale Modification (TSM) zusammengefasst werden. Für die konkrete Implementierung gibt es verschiedene Ansätze, die in zwei Hauptkategorien unterteilt werden können: den Zeitbereich und den Frequenzbereich. Innerhalb dieser Kategorien gibt es unterschiedliche Implementationsmöglichkeiten. In dieser Arbeit werden folgende Verfahren beschrieben:

- Overlap-Add (OLA)
- Waveform Similarity Overlap-Add (WSOLA)
- Phase Vocoder (PV)
- Phase Vocoder mit Phase Locking (PVPL)
- Harmonic-Percussive Source Separation (HPSS)

Im Rahmen dieser Arbeit wurden in Kapitel 2 die genannten Probleme beschrieben und mit Beispielen demonstriert. Anschliessend wurden die oben genannten Verfahren in Kapitel 4 detailliert erläutert, wobei der Schwerpunkt auf OLA und PV liegt, da diese in JavaScript implementiert und auf einer Webseite präsentiert werden. Dies wird in Kapitel 5 behandelt. Abschliessend wird in Kapitel 6 die Implementierung analysiert und diskutiert, um Artefakte und mögliche Verbesserungen aufzuzeigen.

2 Problemstellung

Bevor man sich mit der Time-Scale Modification beschäftigt, muss zuerst verstanden werden, weshalb einfache Verfahren zur Streckung oder Stauchung eines Audiosignals nicht ausreichen. Bei Verfahren wie dem Upsampling oder Down-sampling werden die Frequenzen verändert, was zur Folge hat, dass sich auch der Klang des Signals ändert. Beispielsweise ändert sich bei Wiedergabe mit doppelter Geschwindigkeit die Tonhöhe um eine Oktave. Warum dies genau so ist, wird in den folgenden Abschnitten näher erläutert. Bei den folgenden Abbildungen handelt es sich um vereinfachte künstliche Darstellungen, um die Problematik zu verdeutlichen und stehen nicht im Zusammenhang mit den in Kapitel 5 beschriebenen Verfahren.

Als Beispiel wird ein Sinuston mit einer Frequenz von 440 Hz verwendet, was einem A4-Ton entspricht [12]. In Abbildung 1 ist das Beispielsignal über zwei Perioden dargestellt. An diesem Signal werden nun einfache Verfahren zur Streckung und Stauchung angewendet, um die dabei auftretenden Probleme zu verdeutlichen.

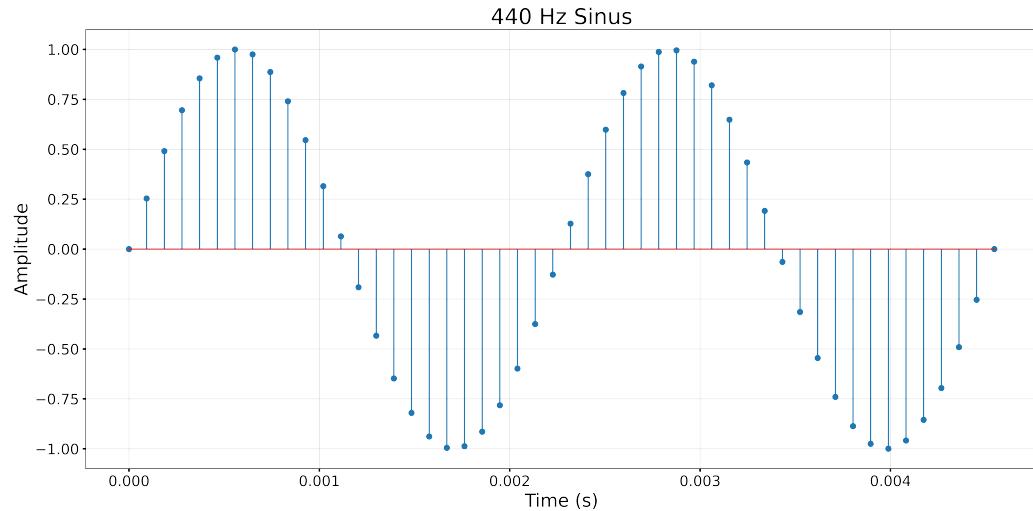


Abbildung 1: Abgetasteter Sinus mit einer Frequenz von 440 Hz über zwei Perioden

2.1 Upsampling

Als erstes wird ein einfaches Upsampling-Verfahren betrachtet. Dabei soll das Signal um den Faktor zwei gestreckt werden. Um dies zu erreichen, gibt es zwei

2 PROBLEMSTELLUNG

Möglichkeiten. Zum einen kann das Signal durch das Einfügen von Nullen gestreckt werden. Zum anderen kann das Signal durch Interpolation gestreckt werden.

Zero Padding: Nehmen wir einmal an das wir ein Signal strecken wollen. Um ein Audiosignal zu strecken, kann es upgesampelt werden und anschliessend mit einem Tiefpassfilter gefiltert werden. Das Upsampling kann mit verschiedenen Methoden durchgeführt werden. Eine Möglichkeit ist das Einfügen von Nullen zwischen den Samples. In Abbildung 2 lässt sich nun erkennen das das Signal tatsächlich um die gewünschte Länge gestreckt wurde. Allerdings sieht man auch ganz klar das sich die Periode des Signals verdoppelt hat. Das bedeutet das dabei die Frequenz halbiert wurde und somit mit 220 Herz die Tonhöhe um eine Oktave tiefer ist [4].

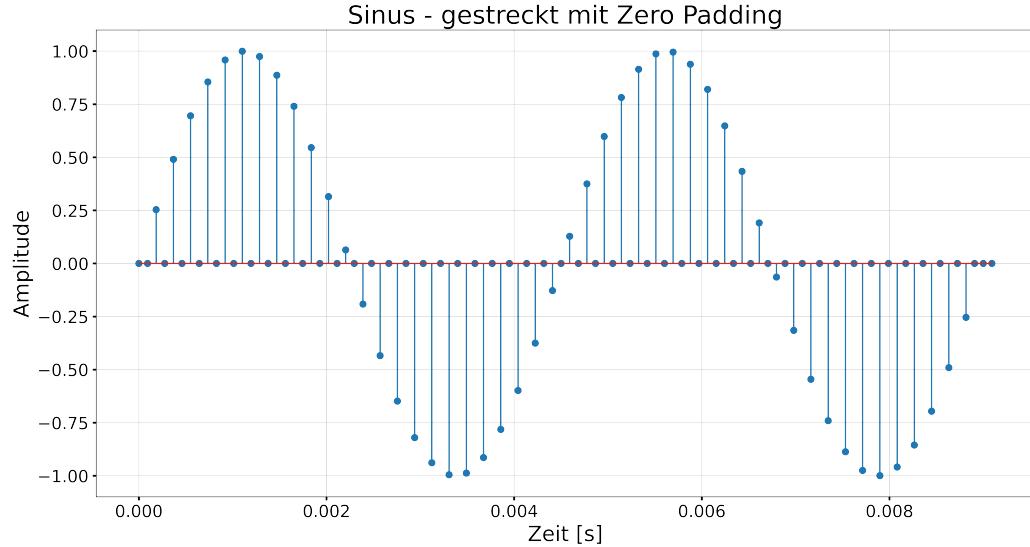


Abbildung 2: Gestreckter Sinus durch Zero Padding

Interpolation: Eine Alternative zum Zero Padding ist die Interpolation der Samples. Dabei werden neue Samples zwischen den vorhandenen generiert. Die Werte dieser neuen Samples werden anhand des Abstands zu den benachbarten Originalsamples berechnet. In Abbildung 3 ist zu erkennen, dass das Signal auch durch Interpolation gestreckt wurde. Dabei verdoppelt sich jedoch die Periode des Signals, was zur Halbierung der Frequenz führt. Dies bedeutet, dass die Tonhöhe um eine Oktave tiefer wird. Dieses Verfahren wird jedoch häufiger zur Visualisierung von Audiosignalen verwendet [17].

2 PROBLEMSTELLUNG

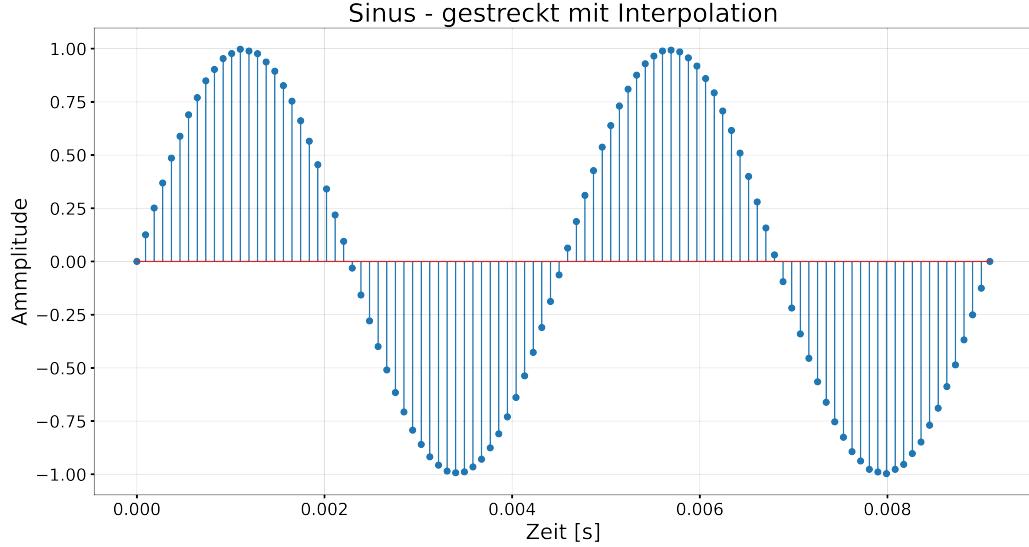


Abbildung 3: Gestreckter Sinus mit Interpolation

Bei beiden Methoden entsteht also das gleiche Problem. Bevor potentielle Lösungen betrachtet werden, sollen im nächsten Abschnitt zuerst die Probleme beim Downsampling untersucht werden.

2.2 Downsampling

Soll das Signal verkürzt beziehungsweise gestaucht werden, können Samples entfernt werden. Abbildung 4 zeigt das heruntergesampelte Signal. In diesem Beispiel wurde jedes zweite Sample entfernt. Das Signal hat dadurch eine Frequenz von 880 Hz, was bedeutet, dass die Tonhöhe um eine Oktave höher ist. Auch beim Downsampling entsteht das Problem, dass sich die Frequenzen des Signals verändern. Das Signal wird höher [4].

2.3 Aufgabenstellung

Wie sich an dem vorherigen Beispiel erkennen lässt, ist es nicht möglich, ein Audiosignal auf eine einfache Weise zu strecken oder zu stauchen, ohne dass sich die Frequenz verändert. Basierend auf dieser Problemstellung sollen in dieser Arbeit verschiedene Methoden zur Time-Scale Modification implementiert, untersucht und verglichen werden. Dabei soll aufgezeigt werden, wie sich die verschiedenen Methoden auf das Audiosignal auswirken und wie gut sie die Tonhöhe (Pitch) und

2 PROBLEMSTELLUNG

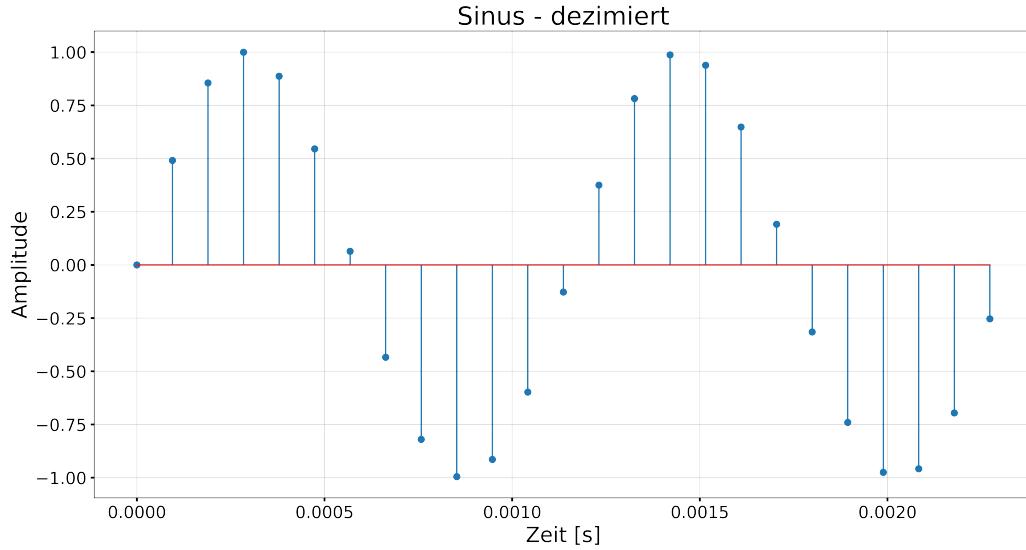


Abbildung 4: Gestrauchter Sinus durch Downsampling

die Zeit (Time) erhalten.

Bei der Implementierung handelt es sich um eine Webapplikation, die es ermöglicht, Audiosignale zu laden und diese mit verschiedenen Methoden zu strecken oder zu stauchen. Der Fokus liegt dabei auf der Overlap-Add-Methode und dem Phase Vocoder. Diese beiden Methoden werden in nativem JavaScript implementiert und auf einer Webseite präsentiert.

Da keiner dieser Verfahren eine perfekte Time-Scale Modification ermöglicht, ohne hörbare Artefakte zu erzeugen, sollen diese Artefakte identifiziert und analysiert werden. Dadurch soll aufgezeigt werden, welche Methode für welchen Anwendungsfall am besten geeignet ist, welche Artefakte bei den verschiedenen Verfahren entstehen und wie sich diese durch bestimmte Techniken optimieren lassen.

3 Methodik

Bevor die verschiedenen Lösungsansätze zur Time-Scale Modification beschrieben werden, wird zunächst die Methodik ausführlich erläutert. Zuerst wird dargelegt, welche Tools und Softwarelösungen für die Entwicklung der Algorithmen, die Analyse der Ergebnisse und die Dokumentation des Projekts verwendet wurden. Dazu gehören Entwicklungsumgebungen, Programmiersprachen und spezifische Bibliotheken. Anschliessend wird detailliert erklärt, welche Methoden und Techniken angewendet werden, um die korrekte Verarbeitung der Audiosignale zu gewährleisten. Dies umfasst die Validierung der implementierten Algorithmen sowie die Verfahren zur Sicherstellung der Signalintegrität während der Modifikationsprozesse.

3.1 Tools

In diesem Kapitel werden die verwendeten Tools beschrieben, die im Verlauf dieses Projekts sowohl für die Implementierung der Algorithmen als auch für die Analyse der Ergebnisse eingesetzt wurden.

Python: Für die Analyse und Erzeugung der Bilder, die zur Dokumentation der Arbeit verwendet wurden, kam Python zum Einsatz. Python ist eine Programmiersprache, die sich durch ihre einfache Syntax und die grosse Anzahl an Bibliotheken auszeichnet. Insbesondere wurde für diese Arbeit die Bibliothek Matplotlib verwendet, mit der sich einfach Graphen und Bilder erzeugen lassen. Für die Verarbeitung der generierten Audiodateien erwies sich SciPy als hervorragend geeignet. SciPy ist eine Bibliothek, die sich auf die Verarbeitung wissenschaftlicher Daten spezialisiert hat und beispielsweise die Erzeugung von Spektrogrammen ermöglicht.

ChatGPT: Kaum ein Tool ist so vielseitig wie ChatGPT. ChatGPT erwies sich als äusserst nützlich, um Informationen aus verschiedenen Perspektiven zu erklären oder zu übersetzen. Auch beim Debugging leistete ChatGPT wertvolle Unterstützung, indem es Fehlermeldungen in verständliche Sprache übersetzte oder Implementierungsfehler schnell identifizierte. Es sei jedoch darauf hingewiesen, dass ChatGPT nicht immer die richtige Antwort liefert, daher ist es wichtig, die Antworten zu überprüfen. Auch beim Ausarbeiten von Texten konnte auf KI zurückgegriffen werden, beispielsweise um Sätze umzuformulieren oder die Rechtschreibung zu überprüfen.

Google Chrome: Google Chrome ist ein Webbrowser, der sich durch seine hohe Geschwindigkeit und Benutzerfreundlichkeit auszeichnet. Für die Implementierung der Algorithmen in JavaScript und die Erstellung der Webseite war Goo-

3 METHODIK

gle Chrome das bevorzugte Tool. Durch die Entwicklerkonsole von Google Chrome konnten Fehler schnell identifiziert und behoben werden. Da während des gesamten Implementierungsprozesses Google Chrome für die Ausführung und das Debugging der Webseite verwendet wurde, besteht die Möglichkeit, dass in anderen Browsern nicht alle Funktionen korrekt funktionieren.

3.2 Analyse

In diesem Kapitel wird beschrieben, wie die Ergebnisse analysiert wurden. Es werden verschiedene Methoden erläutert und gezeigt, wie Artefakte entstehen und wie diese minimiert werden können.

Referenzsignal: Zur Analyse der Ergebnisse der verschiedenen Methoden wurde ein 440 Hz Sinuston als Referenzsignal verwendet. Dieses Signal diente dazu, die Ergebnisse der verschiedenen Methoden zu visualisieren und einen klaren Vergleich zu ermöglichen. Durch die Verwendung eines einfachen Signals ohne Störungen oder Artefakte konnte sichergestellt werden, dass etwaige Abweichungen oder Veränderungen in den Ergebnissen direkt auf die angewandten Methoden zurückzuführen sind. Somit bot das Referenzsignal eine solide Basis für eine präzise Analyse der Leistungsfähigkeit und Genauigkeit der untersuchten Verfahren.

Audacity: Für die Erzeugung des Referenzsignals wurde die Software Audacity verwendet. Audacity ist ein Open-Source-Programm zur Bearbeitung von Audiodateien, das es ermöglicht, Audiodateien aufzunehmen, zu bearbeiten und abzuspielen.

Perfect Reconstruction: Ein wichtiges Konzept in der Audioverarbeitung ist die Perfect Reconstruction. Das bedeutet, dass das Signal nach der Verarbeitung wieder genau so aussieht wie vor der Verarbeitung, sodass Amplitude und Frequenz des Signals erhalten bleiben. Das Perfect Reconstruction ist von zentraler Bedeutung in der Audioverarbeitung, da es sicherstellt, dass das Signal nicht verfälscht wird und somit die Qualität und Integrität des Signals gewährleistet bleibt. Es ist ein entscheidendes Kriterium für die Effektivität und Zuverlässigkeit von Audioverarbeitungsalgorithmen und -systemen [3].

4 Time-Scale Modification

Wie im vorherigen Kapitel angesprochen, werden nun die verschiedenen Verfahren betrachtet, um die genannten Ziele zu erreichen. Noch einmal kurz zusammengefasst: Time-Scale Modification ist ein Verfahren in der digitalen Signalverarbeitung, bei dem Audiosignale gestreckt oder gestaucht werden. Idealerweise hat dies keine Auswirkung auf die Tonhöhe (Pitch) oder die Klangfarbe (Timbre). Da dies in praktischen Anwendungen nahezu unmöglich ist, gilt es, die auftretenden Verzerrungen möglichst in einem nicht merkbaren Bereich zu halten. Ein möglicher Anwendungsbereich ist die Bearbeitung von Audio in der Musik- und Filmindustrie. Overlap-Add und der Phase Vocoder werden im Detail betrachtet.

Für die konkrete Implementierung einer Time-Scale Modification lässt sich das Prinzip in zwei grundlegende Konzepte unterteilen: die Verarbeitung im Zeitbereich und die Verarbeitung im Frequenzbereich. Das generelle Vorgehen ist dabei sehr ähnlich.

4.1 Overlap-Add (OLA)

Begonnen wird mit dem Overlap-Add-Verfahren, da es eine essentielle Technik in der digitalen Signalverarbeitung ist, die verwendet wird, um lange Signale effizient zu filtern und zu transformieren. Genau diese Anwendung wird in der Time-Scale Modification genutzt. Das Signal wird in überlappende Segmente zerlegt, die dann individuell verarbeitet werden. Anschliessend werden die bearbeiteten Segmente wieder zusammengesetzt. Auf diese Weise wird eine hohe Qualität bei der Modifikation der Wiedergabedauer erreicht. Zudem bildet das Overlap-Add-Verfahren die Basis für alle weiteren hier behandelten Methoden. Wie das genau funktioniert, wird in den folgenden Abschnitten genauer erläutert.

Frames: Der erste Schritt beim Overlap-Add-Verfahren ist die Unterteilung des Audiosignals in gleich grosse Frames. Die Länge der Frames sollte dabei nicht willkürlich gewählt werden, sondern hängt vom Audiosignal ab. Die Länge der Frames bestimmt die im Frame auftretenden lokalen Frequenzen. Ist der Frame zu klein, können die Frequenzen möglicherweise nicht korrekt erkannt werden. Ist der Frame zu gross, können die Frequenzen nicht mehr lokalisiert werden. Zudem beeinflusst die Frame-Länge die Auflösung im Frequenzbereich. Diese Faktoren werden im späteren Kapitel 4.3 über den Phase Vocoder noch genauer betrachtet.

Gleichung (1) zeigt wie ein Frame generiert wird. Dabei ist H_a die Analyse HOP-Size und m der Index des Frames. der Index r entspricht dabei dem Index

4 TIME-SCALE MODIFICATION

des Samples. Die Hop-Size wird im nächsten Abschnitt genauer erläutert.

$$x_m(r) = \begin{cases} x(r + mH_a), & \text{wenn } r \in [-N/2 : N/2 - 1], \\ 0, & \text{sonst.} \end{cases} \quad (1)$$

Analyse: Bevor die Frames erzeugt werden können, muss zuerst eine Analyse-HOP-Size bestimmt werden. Die Analyse-HOP-Size ist die Anzahl der Samples, um welche das Zeitfenster verschoben wird, um den nächsten Frame zu generieren. Manchmal ist mit der HOP-Size auch die Überlagerung der Frames gemeint anstelle der Verschiebung. Bei der Wahl der Analyse-HOP-Size ist zu beachten, dass ein kleiner Wert zu einer besseren Zeitauflösung führt, jedoch zu einer schlechteren Frequenzauflösung. Ein grösserer Wert führt zu einer besseren Frequenzauflösung, jedoch zu einer schlechteren Zeitauflösung. Die übliche Wahl für die Analyse-HOP-Size ist $N/2$ oder $N/4$, wobei N die Länge des Frames ist.

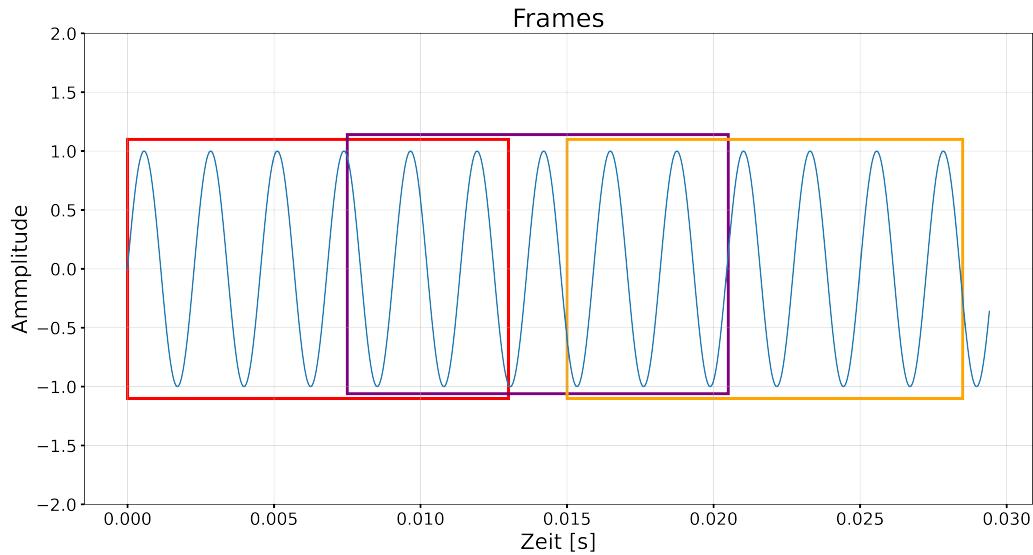


Abbildung 5: Aufteilung des Signals in Frames

Synthese: Neben der Analyse-HOP-Size muss auch die Synthese-HOP-Size bestimmt werden. Die Synthese-HOP-Size ist die Überlagerung oder auch die Verschiebung der Frames, wenn diese wieder zusammengesetzt werden. Das Zusammenspiel beziehungsweise das Verhältnis von Analyse-HOP-Size und Synthese-HOP-Size entscheidet über die resultierende Streckung des Audiosignals. Dieser

4 TIME-SCALE MODIFICATION

Wert wird als Stretching Factor bezeichnet und wird in der Regel mit α bezeichnet. Er ist abhängig von der gewünschten Streckung oder Stauchung des Signals. Die Formel für den Stretching Factor ist in Gleichung (2) beschrieben. Die Frames werden nun also wie in Abbildung 6 miteinander überlagert.

$$\alpha = \frac{H_s}{H_a} \quad (2)$$

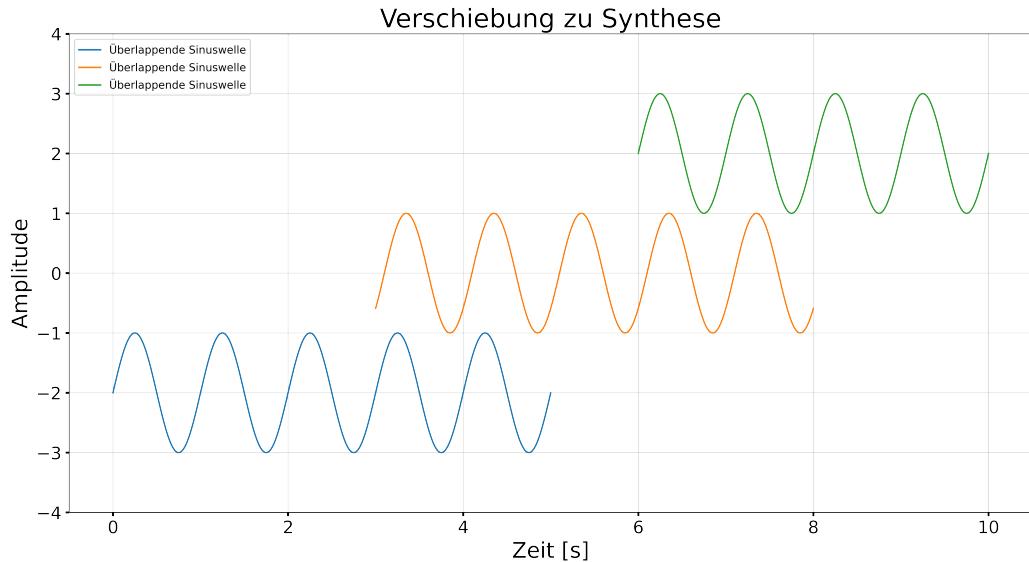


Abbildung 6: Überlagerung der Frames

Auf den ersten Blick scheint alles sehr straightforward zu sein. Schaut man jedoch genauer auf das dabei erzeugte Signal, so fallen einige Probleme auf. In Abbildung 7 sind die Problemstellen markiert. Die violette Linie zeigt die ursprüngliche Amplitude des Signals. Es ist klar zu erkennen, dass die Amplituden des neuen Signals stark schwanken und teilweise weit über den ursprünglichen Wert steigen.

Ein weiteres Problem sind die Übergänge zwischen den einzelnen Frames. Diese sind in Abbildung 7 rot markiert. An diesen Stellen kommt es zu sogenannten Phasensprünge, also plötzlichen Änderungen der Phase. Dies kann hörbare Effekte verursachen, wie beispielsweise Klangverzerrungen oder Auslöschen, besonders wenn mehrere Audiosignale miteinander kombiniert werden. Dazu mehr im Kapitel 6.2. Wie diese Probleme gelöst werden können, wird im nächsten Abschnitt erläutert.

4 TIME-SCALE MODIFICATION

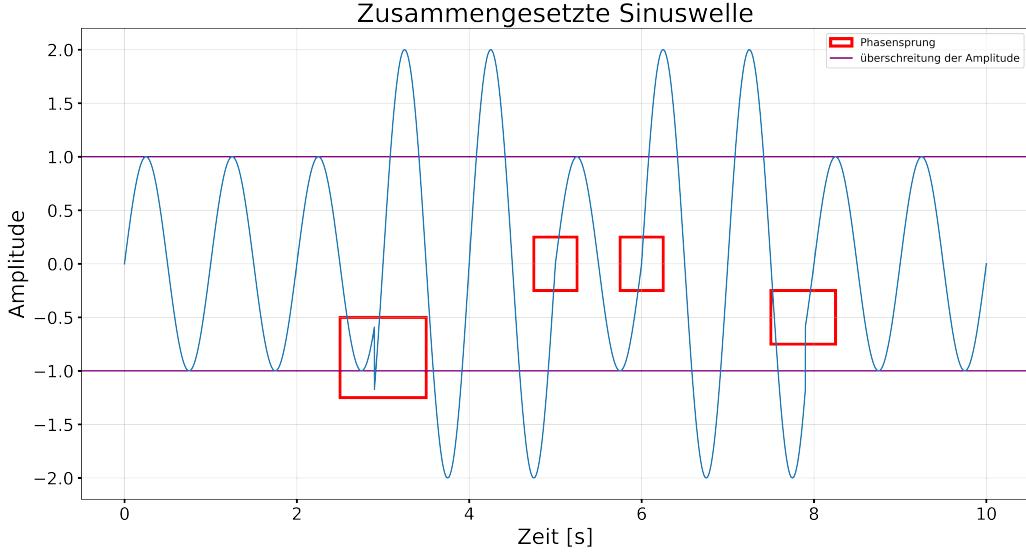


Abbildung 7: Artefakte bei der Synthese von Frames

Window function: Um die eben beschriebenen Probleme zu lösen, wird eine sogenannte Window Funktion verwendet. Eine Window-Funktion in der Signalverarbeitung ist eine mathematische Funktion, die auf ein Signal angewendet wird, um es für die Analyse oder Verarbeitung vorzubereiten. Im Zusammenhang der Time-Scale Modification wird die Window-Funktion verwendet, um die Amplituden der Frames und den Enden, bzw. den Übergängen, zu dämpfen. Die Wahl der Window-Funktion ist dabei entscheidend. Die bekanntesten Window-Funktionen sind das Rechteckfenster, das Hann-Fenster und das Hamming-Fenster. Die Wahl der Window-Funktion ist abhängig von der Anwendung. Im Falle der Time-Scale Modification wird meistens das Hann-Fenster verwendet. Die Formel für das Hann-Fenster ist in Gleichung (3) beschrieben. Dabei ist r der Index des Samples und N die Länge des Frames [5].

$$w(r) = \frac{1}{2} \left(1 - \cos \left(\frac{2\pi r}{N} \right) \right) \quad (3)$$

Nach Anwendung der Window Function auf einen Frame lässt sich in Abbildung 8 erkennen, dass die Amplituden der Frames nun gedämpft sind. Die Amplituden der Frames sind nun nicht mehr so stark schwankend und liegen näher an den ursprünglichen Amplituden des Signals. Die Übergänge zwischen den Frames sind nun auch nicht mehr so stark ausgeprägt. Die Phasensprünge sind jedoch immer noch vorhanden.

4 TIME-SCALE MODIFICATION

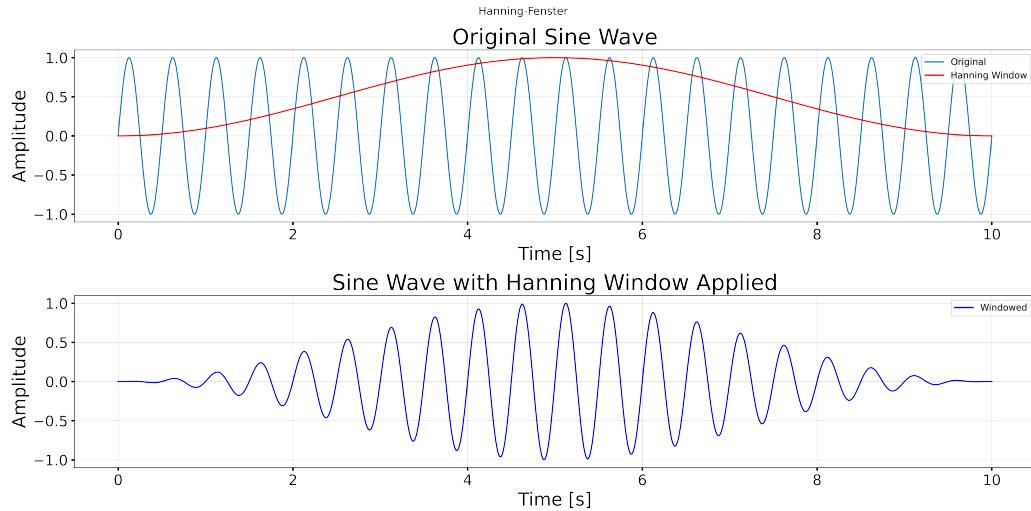


Abbildung 8: Hanning Fenster angewendet auf einen Frame

Dan nun die Enden eines jeden Frames abgeschwächt sind, können die Frames nun wie in Abschnitt der Synthese beschrieben, wieder miteinander überlagert werden. Veranschaulicht wird das in Abbildung 9.

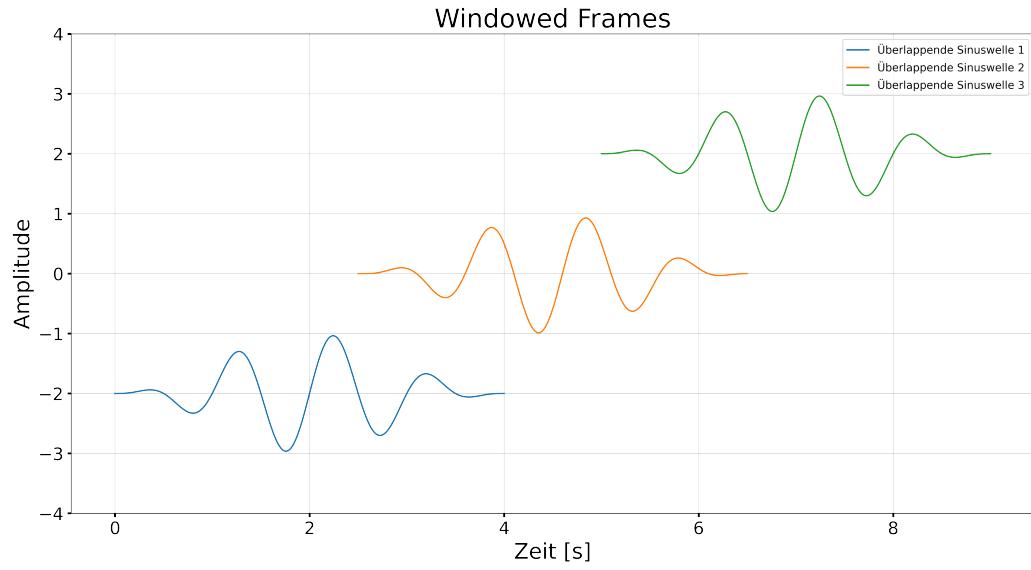


Abbildung 9: Überlagerung der Frames nach Anwendung der Window Funktion

Danach werden die Frames wieder zusammengefügt. Das Ergebnis ist in Abbildung 10 dargestellt. Es ist klar zu erkennen, dass die Amplituden der Frames nun nicht mehr so stark schwanken und die Übergänge zwischen den Frames nicht mehr so stark ausgeprägt sind. Die Phasensprünge sind jedoch immer noch vorhanden. Welche weitere Schritte unternommen werden können, um die Phasensprünge zu minimieren, wird im nächsten Abschnitt kurz erläutert.

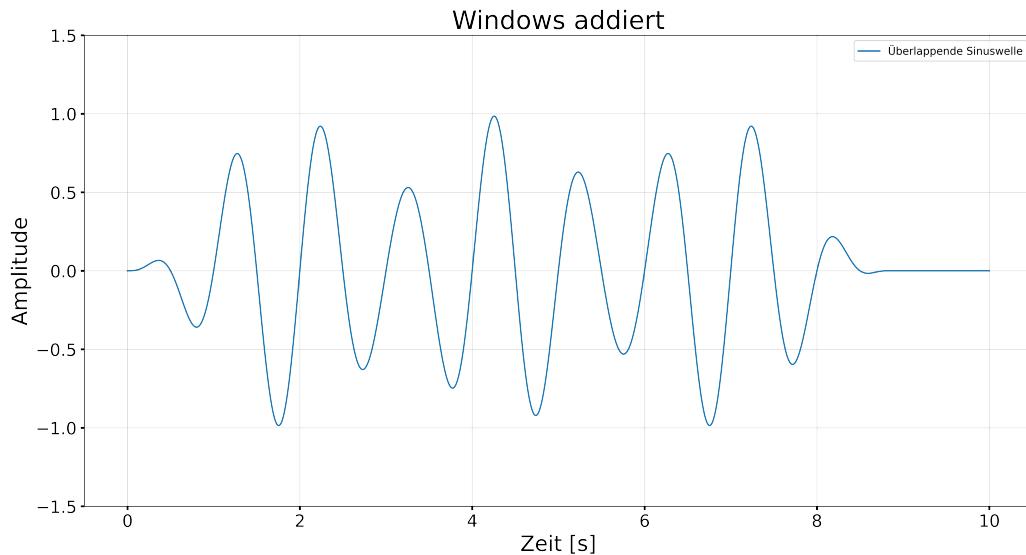


Abbildung 10: Addition der Frames nach Anwendung der Window Funktion

4.2 Waveform Similarity Overlap-Add (WSOLA)

Um die im vorherigen Kapitel aufgeführten Probleme zu minimieren, wurde das Waveform Similarity Overlap-Add (WSOLA) Verfahren entwickelt. WSOLA ist eine erweiterte Methode des Overlap-Add-Verfahrens, um die hörbaren Artefakte, welche durch Phasensprünge und Amplitudenschwankungen entstehen zu minimieren. WSOLA verbessert die Qualität der Zeitskalierung, indem es die besten Überlappungspunkte zwischen benachbarten Frames sucht, um eine möglichst glatte Übergang zu gewährleisten.

Im Gegensatz zum einfachen Overlap-Add-Verfahren analysiert WSOLA die Waveform und verschiebt sie leicht, um die Ähnlichkeit an den Überlappungsbereichen zu maximieren. Dies reduziert Phasensprünge und führt zu einem natürlicheren Klang. Dabei besteht WSOLA aus den folgenden Schritten:

1. Unterteilung des Signals in überlappende Frames.
2. Suche nach den optimalen Überlappungspunkten, um die Ähnlichkeit der Wellenformen zu maximieren.
3. Anpassung und Verschiebung der Frames, um die besten Überlappungspunkte zu erreichen.
4. Zusammensetzen der Frames, um das skalierten Audiosignal zu erzeugen.

Für die passende Verschiebung der Frames wird ein Algorithmus verwendet, der die Ähnlichkeit der Wellenformen maximiert. Dieser Algorithmus basiert auf der Kreuzkorrelation. Damit lässt sich die Ähnlichkeit zwischen zwei Signalen messen. Dies zeigt, wie gut ein Signal mit einem verschobenen zweiten Signal übereinstimmt.

Die Kreuzkorrelation $R_{xy}(k)$ von zwei diskreten Signalen $x[n]$ und $y[n]$. Hierbei gibt k die Verschiebung an. Ein hoher Wert von $R_{xy}(k)$ bedeutet, dass die Signale in diesem Bereich gut übereinstimme [10]. Die Kreuzkorrelation kann wie folgt berechnet werden:

$$R_{xy}(k) = \sum_n x[n] \cdot y[n + k] \quad (4)$$

Somit kann WSOLA eine hohe Qualität bei der Modifikation der Wiedergabedauer des Audiosignals erreichen und dabei die hörbaren Artefakte minimieren. Allerdings kann WSOLA nur die dominantesten Frequenzen korrekt verarbeiten, da die Suche nach den optimalen Überlappungspunkten auf den Amplituden der Frames basiert. Feinere Frequenzen werden dabei vernachlässigt bzw. nur teilweise korrekt verarbeitet [9].

4.3 Phase Vocoder (PV)

Die im vorherigen Teil betrachteten Verfahren reduzierten sich lediglich auf den Zeitbereich. Mit dem Phase Vocoder steht ein leistungsstarkes Tool für die Manipulation der zeitlichen und spektralen Eigenschaften zu Verfügung und ist bestens geeignet für die Anwendung der TSM.

Wie bereits erwähnt, führen gezeigten Methoden zur TSM, wie das OLA, oft zu unerwünschten Artefakten . Der Phase Vocoder überwindet diese Einschränkungen durch eine Technik, die auf der Kurzzeit-Fourier-Transformation (STFT) basiert.

Diese Technik ermöglicht es, die zeitlichen Eigenschaften eines Signals unabhängig von dessen Frequenzinhalt zu modifizieren, indem die Phase und Amplitude der spektralen Komponenten des Signals präzise angepasst werden.

OLA im Phase Vocoder: Die Grundlage für den Phase-Vocoder bildet das Overlap-Add-Verfahren, welches bereits im Kapitel refsec:ola beschrieben wurde. Denn auch der Phase-Vocoder zerlegt das Audiosignal zuerst in überlappende Frames. Doch anstelle das die Verschiebung direkt vorgenommen wird, werden die einzelnen Frames zuerst in den Frequenzbereich transformiert. Dies geschieht durch die Kurzzeit-Fourier-Transformation welche im nächsten Abschnitt behandelt wird. Nach der Transformation in den Frequenzbereich findet dann die eigentliche Zeitdehnung statt. Nach der Manipulation entlang der Zeitachse wird das Signal wieder in Zeitbereich transformiert und die Frames wieder zusammengefügt.

Diskrete Fourier-Transformation: Bevor die Kurzzeit-Fouriertransformation betrachtet werden kann, muss zuerst die Fouriertransformation erläutert werden. Die Fouriertransformation ist eine mathematische Methode, die ein Signal in den Frequenzbereich transformiert [10].

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

Die diskrete Fouriertransformation ist eine rechenintensive Operation und kann zu Verzögerungen in der Verarbeitung führen. Die meisten Implementationen der Fouriertransformation verwenden die schnelle Fouriertransformation (FFT), um die Berechnung zu beschleunigen. Die FFT ist ein Algorithmus, der die diskrete Fouriertransformation in $O(n \log n)$ Zeit berechnet. Eine Voraussetzung dafür ist jedoch, dass die Länge des Signals eine Potenz von 2 ist. Somit wird bei der Erstellung der Frames darauf geachtet, dass diese eine Länge von Potenzen von 2 aufweisen [7].

Kurzzeit-Fourier-Transformation (STFT): Wie im vorherigen Abschnitt angesprochen, wird das Audiosignal zuerst in den Frequenzbereich transformiert. Für den Phase Vocoder (PV) wird dafür die Kurzzeit-Fouriertransformation (STFT) benötigt. Die STFT ist eine leistungsstarke Technik in der digitalen Signalverarbeitung, die verwendet wird, um die spektralen Eigenschaften eines Signals in kurzen Zeitfenstern zu analysieren. Die STFT zerlegt ein Signal in überlappende Frames, die dann in den Frequenzbereich transformiert werden. Dies ermöglicht

4 TIME-SCALE MODIFICATION

es, die spektralen Eigenschaften eines Signals im Zeitverlauf zu verfolgen und zu modifizieren.

Die STFT eines Signals $x(t)$ wird durch die Gleichung (5) definiert, wobei $X(m, k)$ die STFT des Signals ist, $x_m(r)$ das m -te Frame des Signals ist, $w(r)$ die Fensterfunktion ist, und N die Länge des Frames ist.

$$X(m, k) = \sum_{r=-N/2}^{N/2-1} x_m(r)w(r) \exp(-2\pi i kr/N) \quad (5)$$

Durch die Anwendung der STFT auf ein Audiosignal wird ein Spektrogramm generiert. Dies bildet die Amplitude und Phase auf der Zeit- und Frequenzebene ab. Ein Beispiel für ein Spektrogramm ist in Abbildung 11 dargestellt. Die Farbskala gibt dabei die Amplitude des Signals an, wobei helle Farben hohe Amplituden und dunkle Farben niedrige Amplituden repräsentieren. Das Spektrogramm ermöglicht es, die spektralen Eigenschaften eines Signals im Zeitverlauf zu visualisieren und zu analysieren. Dies ist besonders nützlich bei der Zeitdehnungsmodifikation von Audiosignalen, da es ermöglicht, die spektralen Komponenten des Signals präzise zu manipulieren.

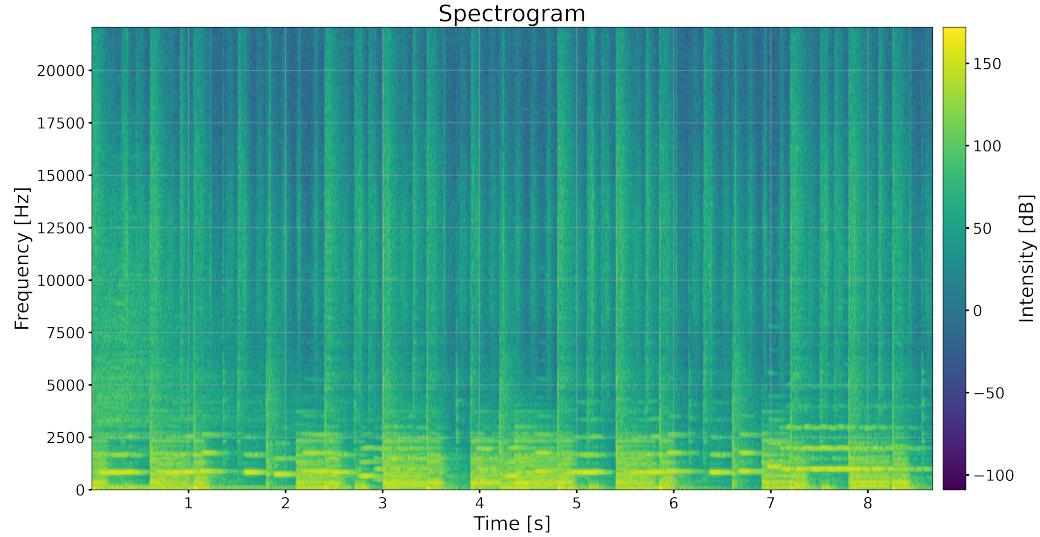


Abbildung 11: Spectrogram nach der STFT

Die Kurzzeit-Fourier-Transformation unterliegt einigen Einschränkungen. Zum einen ist die Auflösung der Frequenzen abhängig von der Länge des Frames. Die

maximale Auflösung der Frequenzen berechnet sich durch Gleichung (6), wobei f_s die Abtastrate des Signals und N die Länge des Frames ist.

$$\Delta f = \frac{f_s}{N} \quad (6)$$

Ein zu kleiner Frame kann dazu führen, dass die Frequenzen nicht korrekt erkannt werden. Ein zu grosser Frame kann dazu führen, dass die Frequenzen nicht mehr lokalisiert werden können. Diese Faktoren sind entscheidend für die Qualität der Zeitdehnungsmodifikation.

Die maximale erkennbare Frequenz wird durch die Nyquist-Frequenz bestimmt. Die Nyquist-Frequenz, gemäss Gleichung (7), ist die Hälfte der Abtastrate des Signals.

$$f_{\text{nyquist}} = \frac{f_s}{2} \quad (7)$$

Ein weiteres Problem ist die Phaseninformation. Diese ist entscheidend für die Rekonstruktion des Signals im Zeitbereich. Die Phaseninformation kann jedoch durch die STFT verloren gehen, was zu Phasensprüngen führen kann, die hörbare Artefakte verursachen. Um diese Probleme zu lösen, wird im Phase-Vocoder die Phaseninformation präzise verarbeitet. Dazu mehr im nächsten Abschnitt.

Lineare Interpolation: Nach der Kurzzeit-Fourier-Transformation (STFT) werden die spektralen Komponenten des Signals präzise manipuliert, um die Zeitdehnung zu erreichen. Die lineare Interpolation ermöglicht es, die spektralen Eigenschaften eines Signals im Zeitverlauf zu modifizieren, ohne die Tonhöhe zu beeinflussen. Die lineare Interpolation wird durch die Gleichung (8) definiert, wobei $X(m, k)$ die STFT des Signals ist, $X'(m, k)$ die interpolierte STFT des Signals ist und α der Stretching Factor ist. Die Gleichung beschreibt, wie die interpolierte STFT $X'(m, k)$ als gewichtete Kombination der STFTs $X_1(m, k)$ und $X_2(m, k)$ der benachbarten Zeitpunkte t und $t + 1$ berechnet wird. Der Stretching Factor α steuert das Ausmass der Interpolation zwischen den beiden STFTs. Wenn $\alpha = 0$, wird die STFT des ersten Zeitpunkts verwendet, wenn $\alpha = 1$, wird die STFT des zweiten Zeitpunkts verwendet, und $0 < \alpha < 1$ führt zu einer Interpolation zwischen den beiden [12].

$$X(m, k) = (1 - \alpha) \cdot X_1(m, k) + \alpha \cdot X_2(m, k) \quad (8)$$

4.4 Phase Vocoder mit Phase Locking (PVPL)

Der Vollständigkeitshalber wird hier das Phase Locking des Phase Vocoder beschrieben. Aufgrund der fehlenden Implementation dazu wird dies nur kurz

überflogten. PVPL ist eine Technik, die verwendet wird, um die Phaseninformation der spektralen Komponenten eines Signals präzise zu verarbeiten. Die Phaseninformation ist entscheidend für die Rekonstruktion des Signals im Zeitbereich. Durch das Phase Locking wird sichergestellt, dass die Phaseninformation der spektralen Komponenten konsistent bleibt, auch wenn das Signal manipuliert wird. Dies minimiert Phasensprünge und verhindert hörbare Artefakte[9].

4.5 Harmonic-Percussive Source Separation (HPSS)

Auch Harmonic-Percussive Source Separation (HPSS) wird hier nur kurz zur Vollständigkeit erläutert. Bei HPSS handelt es sich um eine Technik, die im Zusammenhang mit der Time-Scale Modification (TSM) von Audiosignalen verwendet wird. HPSS ist eine Methode zur Trennung von harmonischen und perkussiven Komponenten eines Signals. Harmonische Komponenten sind periodische Schwingungen, die durch harmonische Oberschwingungen charakterisiert sind, wie z.B. Musikinstrumente oder Gesang. Perkussive Komponenten sind nicht-periodische Schwingungen, die durch Impulse oder Stöße charakterisiert sind, wie z.B. Schlagzeug oder Percussion. HPSS ermöglicht es, harmonische und perkussive Komponenten eines Signals getrennt zu analysieren und zu modifizieren, was in der Musikproduktion und Audioverarbeitung nützlich ist [9].

5 Implementation

In diesem Kapitel wird die Implementierung des Projekts detailliert beschrieben. Es wird erläutert, wie die einzelnen Komponenten entwickelt wurden und wie sie nahtlos zusammenarbeiten. Zunächst wird der Prozess des Einlesens der Daten und deren Weiterleitung durch die Pipeline beschrieben. Anschliessend wird die Gestaltung und Funktionalität der Benutzeroberfläche erläutert. Zum Schluss wird auf die Methoden und Techniken der Audioverarbeitung eingegangen. Abbildung 12 zeigt den groben Datenfluss durch die Implementierung.

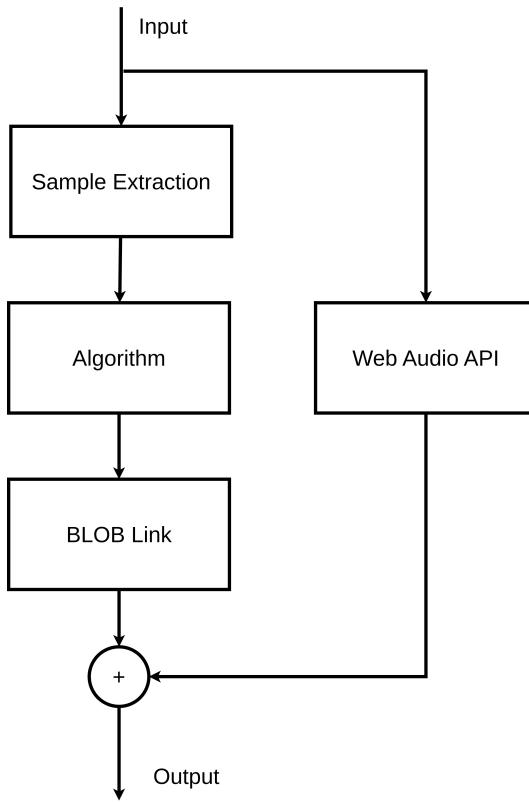


Abbildung 12: Datenflussdiagramm der Implementierung

5.1 File Handling

Damit der Benutzer Audiodaten hochladen kann, muss eine Möglichkeit geschaffen werden, die Daten zu empfangen und zu verarbeiten. Dafür wurde eine Schnittstelle entwickelt, die es dem Benutzer ermöglicht, Audiodaten hochzuladen. Das

5 IMPLEMENTATION

File wird über ein Formular hochgeladen welches über einen Button im Header aktiviert wird. Das aufgerufene Formular besteht aus einem File-Input Feld, in dem der Benutzer die Datei auswählen kann, die hochgeladen werden soll. Wenn der Benutzer die Datei ausgewählt hat, wird ein Event-Listener auf das Input-Feld registriert, der aufgerufen wird. In diesem Event-Listener wird eine Funktion aufgerufen welche die Datei einliest und in ein Array von Bytes umgewandelt. Dieses Array von Bytes wird dann an die Datenverarbeitung weitergegeben. Dazu mehr im Kapitel Signalverarbeitung 5.2.

Wavefile: Für die Implementation wurde entschieden mit Wavefiles, auch bekannt als WAV-Datei (Waveform Audio File Format), zu arbeiten. Diese sind unkomprimiert und können somit direkt eingelesen werden. Ein Wavefile ist ein Dateiformat für die Speicherung von digitalen Audiodaten und speichert diese in einem unkomprimierten Format, was bedeutet, dass es eine genaue und verlustfreie Darstellung des Originalsignals ermöglicht[8]. Es ist eines der häufigsten Dateiformate für die Speicherung von Audiodaten auf Computern und wird häufig in der Musikproduktion, in Multimedia-Anwendungen und in der Audiobearbeitung verwendet. Wavefiles enthalten Informationen wie die Abtastrate, die Bit-Tiefe (Anzahl der Bits pro Sample), die Anzahl der Audiokanäle (mono oder stereo) und die Länge der Audiodaten.

Um effektiv and die Samples und andere wesentliche Informationen zu gelangen, wurde die Bibliothek wavefile.js verwendet. WaveFile.js ist eine JavaScript-Bibliothek, die es ermöglicht, Wavefiles einfach einzulesen, bearbeitet und gespeichert werden. Sie bietet Funktionen zur direkten Manipulation von Audiodaten, Formatkonvertierung, Bearbeitung von Metadaten und Erstellung neuer Wavefiles. WaveFile.js ist sowohl im Browser als auch in Node.js einsetzbar und erleichtert die Arbeit mit Audiodateien erheblich [2]. Bei den verwendeten funktionen handelt es sich um *getSamples()* um die Samples zu extrahieren und *fromScratch()* um ein Wavefile wieder zu erstellen. Zusätzlich wurden noch einige funktionen gebraucht um Eigenschaften des Files zu extrahieren, wie Anzahl Channels, Bitrate und Samplerate.

Das hochgeladene Wavefile wird zusätzlich, bevor es zur Datenverarbeitung weitergegeben wird, noch direkt an ein Audio Element übergeben, um das originale, unverarbeitete Audiofile abspielen zu können. Die geschieht über das erstellen eines Blob Objektes.

Blob URL: Ein Blob (Binary Large Object) ist ein JavaScript-Objekt, das binäre Daten speichert. Es wird häufig verwendet, um grosse Mengen von binären

5 IMPLEMENTATION

Daten, wie z.B. Bilder, Videos oder Audiodaten, zu handhaben. Blobs bieten eine effiziente Möglichkeit, solche Daten zu speichern und zu verarbeiten. Ein Blob-Objekt kann aus verschiedenen Quellen erstellt werden, wie z.B. Arrays von Byte-Daten, Text oder anderen Blobs[6]. Bei Audiodaten können diese aus einer Datei oder einem anderen binären Datenstrom stammen. Ein Blob-Objekt kann mit einem HTML-Audio-Tag verwendet werden, um Audiodaten im Browser abzuspielen. Dazu muss das Blob-Objekt in eine URL umgewandelt werden, die das Audio-Tag verwenden kann.

5.2 Signalverarbeitung

Nun werden ein paar Schlüsselpunkte und Implementationseinscheidungen der Signalverarbeitung erläutert. Im grossen und ganzen folgte die Implementation den in Kapitel 4 beschriebenen Methoden. Dabei wurde darauf geachtet, dass die Methoden möglichst einfach und verständlich implementiert wurden. Da JavaScript jedoch nicht für die Verarbeitung von Audiodaten optimiert ist, mussten einige Workarounds gefunden werden.

Tensor Flow: Zum Beispiel wurde für die Implementation des Phase Vocoder aus Kapitel 4.3 die Bibliothek TensorFlow verwendet. Denn JavaScript besitzt nativ keinen Support für die Entwicklung mit komplexen Zahlen. Das würde auch die Arbeit mit der Fast Fourier Transformation erschweren. Aus diesem Grund wurde entschieden, die Drittbibliothek TensorFlow zu verwenden. TensorFlow ist eine Open-Source-Softwarebibliothek für maschinelles Lernen, die von Google entwickelt wurde [1]. Sie bietet eine Vielzahl von Funktionen und Werkzeugen zur Entwicklung und Implementierung von maschinellen Lernalgorithmen. Doch der Grund, weshalb in dieser Arbeit auf TensorFlow zurückgegriffen wurde, ist die Möglichkeit, die Fast Fourier Transformation zu verwenden.

Damit dies funktionierte, mussten die JavaScript-Arrays zuerst in Tensoren umgewandelt werden. Der Umgang mit Tensoren ist nicht genau gleich wie mit Arrays, weswegen einige Funktionen umgeschrieben werden mussten. Die Arbeit mit Tensoren hat auch seine Vorzüge, denn TensorFlow bietet eine Vielzahl von Funktionen zur Verarbeitung von Tensoren, wie z.B. mathematische Operationen und lineare Algebra, was der Numpy-Bibliothek in Python ähnelt. Wenn von Anfang an mit Tensoren gearbeitet worden wäre, hätte dies die Implementierung vereinfacht und die Performance verbessert.

5.3 User Interface

Eines der Hauptziele dieser Arbeit besteht darin, die Verarbeitung von Audiodaten auf eine anschauliche und intuitive Weise zu präsentieren. Dies beinhaltet die Gestaltung einer benutzerfreundlichen Benutzeroberfläche, die leicht zugänglich ist und es dem Benutzer ermöglicht, die verarbeiteten Audiodaten strukturiert zu visualisieren und miteinander zu vergleichen. Des Weiteren ist vorgesehen, verschiedene Visualisierungstechniken zu implementieren, wie beispielsweise Spektrogramme, die für alle Audiodateien verfügbar sind, sowie Echtzeit-Waveforms und Spektrogramme speziell für das Original-Audiofile.

Tailwind CSS: Für die Gestaltung der Benutzeroberfläche wurde das CSS-Framework Tailwind CSS verwendet. Tailwind CSS verwendet einen Utility-First-Ansatz, der es ermöglicht, Eigenschaften wie Grösse, Abstand, Farbe und Schriftart direkt im HTML-Markup festzulegen. Dieser Ansatz ermöglicht eine präzise und flexible Definition des Erscheinungsbilds durch die Kombination von Klassen. Darüber hinaus ist Tailwind CSS hochgradig konfigurierbar, wodurch benutzerdefinierte Anpassungen wie Farben, Schriftarten und Abstände ermöglicht werden. Zur Optimierung der Performance bietet Tailwind CSS Werkzeuge wie PurgeCSS und JIT-Kompilierung, die die Dateigrösse reduzieren. Aufgrund dieser Eigenschaften hat sich Tailwind CSS als effizientes Werkzeug zur Erstellung benutzerdefinierter Benutzeroberflächen in der Frontend-Entwicklung etabliert [11].

Web Audio API: Zu Beginn der Arbeit wurden verschiedene Technologien evaluiert. Dabei wurde auch die Web Audio API in Betracht gezogen. Die Web Audio API ist eine JavaScript-API zur Erzeugung, Verarbeitung und Wiedergabe von Audio in Webanwendungen. Sie ermöglicht die Steuerung von Audioquellen, Effektknoten und Zielen, wodurch komplexe Audiopipelines erstellt werden können. Die API unterstützt die Wiedergabe von Audiodaten aus Dateien oder Streams und die Anwendung von Echtzeit-Effekten wie Reverb oder Delay. Außerdem können Audiodaten aus Mikrofonen aufgenommen und die Wiedergabe gesteuert werden [13]. Einige Nodes der Web Audio API:

- **AudioContext:** Schnittstelle zur Audiohardware
- **AnalyserNode:** Analysiert Audiodaten
- **ScriptProcessorNode:** Benutzerdefinierte Audioverarbeitung
- **MediaStreamAudioSourceNode:** Liest Audiodaten aus einem MediaStream

5 IMPLEMENTATION

Die API ermöglicht auch benutzerdefinierte Nodes zur Implementierung komplexer Audioverarbeitungsalgorithmen. Abbildung 13 zeigt die Verknüpfung der einzelnen Nodes.

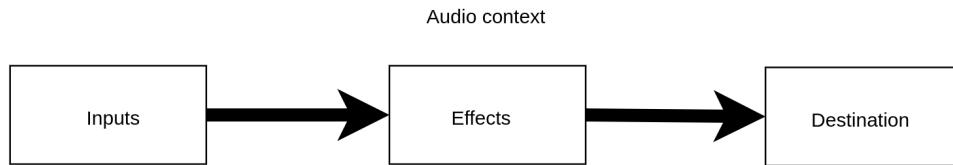


Abbildung 13: Visualisierung der Web Audio API Nodes

Das Problem bei der Verwendung der Web Audio API liegt darin, dass sie auf die Echtzeitverarbeitung von Audiodaten ausgelegt ist. Das bedeutet, dass die Audiodaten in kleinen Blöcken verarbeitet werden, was zu einer höheren Komplexität und speziellen Implementierungserfordernissen führt. Daher wurde entschieden, die Web Audio API nicht für die Verarbeitung der Audiodaten zu verwenden. Stattdessen wurde die zuvor erwähnte Bibliothek Wavefile.js genutzt.

Da im Rahmen der Evaluierung der Web Audio API Zeit investiert wurde, um ihr Potenzial und ihre Funktionsweise zu prüfen, wurde entschieden, die Web Audio API für die Visualisierung der Audiodaten zu verwenden. Auf diese Weise konnten die bereits getätigten Aufwände genutzt werden. Ein AnalyserNode wurde erstellt, um die Audiodaten zu analysieren und die Ergebnisse in Echtzeit anzuzeigen.

5.4 Probleme

Im Rahmen dieses Projektes wurden viele neue theoretische Konzepte, Technologien und Verfahren erkundet, was zu zahlreichen Problemen führte, die gelöst werden mussten. Das Projekt war ein Lernprozess mit vielen Herausforderungen und befindet sich eindeutig noch in der Entwicklungsphase. Daher basieren einige Teile der Implementierung auf Workarounds. Die Struktur des Codes ist noch nicht optimal, und es gibt viele Verbesserungspotenziale, auch in Bezug auf die Performance.

Performance: Eine Wartezeit von einigen Sekunden ist bei der Verarbeitung von Audiodaten kaum zu vermeiden. Die Verarbeitung ist ein rechenintensiver Prozess, besonders bei grossen Audiodateien oder komplexen Algorithmen. Bei

5 IMPLEMENTATION

dieser Implementierung gibt es jedoch einige Entscheidungen, die die Performance beeinträchtigen und die Verarbeitungszeit verlängern. Die Anzahl Frames entscheidet über den Aufwand der Verarbeitung. Je mehr Frames erzeugt werden, desto aufwändiger wird die in Kapitel 4.3 beschriebene Phase Vocoder Methode. Eine Möglichkeit, die Performance zu verbessern, wäre die Reduzierung der Anzahl der Frames, was jedoch die Qualität des Audiosignals beeinträchtigen würde.

Eine weitere Möglichkeit wäre es, die Verarbeitung der Audiodaten in einen Web Worker auszulagern. Ein Web Worker ist ein Hintergrundprozess, der im Browser ausgeführt wird und es ermöglicht, rechenintensive Aufgaben auszulagern und die Benutzeroberfläche zu entlasten [14]. Durch die Verwendung von Web Workern kann die Leistung verbessert und die Reaktionsfähigkeit der Benutzeroberfläche erhöht werden, da die bereits verarbeiteten Daten geladen werden können, ohne auf die vollständige Verarbeitung warten zu müssen.

Ein weiteres Problem für die Performance ist die Erzeugung der Spektrogramme. Die derzeitige Implementierung führt eine zweite Kurzzeit-Fouriertransformation durch, um das Spektrogramm zu erzeugen. Dies ist jedoch nicht optimal, da die Daten bereits verarbeitet wurden und die Spektrogramme direkt aus diesen Daten extrahiert werden könnten. Eine Möglichkeit, die Performance zu verbessern, wäre es, die Spektrogramme direkt aus den verarbeiteten Daten zu generieren, anstatt eine zusätzliche Transformation durchzuführen. Dies würde die Verarbeitungszeit verkürzen und die Leistung verbessern.

Paradigma: Die derzeitige Struktur des Codes ist auf eine prototypische und schnelle Implementierung ausgelegt. Dies führt dazu, dass der Code nicht optimal strukturiert ist und es zu Redundanzen kommt. Es wurde versucht, den Code auf verschiedene Dateien aufzuteilen und die Funktionen sinnvoll zu verallgemeinern. Dennoch gibt es viel Verbesserungspotential.

Eine Möglichkeit, die Struktur des Codes zu verbessern, wäre es, die Funktionen in sinnvolle Klassen zu gruppieren und die Datenverarbeitung in separate Module auszulagern. Dies würde die Lesbarkeit und Wartbarkeit des Codes verbessern und die Redundanz reduzieren. Außerdem könnte die Verwendung von Design Patterns wie MVC (Model-View-Controller) oder MVVM (Model-View-ViewModel) helfen, die Struktur des Codes zu verbessern und die Trennung von Daten, Logik und Präsentation zu erleichtern.

Framework Eine andere Überlegung wäre, auf ein Framework zurückzugreifen, das die Struktur des Codes vorgibt. Ein Framework wie React oder Angular würde

5 IMPLEMENTATION

die Struktur des Codes vorgeben und die Entwicklung erleichtern. React ist eine JavaScript-Bibliothek für die Erstellung von Benutzeroberflächen, die von Facebook entwickelt wurde. Sie ermöglicht die Entwicklung von komponentenbasierten UIs für Webanwendungen [15]. Dies würde auch die Implementierung in TypeScript ermöglichen.

TypeScript ist eine von Microsoft entwickelte Programmiersprache, die statische Typisierung für JavaScript bietet. Sie erweitert JavaScript um statische Typen, Klassen und Module, was zu sichererem und besser strukturiertem Code führt [16]. TypeScript-Code wird in JavaScript kompiliert und findet breite Anwendung in der Web- und Softwareentwicklung. Allerdings würde dies bedeuten, dass die Implementierung von Grund auf neu gemacht werden müsste.

Redundanz Ein weiteres Problem ist die Redundanz des Codes. Es gibt viele Stellen im Code, an denen ähnliche Funktionen wiederholt werden. Dies führt zu erhöhter Komplexität und erhöhtem Wartungsaufwand. Eine Möglichkeit, die Redundanz zu reduzieren, wäre die Verwendung von Funktionen und Klassen, um gemeinsame Funktionalitäten zu kapseln und wiederzuverwenden. Dies würde die Wartbarkeit des Codes verbessern und die Lesbarkeit erhöhen.

Vor allem bei der Gestaltung der HTML-Elemente gibt es viele Redundanzen. Die HTML-Elemente werden in den verschiedenen Komponenten immer wieder neu erstellt. Eine Möglichkeit, die Redundanz zu reduzieren, wäre es, die HTML-Elemente in separate Dateien auszulagern und sie bei Bedarf zu importieren. Dies würde ebenfalls die Wartbarkeit des Codes verbessern und die Lesbarkeit erhöhen.

6 Auswertung

In diesem Kapitel werden die Ergebnisse der Implementierung analysiert und diskutiert. Dabei wird auf die Qualität der Ergebnisse eingegangen und die Unterschiede in den Ergebnissen, zwischen den verschiedenen Methoden aufgezeigt. Dabei wurde ein 440 Herz Sinus als Referenzsignal Signal verwendet, um die Ergebnisse zu visualisieren. Das Signal hat eine Zeitspanne von fünf Sekunden. Damit lassen sich die Unterschiede zwischen den verschiedenen Methoden besser darstellen und können mit der Theorie in Kapitel 4 verglichen werden. Zudem werden mögliche Schritte zur Verbesserung und der Weiterentwicklung der Implementierung aufgezeigt. Zum Schluss wird ein Rückblick auf die Arbeit gegeben.

6.1 Perfect Reconstruction

Eine Möglichkeit, die Implementierung auf ihre Richtigkeit zu prüfen, zumindest in einem gewissen Rahmen, ist durch die Perfect Reconstruction. Das bedeutet, dass wenn das Eingangssignal den Algorithmus mit einem Streckungsfaktor von eins durchläuft, das Ausgangssignal identisch zum Eingangssignal sein muss.

OLA: Perfect Reconstruction: Um die Perfect Reconstruction für das OLA-Verfahren zu überprüfen, wird das Referenzsignal, ein 440 Hz Sinus, mit einem Streckungsfaktor von 1 durch den Algorithmus geführt. Das Ergebnis ist in Abbildung 14 dargestellt. Dabei ist zu erkennen, dass das Ausgangssignal fast identisch zum Eingangssignal ist. Dies bedeutet, dass das OLA-Verfahren die Perfect Reconstruction weitgehend erfüllt.

Jedoch scheint es am Anfang des Signals zu leichten Abweichungen zu kommen. Da diese Abweichung nur am Anfang auftritt, könnte dies auf einen Fehler bei der Generierung des ersten Frames hindeuten. Diese Abweichungen sind jedoch sehr kurz und treten nicht periodisch auf, sodass ausgeschlossen werden kann, dass sie hörbar sind.

6 AUSWERTUNG

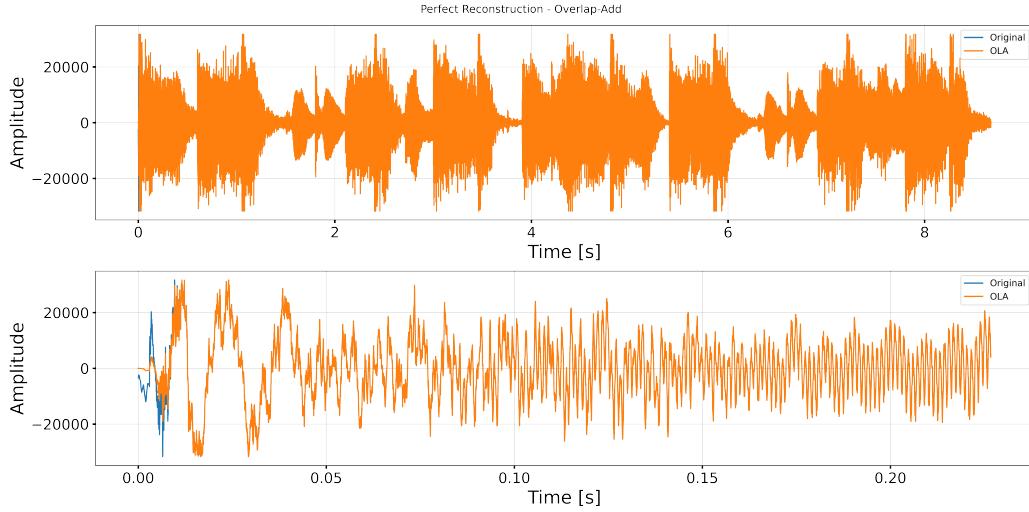


Abbildung 14: Visualisierung der Perfect Reconstruction für OLA

PV: Perfect Reconstruction: Das gleiche Vorgehen wird auch für das Phase-Vocoder-Verfahren (PV) durchgeführt. Dabei wird das Audiosignal mit einem Streckungsfaktor von 1 durch den Algorithmus geführt. Das Ergebnis ist in Abbildung 15 dargestellt. Auch hier sind die Signale fast ausnahmslos identisch. Allerdings sind auch hier leichte Abweichungen am Anfang des Signals zu erkennen.

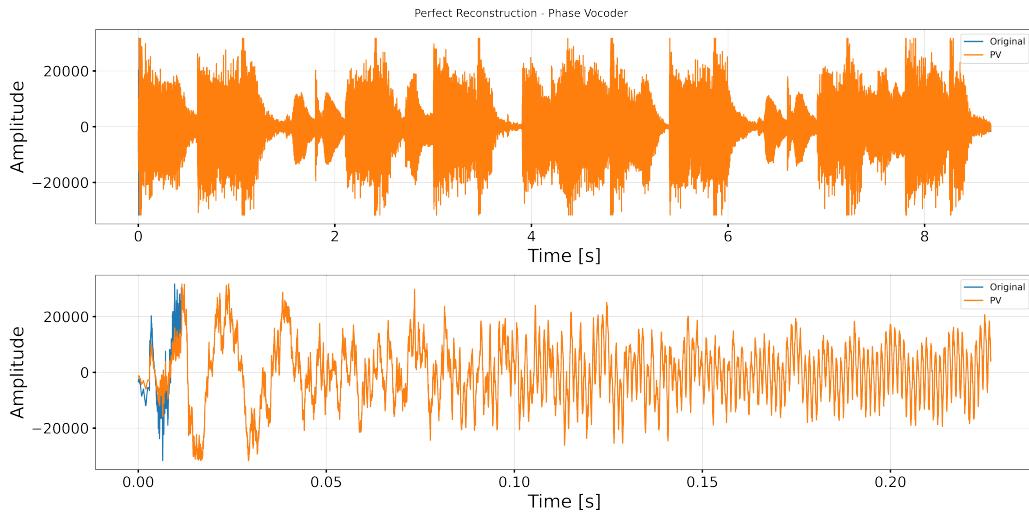


Abbildung 15: Visualisierung der Perfect Reconstruction für PV

6 AUSWERTUNG

Da bei beiden Verfahren die gleichen Probleme auftauchen, lässt sich darauf schliessen, dass diese durch die Implementierung des OLA-Verfahrens hervorgerufen werden. Denn dieses bildet die Grundlage für das PV-Verfahren. Während der Entwicklungsphase wurde mit einem etwas anderen Signal getestet, das andere Eigenschaften am Anfang des Signals aufwies. Daher blieb der Fehler bis zum Schluss unentdeckt.

6.2 Artefakte

Audioartefakte sind unerwünschte Veränderungen oder Störungen in einem Audiosignal, die während der Aufnahme, Bearbeitung, Übertragung oder Wiedergabe auftreten können. Sie können verschiedene Formen annehmen, darunter Rauschen, Verzerrungen, Knacken, Flimmern, Dropouts, Aliasing und Echo. Diese Artefakte können durch eine Vielzahl von Faktoren verursacht werden, wie technische Einschränkungen von Geräten oder Software, unzureichende Datenkompression, unge nau e Signalverarbeitungsalgorithmen oder Störungen während der Übertragung. In der Audioverarbeitung sind Artefakte oft unerwünscht, da sie die Qualität des Audiosignals beeinträchtigen und die Wahrnehmung des Zuhörers stören können. Daher ist es wichtig, Artefakte zu minimieren oder zu eliminieren, um eine hohe Audioqualität zu gewährleisten. Nachfolgend werden die Artefakte der Implementierung betrachtet. Dafür

OLA Streckung 0.5: Als erstes wird das Referenzsignal, ein 440 Hz Sinus, mit dem OLA-Verfahren um den Faktor 0.5 gestreckt. Das Ergebnis ist in Abbildung 16 dargestellt. Dabei ist zu erkennen, dass das Signal tatsächlich gestreckt wurde. Allerdings ist auch klar zu erkennen, dass das Signal nicht mehr genau dem Referenzsignal entspricht. Das bedeutet, dass Artefakte entstanden sind.

6 AUSWERTUNG

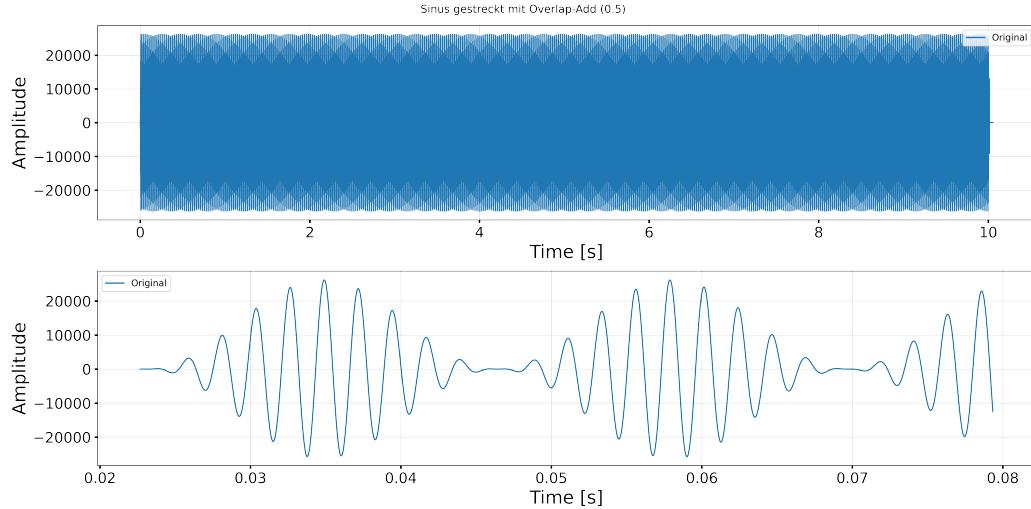


Abbildung 16: Gestrecktes Referenzsignal mit OLA

Die Artefakte entstehen durch die mit dem Referenzsignal überlagerte Fensterfunktion (Hann-Window), wie in Kapitel 4.1 beschrieben. Zwar können dabei die Phasensprünge minimiert werden, allerdings entstehen dadurch andere Artefakte. Vor allem bei der Streckung werden die Abstände zwischen den Frames grösser. Dadurch kommt es zu Schwankungen in der Amplitude. Diese Schwankungen sind in Abbildung 16 gut zu erkennen. Diese Amplitudenschwankungen sind als Flatteneffekt hörbar.

OLA verkürzung 1.5: Als nächstes wird das Referenzsignal mit dem Overlap-Add (OLA) Verfahren um den Faktor 1,5 gestaucht. Das Ergebnis dieser Stauchung ist in Abbildung 17 dargestellt. Ähnlich wie bei der zuvor behandelten Streckung ist die Amplitude des Signals nicht konstant, was auf die Entstehung von Artefakten hinweist. Diese Schwankungen sind jedoch deutlich schwächer, da die Frames stärker überlagert wurden. Die entstehenden Artefakte sind in Abbildung 17 deutlich erkennbar. Es wäre zu erwarten gewesen das mehr Artefakte entstehen, da die Frames stärker überlagert wurden. Dies ist jedoch nicht der Fall. Das könnte daran liegen, dass die einzelnen Frames in diesem Beispiel vorteilhaft übereinander liegen, womit es zu nicht merkbaren Phasensprüngen kommt.

6 AUSWERTUNG

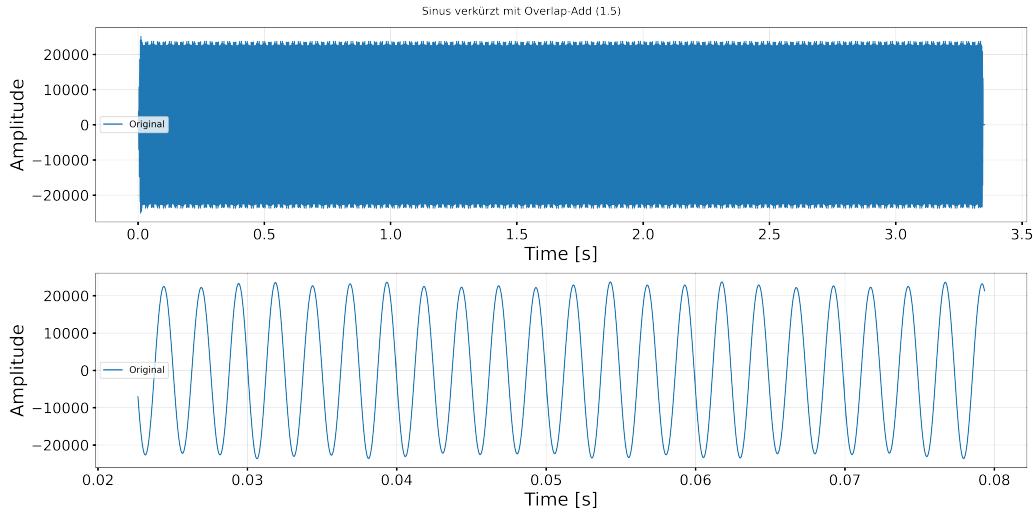


Abbildung 17: Verkürztes Referenzsignal mit OLA

PV Streckung 0.5: Nun wird wie die OLA implementation die PV implementation analysiert. Dabei wurde das Referenzsignal mit dem PV verfahren um den Faktor 0.5 gestreckt. Das Ergebnis ist in Abbildung 18 dargestellt. Hier sind deutlich mehr Phasensprünge zu erkennen. Diese entstehen durch die Veränderung der Phase bei der Streckung.

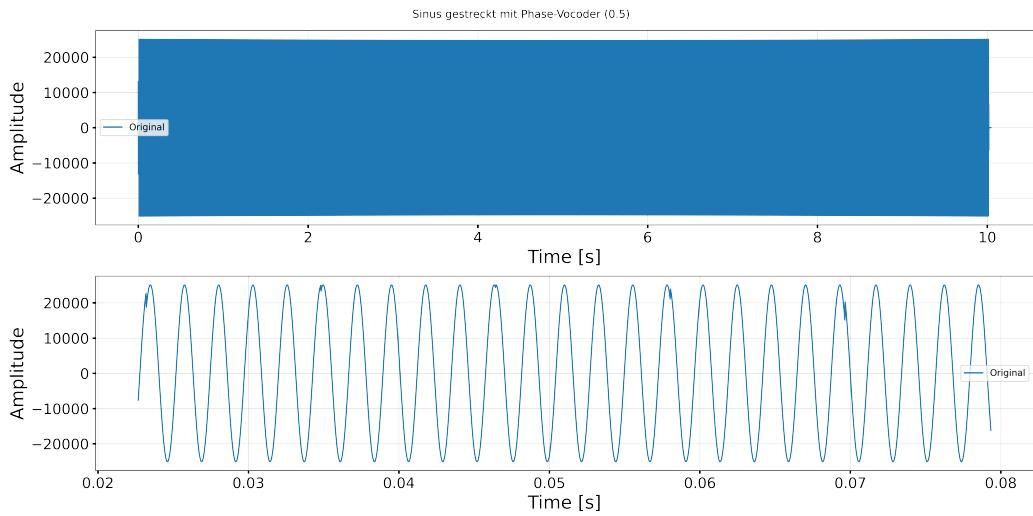


Abbildung 18: Gestrecktes Referenzsignal mit PV

6 AUSWERTUNG

PV verkürzung 1.5: Auch für den PV wird das Signal neben der Streckung auch mit einem Faktor von 1.5 verkürzt. Hier scheint es jedoch zu weniger Phasensprüngen zu kommen. Das Ergebnis ist in Abbildung 19 dargestellt.

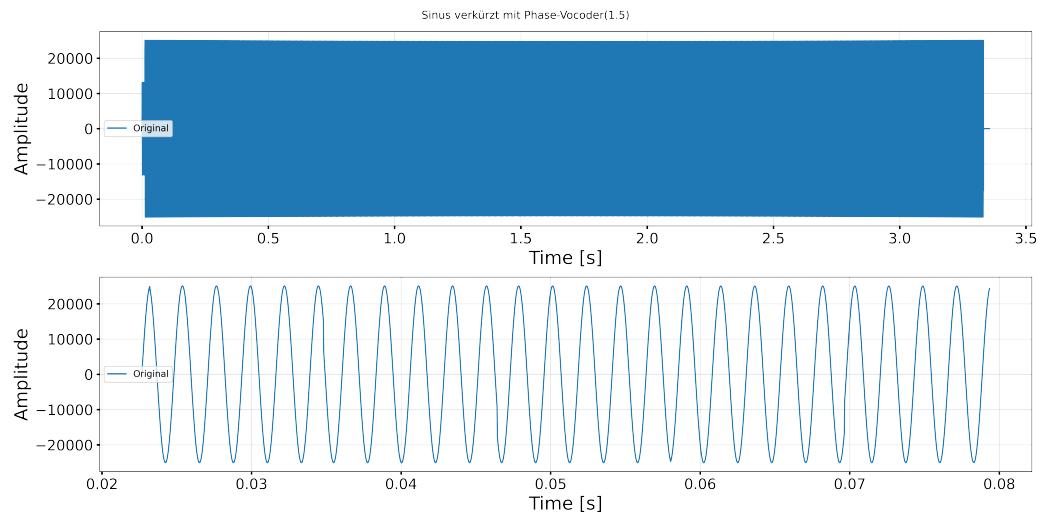


Abbildung 19: Gestrecktes Referenzsignal mit PV

Die Implementierung scheint grundlegend zu funktionieren und die gewünschte Zeitdehnung oder -stauchung zu erreichen. Es sind jedoch ganz klar Artefakte zu erkennen. Manche entstehen aus den Limitierungen des einzelnen Verfahrens. Andere hingegen entstehen durch die Implementierung, wie z.B. das ungewollte Verhalten beim ersten Frame. Bei der Implementierung besteht noch Potential zur Verbesserung für eine robustere Verarbeitung für alle Signale. Ebenfalls wären nach weitere Testfälle sinnvoll, wie die Analyse bei verschiedenen Streckungsfaktoren um potentielle Artefakte zu erkennen. Denn nicht jede Zeitdehnung führt zu der gleichen Ausprägung von Artefakten.

6.3 Ausblick

In diesem Kapitel werden mögliche Schritte zur Verbesserung und Weiterentwicklung der Implementierung aufgezeigt. Dabei werden verschiedene Aspekte betrachtet, die zur Verbesserung der Leistungsfähigkeit und Genauigkeit der Anwendung beitragen können. Dazu gehören die Benutzeroberfläche, die Implementierung von TSM-Verfahren, die Unterstützung verschiedener Formate und vieles mehr.

6 AUSWERTUNG

Benutzeroberfläche: Die derzeitige Benutzeroberfläche ist sehr einfach gehalten und bietet nur grundlegende Funktionen. Um die Benutzerfreundlichkeit zu erhöhen und die Anwendung attraktiver zu gestalten, könnten weitere Funktionen hinzugefügt werden. Dazu gehören beispielsweise die Möglichkeit, verschiedene TSM-Verfahren auszuwählen, die Einstellung von Parametern wie der Streckungsfaktor oder die Wahl des Fensters, die Anzeige von Spektrogrammen oder anderen visuellen Darstellungen des Audiosignals, die Möglichkeit, das Ergebnis zu speichern oder zu exportieren, und vieles mehr. Durch die Implementierung dieser Funktionen könnte die Anwendung erheblich erweitert und verbessert werden. Zudem könnte die Benutzeroberfläche für verschiedene Geräte und Bildschirmgrößen optimiert werden, um eine optimale Benutzererfahrung zu gewährleisten.

TSM-Verfahren: In dieser Arbeit wurden nur zwei TSM-Verfahren implementiert. Es gibt jedoch viele weitere Verfahren, die in der Literatur beschrieben sind und die für verschiedene Anwendungsfälle geeignet sind. Einige dieser Verfahren könnten in zukünftigen Arbeiten implementiert und untersucht werden, um die Leistungsfähigkeit und Genauigkeit der verschiedenen Methoden zu vergleichen. Dazu gehören beispielsweise WSOLA, HPSS und andere Verfahren, die in der Literatur beschrieben sind. Durch den Vergleich verschiedener Verfahren können die Vor- und Nachteile der einzelnen Methoden besser verstanden und bewertet werden.

Formate: Die Implementierung unterstützt nur das WAV Format. Um die Benutzerfreundlichkeit zu erhöhen, sollten auch andere Formate wie MP3 oder FLAC unterstützt werden. Dies würde die Anwendung für eine breitere Benutzerbasis zugänglich machen und die Flexibilität erhöhen.

Testing: Die komplette Software wurde nur rudimentär getestet um die Kernfunktionalität sicherzustellen. Darüber hinaus besteht noch viel Potential die Implementation zu testen. Dazu gehört auch die Ausführung auf verschiedenen Plattformen und Browsern. Ebenfalls sollten die Benutzeroberfläche auf Intuitivität, Benutzerfreundlichkeit und Missbrauch durch Dritte getestet werden.

6.4 Rückblick

Keine praktische Arbeiten verläuft ohne Komplikationen und Rückschritte. Jedoch sind genau diese Misserfolge und Rückschritte diejenigen, die am meisten lehren. In dieser Arbeit wurden viele Herausforderungen gemeistert und viele neue Erkenntnisse gewonnen. Die Implementierung der verschiedenen TSM-Verfahren war eine komplexe Aufgabe, die viel Zeit und Geduld erforderte. Die Auseinandersetzung

6 AUSWERTUNG

mit den theoretischen Grundlagen und die Umsetzung in die Praxis waren eine wertvolle Erfahrung, die das Verständnis für die Audioverarbeitung vertieft hat. Die Analyse und Diskussion der Ergebnisse haben gezeigt, dass die Implementierung grundsätzlich erfolgreich war und die grundlegenden Ziele erreicht wurden. Nichtsdestotrotz bestehn noch viel Potential um die Qualität zu verbessern. Die Auswertung der Artefakte hat gezeigt, dass die verschiedenen TSM-Verfahren unterschiedliche Artefakte erzeugen und dass es wichtig ist, diese zu verstehen und zu minimieren. Insgesamt war die Arbeit eine wertvolle Erfahrung. Sie zeigte eine abgerundete Sicht auf die Audioverarbeitung, Webentwicklung und Projektdokumentation.

Literatur

- [1] Martín Abadi u. a. „TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems“. In: *ArXiv* abs/1603.04467 (2016). URL: <https://api.semanticscholar.org/CorpusID:5707386>.
- [2] Rafael Alves. *WaveFile.js: A JavaScript library for WAV files manipulation*. Accessed: 2024-05-18. 2020. URL: <https://github.com/rochars/wavefile>.
- [3] Tom Bäckström u. a. *Introduction to Speech Processing*. 2. Aufl. 2022. DOI: 10.5281/zenodo.6821775. URL: <https://speechprocessingbook.aalto.fi>.
- [4] Marina Bosi, Richard E. Goldberg und Joan L. Mitchell. „Introduction to Digital Audio Coding and Standards“. In: *J. Electronic Imaging* 13 (2004), S. 83. URL: <https://api.semanticscholar.org/CorpusID:8313468>.
- [5] M.G. Christensen. *Introduction to Audio Processing*. Springer, 2019. ISBN: 9783030117825. URL: <https://books.google.ch/books?id=460fzQEACAAJ>.
- [6] Mozilla Developer Network (MDN) contributors. *Using Blob Objects with Audio Data and Audio Tags in JavaScript*. Accessed: 2024-05-18. 2024. URL: <https://developer.mozilla.org/en-US/docs/Web/API/Blob>.
- [7] James W. Cooley und John W. Tukey. „An algorithm for the machine calculation of complex Fourier series“. In: *Mathematics of Computation* 19 (1965), S. 297–301. URL: <https://api.semanticscholar.org/CorpusID:121744946>.
- [8] IBM Corporation und Microsoft Corporation. *Multimedia Programming Interface and Data Specifications 1.0*. Redmond, WA: Microsoft Press, 1991. URL: <https://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/Docs/riffmci.pdf>.
- [9] Jonathan Driedger und Meinard Müller. „A Review of Time-Scale Modification of Music Signals †“. In: *Applied Sciences* 6 (2016), S. 57. URL: <https://api.semanticscholar.org/CorpusID:14148964>.

LITERATUR

- [10] T. Frey, N. Fliege und M. Bossert. *Signal- und Systemtheorie*. Informationstechnik. Vieweg+Teubner Verlag, 2013. ISBN: 9783322967275. URL: https://books.google.ch/books?id=_zoFBgAAQBAJ.
- [11] Tailwind Labs. *Tailwind CSS*. Accessed: 2024-05-18. 2022. URL: <https://tailwindcss.com/>.
- [12] Victor Lazzarini. „Computer Music Instruments II“. In: *Cambridge International Law Journal*. 2019. URL: <https://api.semanticscholar.org/CorpusID:85542603>.
- [13] Mozilla Developer Network (MDN) contributors. *Web Audio API*. Accessed: 2024-05-18. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API.
- [14] Mozilla Developer Network (MDN) contributors. *Web Workers API*. Accessed: 2024-05-30. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API.
- [15] React contributors. *React*. Accessed: 2024-05-18. 2022. URL: <https://reactjs.org/>.
- [16] TypeScript contributors. *TypeScript*. Accessed: 2024-05-18. 2022. URL: <https://www.typescriptlang.org/>.
- [17] Martin Werner. *Digitale Signalverarbeitung mit MATLAB®*. Springer Vieweg, 2009. ISBN: 978-3-658-21724-8. URL: <https://link.springer.com/book/10.1007/978-3-8348-9243-0>.