Dokumentation der Projektarbeit

im Fach

Digitale Signalverarbeitung & Mustererkennung

Teil II - Digitale Signalverarbeitung

vorgelegt von

Maximilian Gayer

Matrikel-Nr.: 204130

Studiengang: MAS

und

Markus Ullenbruch

Matrikel-Nr.: 204185

Studiengang: MAS

WS 2019/2020

Inhaltsverzeichnis

1 Teil II - Digitale Signalverarbeitung

1.1 Einleitung und Aufgabenstellung

Im zweiten Teil der Vorlesung *Digitale Signalverarbeitung & Mustererkennung* wurde der Teil der *digitalen Signalverarbeitung* bearbeitet. Als Programmiersprache wurde *Python* gewählt und es wurde mit den Packages *numpy*, *scipy* für mathematische Operationen und *librosa* zum importieren für .mp3 und .wav Audio Dateien gearbeitet. Als Entwicklungsumgebung wurde *Jupyter Notebook* als interaktive web-basierte Umgebung gewählt mit der Jupyter Notebook Dokumente mit der Endung .ipynb erstellt werden. Diese Wahl wurde aufgrund der hervorragenden Eigenschaften der Jupyter Notebooks bezüglich der Dokumentation, Lesbarkeit, Übersicht, Tests und Visualisierung getroffen. In einem Jupyter Notebook ist jeweils die Implementierung im ersten Block und das Testen der Implementierung im darauffolgenden Block im selben Dokument enthalten. Zusätzlich werden pdf Dateien der Dokumente mitgesendet.

Es wurden zuerst verschiedene Algorithmen rund um die in der Signalverarbeitung wichtige diskrete Fourier-Transformation und andere mathematische Operationen als separate Dokumente implementiert und diese jeweils mit den Implementierungen der *numpy*-Packages verglichen. Folgende Implementierungen wurden programmiert: Diskrete Faltung im Zeitbereich, Diskrete Fourier Transformation (DFT) und deren Inverse Transformation (IDFT), Fast-Fourier-Transformation (FFT) rekursiv, Fast-Fourier-Transformation (FFT) iterativ, schnelle Faltung mit FFT und einen Tiefpassfilter im Zeitbereich.

Anschließend wurden 2 Audiosignale von Musiktiteln einer Modulation unterzogen um so ein einziges Träger-Signal zu erhalten, welches die Informationen der Audio Signale enthält. Dadurch wird in der Praxis eine hochfrequente Übertragung der niederfrequenten Nutzsignale (hier Audiodateien) möglich. Die zu sendenden Signale belegen im Bereich der Trägerfrequenz eine vom Nutzsignal abhängige Bandbreite, daher ist es wichtig, dass die Nutzsignale vor der Modulation tiefpassgefiltert werden um so bandbegrenzte Signale zu erhalten, deren Frequenzbereiche sich im Trägermedium nicht überlappen. Dieses Trägersignal wurde anschließend einer Demodulation mit anschließender Tiefpassfilterung unterzogen um so die zwei ursprünglichen Audio Signale aus dem Trägersignal zu rekonstruieren. Hierbei soll die im Tiefpassfilter notwendige Faltung im Zeitbereich ersetzt werden durch die schnelle Faltung mit FFT, da diese deutliche Vorteile bezüglich der Rechenzeit und Komplexität aufweist.

1.2 Faltung im Zeitbereich

Um die selbst geschriebene Funktion der diskreten Faltung zu testen wurden zwei Testsignale generiert, die miteinander gefaltet werden. Diese diskreten Testsignale sind in Abbildung 1.1 dargestellt. Die Samplingrate beträgt $f_s = 20\text{Hz}$. In Abbildung 1.2 ist das Ergebnis der

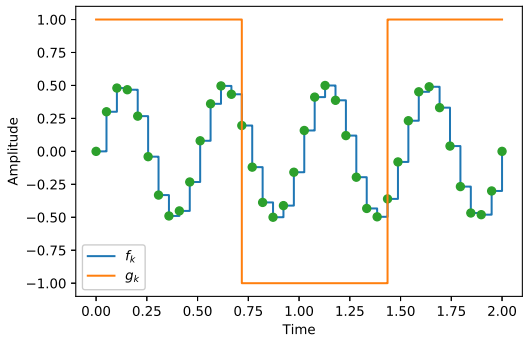

Abbildung 1.1: Testsignale mit denen Faltung durchgeführt wird.

Faltung mit dem selbst geschriebenen Code und der Faltung von numpy gegenübergestellt. In Abbildung 1.3 ist die Differenz der Ergebnisse dargestellt.

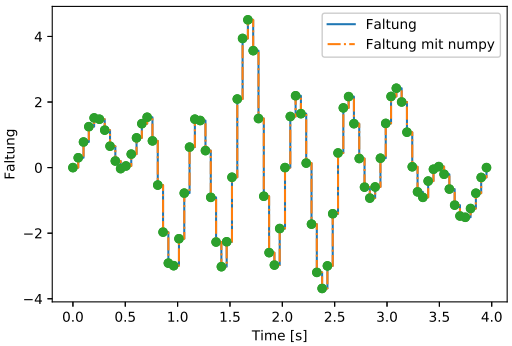

Abbildung 1.2: Ergebnis Faltung

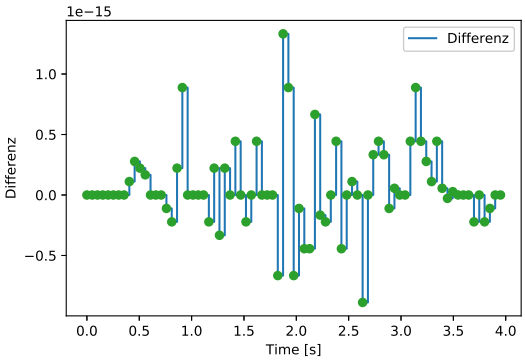

Abbildung 1.3: Differenz der Ergebnisse

Die Ergebnisse unterscheiden lediglich im $\Delta = 10^{-15}$ Bereich voneinander was auf numerische Rundungsfehler schließen lässt. Die programmierte diskrete Faltung funktioniert somit einwandfrei.

1.3 Tiefpass-Filter im Zeitbereich

Der Tiefpass-Filter im Zeitbereich wurde wie im Skript beschrieben implementiert. Um das Ergebnis zu verbessern wurde das Hammingfenster w_k verwendet und mit den Filterkoeffizienten x_k multipliziert. Der Tiefpass läuft mit der selbst geschriebenen schnellen Faltung und der rekursiven FFT.

Test mit Audio Signalen

Um den Tiefpassfilter zu testen wurde eine .wav Dateie mit dem Python Package *librosa* geladen, ein Ausschnitt mit definierter Länge herausgeschnitten (ca 6s) und die im Signal enthaltenen Frequenzen über der Zeit farbcodiert geplottet. Der Amplitudenverlauf des Si-

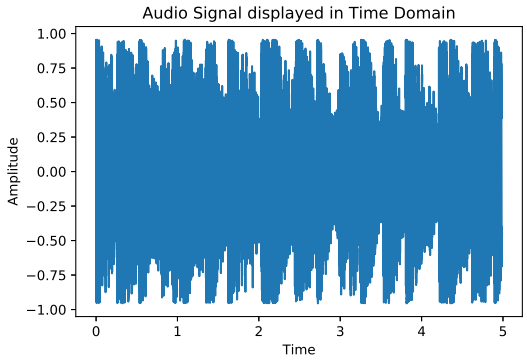

Abbildung 1.4: Amplitude des 6 Sekunden Auschnitts des verwendeten Audio Signals.

gnals ist in Abbildung 1.4 darsgestellt und das Spektogramm ist in Abbildung 1.5 dargestellt. Der Audio Ausschnitt der .wav Datei hat 140.000 Samples und eine Abtastrate von $f_s = 44,1$kHz. Der Tiefpass-Filter wurde mit einer Cutoff-Frequenz von $f_c = 4$ kHz und einer Ordnung von $n = 800$ auf das Audio Signal angewendet. Das Ergebnis dieser Filterung ist in 1.6 als Spektogramm zu sehen. Die Frequenzen oberhalb von 4 kHz wurden wie erwartet rausgefiltert. Das Spektogramm ist in der Dezibel Skala dargestellt. Die Audio Dateien können mit IPython.display.Audio(X) im Jupyter Notebook direkt angehört werden. Somit kann der Effekt der Filterung direkt hörbar gemacht werden. Wie erwartet fehlen die hohen Frequenzen durch die Filterung. Werden noch tiefere Cutoff-Frequenzen gewählt, so hört sich das Audiosignal nach der Filterung immer "tieferünd basslastiger an.

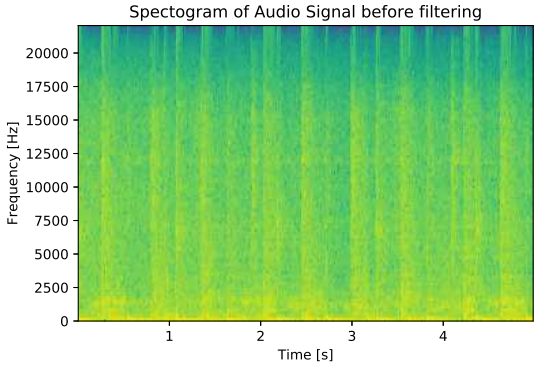

Abbildung 1.5: Audio Signal vor der Filterung.

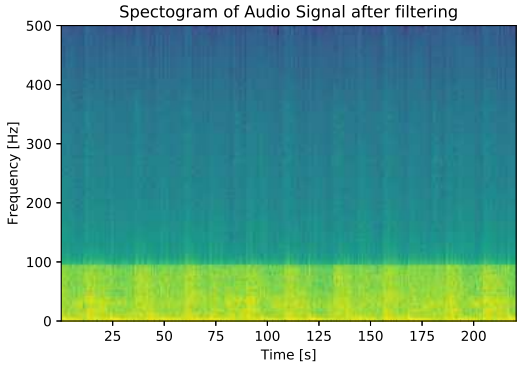

Abbildung 1.6: Audio Signal nach der Tiefpass-Filterung mit $f_c = 4$kHz und $n = 800$.

Test mit harmonischen Signalen

Um die Zeitverzögerung, die durch das Tiefpassfilter entsteht zu verdeutlichen wurde die Filterung noch an einem Signal bestehend aus zwei harmonischen Signalen gleicher Amplitude mit den Frequenzen $f_1 = 2\text{Hz}$ und $f_2 = 10\text{Hz}$ durchgeführt. Das Signal mit seinen Anteilen ist in Abbildung 1.7 zu sehen. In Abbildung 1.8 ist das gefilterte Signal zu sehen. Es ist eine deutliche zeitliche Verzögerung von $\Delta t = 0,45\,\text{s}$ zu erkennen. Als Ordnung des Filters wurde $n = 1200$ gewählt. Weitere Parameter sind $f_c = 7\,\text{Hz}$ und $f_s = 1\,\text{kHz}$. Die

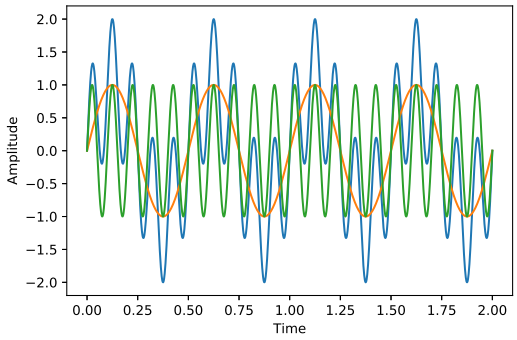

Abbildung 1.7: Signal bestehend aus harmonischen mit den Amplituden $A_{1,2} = 1$ und den Frequenzen $f_1 = 2\text{Hz}$ und $f_2 = 10\text{Hz}$.

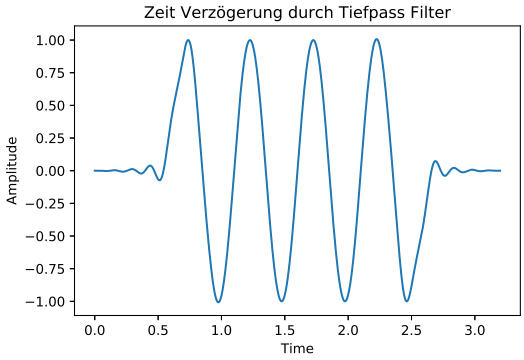

Abbildung 1.8: Gefiltertes Signal mit $f_c = 7\text{Hz}$.

Filterung des Signals wird mit zunehmender Ordnung n des Filters genauer, jedoch erhöht sich dadurch auch proportional die zeitliche Verzögerung Δt.

1.4 Diskrete Fourier Transformation (DFT)

Die Diskrete Fourier-Transformation (DFT) und ihre Inverse diskrete Fourier-Transformation (IDFT) wurden in DFT_IDFT.ipynb implementiert. Die Implementierung funktioniert sehr gut.

Die berechneten Filterkoeffizienten wurden mit der DFT von Python/ numpy verglichen. Die Differenzen des Realteils und der Imaginärteile der eigen geschriebenen Funktion DFT(Data) und der des numpy Package sind in Abbildung 1.9 und 1.10 dargestellt. In Abbildung 1.11 ist das ursprüngliche Testsignal und das rekonstruierte Signal IDFT(DFT(Data)) dargestellt. Die Signale unterscheiden sich nicht voneinander. Die Differenzen befinden sich im 10^{-12} Bereich.

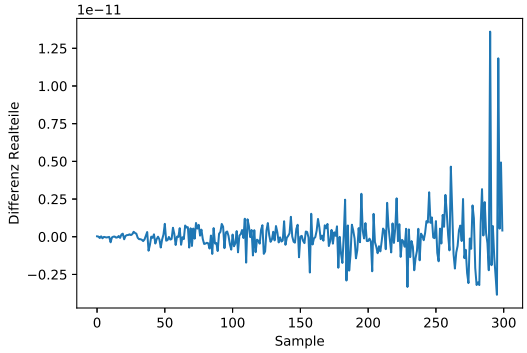

Abbildung 1.9

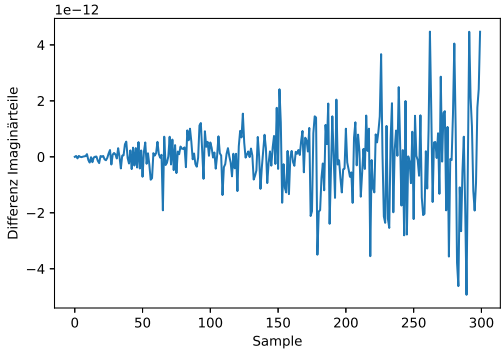

Abbildung 1.10

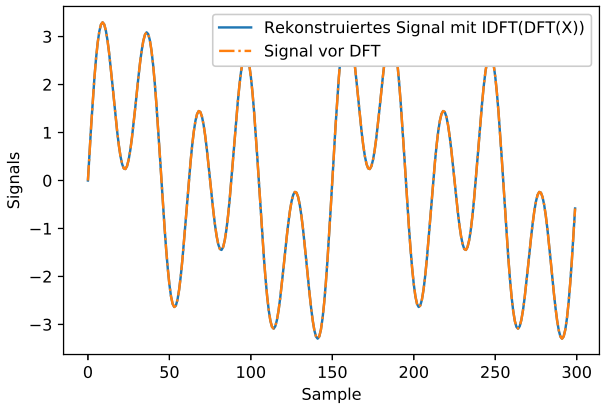

Abbildung 1.11

1.5 Fast Fourier Transformation (FFT) rekursiv

Für die rekursive FFT muss ein Eingangssignal gewählt werden deren Anzahl n an Samples eine Zweierpotenz darstellt $n = 2^N, N \in \mathbb{N}$. Der Algorithmus der rekursiven FFT wird mit einem Signal bestehend aus der Addition von 3 harmonischen Sinus Termen, siehe Abbildung 1.12, überprüft. Das Ergebnis wurde in Imaginär- und Realteil aufgeteilt und mit der python-integrierten FFT numpy.fft.fft(Samples) verglichen. Dieser Vergleich ist in den Abbildungen 1.13 und 1.14 zu sehen.
Die Ergebnisse liegen sehr gut übereinander. Nimmt man die Differenz der Werte so sieht man, dass die Differenzen der Ergebnisse im Bereich von $\Delta = 10^{-14}$ liegen.

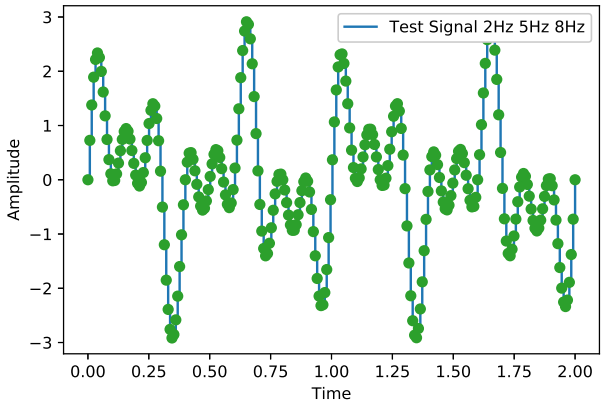

Abbildung 1.12: Testsignal mit dem FFT durchgeführt und verglichen wird.

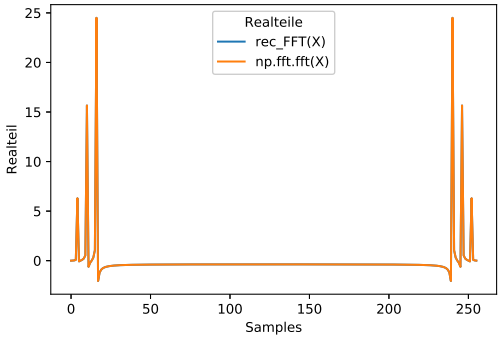

Abbildung 1.13: Realteil der Fourierkoeffizienten.

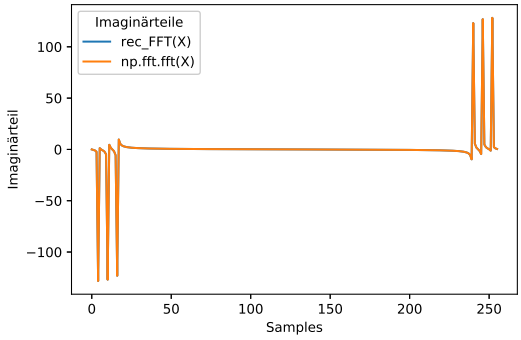

Abbildung 1.14: Imaginärteil der Fourierkoeffizienten.

1.6 Fast Fourier Transformation (FFT) iterativ

1.7 Schnelle Faltung mit FFT

Das richtige Funktionieren der schnellen Faltung mit FFT wurde mit denselben Signalen getestet wie bei der Faltung im Zeitbereich. Das Ergebnis ist in Abbildung 1.15 zu sehen. Die Ergebnisse der schnellen Faltung stimmen mit denen der diskreten Faltung im Zeitbereich überein.

Die Rechenzeit kann mit dem Jupyter Notebook modul *%%timeint* gemessen werden. Bei zwei anderen Testsignalen (harmonische Signale) mit jeweils 6.800 Samples beträgt die Rechenzeit bei der Faltung beider Signale im Zeitbereich 93s und bei der schnellen Faltung mit FFT $15,2$ms. Die schnelle Faltung rechnet bei diesem Beispiel somit 6118-mal schneller als die Faltung im Zeitbereich. Dies ist ein enormer Unterschied! In Abbildung 1.16 ist die Rechenzeit der schnellen Faltung mit FFT und der Faltung im Zeitbereich für jeweils 2 Inputsignale, bei denen die Signallänge variiert wurde, gegenübergestellt. Es wurden die selbst

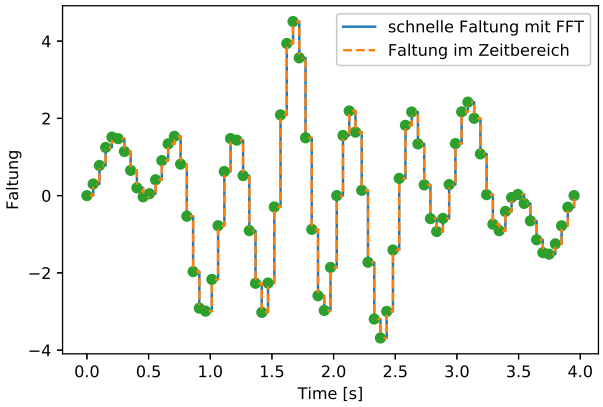

Abbildung 1.15: Ergebnisse der schnellen Faltung mit FFT und der Faltung im Zeitbereich.

implementierten Funktionen verwendet. Bei zunehmender Signallänge/Samples der zu faltenden Signale steigt die Rechenzeit der Faltung im Zeitbereich enorm an während sie bei der schnellen Faltung um Zehnerpotenzen langsamer ansteigt. Die unterschiedliche Komplexität der beiden Verfahren und die enorme Bedeutung der schnellen Faltung wird hier ersichtlich.

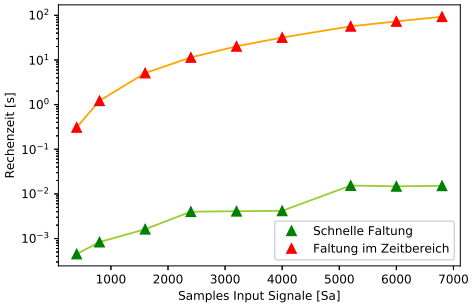

Abbildung 1.16: Rechenzeit der schnellen Faltung und der Faltung im Zeitbereich im Vergleich.

1.8 Modulation & Demodulation

Es wurde ein Ausschnitt eines Songs der Red Hot Chili Peppers benutzt um diesen mit einer Modulationsfrequenz von $f_{mod} = 12$kHz zu modulieren. Das Ergebnis dieser Modulation ist in Abbildung 1.17 zu sehen. In Abbildung 1.18 sieht man das Frequenzspektrum des Original Ausschnitts des verwendeten Songs wie er vor der Modulation war. In Abbildung 1.19 ist das durch Demodulation rekonstruierte Signal zu sehen. Auffallend ist sofort, dass Störfrequenzen im demodulierten Signal mit hoher Intensität vorhanden sind, die im Ursprungssignal vor der Modulation nicht in dieser Intensität bzw. kaum vorhanden waren. Das demodulierte Signal hört sich jedoch qualitativ hervorragend an und das obwohl kein Tiefpassfilter benutzt

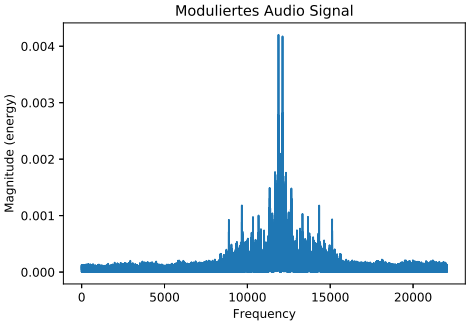

Abbildung 1.17: Moduliertes Signal mit $f_{mod} = 12$kHz.

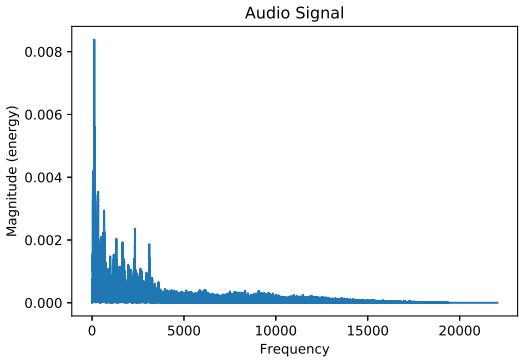

Abbildung 1.18: Ausgangssignal für die Modulation.

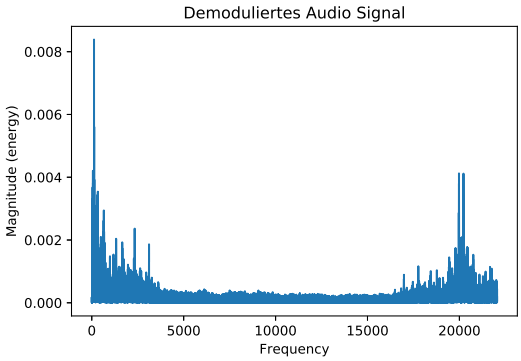

Abbildung 1.19: Durch Demodulation rekonstruiertes Signal.

wird. Im rekonstruierten Signal sind zwar Störfrequenzen enthalten, die im ursprünglichen Signal nicht enthalten sind, diese sind jedoch oberhalb von 16kHz, mit der größten Intensität um 20kHz herum, und können so von erwachsenen Menschen nicht mehr bzw. kaum wahrgenommen werden. Deshalb hört sich das Signal hervorragend an, obwohl der Blick aufs Frequenzspektrum auf den ersten Blick was anderes vermuten lassen würde, wenn man die maximale Frequenz, die Menschen hören können, nicht kennen würde. Wird die Modulations/Demodulationsfrequenz niedriger gewählt, so bilden sich die Störfrequenzen im rekonstruierten Signal in einem niedrigeren Frequenz-Bereich aus und können wahrgenommen werden. Der Hörgenuss ist dann nicht zufriedenstellend und man müsste einen Tiefpassfilter bemühen, der diese höheren Frequenzen rausfiltert.

1.9 Gesamtsystem

Für das Gesamtsystem wurden zwei .mp3 Songs verwendet und in das Jupyter-Notebook geladen. Diese zwei Audio Dateien sollen auf ein Trägersignal moduliert und anschließend demoduliert werden um die zwei Signale wieder zu rekonstruieren. Das Ergebnis kann im Jupyter-Notebook direkt angehört und abgespielt werden. So lassen sich die Effekte der Mo-

dulationsfrequenzen $f_{mod,1}$, $f_{mod,2}$, der Filterordnung n und der Cutoff-Frequenz f_c direkt hör- und visualisierbar machen.

Zuerst wurden die zwei verwendeten Audio Signale, in Abbildung 1.20 zu sehen (Länge ca. 6s), jeweils tiefpassgefiltert um bandbegrenzte Signale zu erhalten. Diese Filterung wurde mit einer Cutoff-Frequenz von $f_c = 3,9$kHz durchgeführt. Die Audiosignale besitzen eine Abtastfrequenz von $f_s = 40,1$kHz. Anschließend sind die beiden Signale jeweils mit

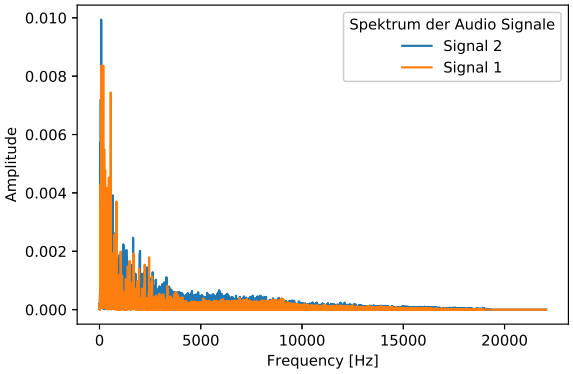

Abbildung 1.20: Amplitudenspektrum der Original Audiosignale.

einer passenden Modulationsfrequenz zu modulieren. Diese wurde zuerst so gewählt, dass sich die modulierten Signale im Frequenzbereich gegenseitig nicht überlappen, dass es zu keinen Störungen und Beeinflussungen bei der späteren Rekonstruktion kommt. Es wurde $f_{mod,1} = 5$kHz und $f_{mod,2} = 15$kHz gewählt. Nachdem die Signale moduliert wurden lässt sich das Spektrum beider Signale plotten (Abbildung 1.21) und es kann überprüft werden ob eine Überlappung im Frequenzbereich stattfindet oder nicht. Es findet keine Überlap-

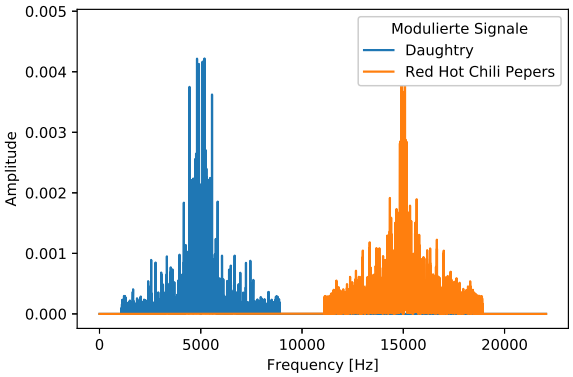

Abbildung 1.21: Amplitudenspektrum der modulierten Signale. Die Frequenzbänder beider Signale überlappen sich nicht. Die Modulationsfrequenzen sind $f_{mod,1} = 5$kHz und $f_{mod,2} = 15$kHz. Die Cutoff-Frequenz der vorangegangenen Tiefpassfilterung beträgt $f_c = 3,9$kHz.

pung der Bereiche statt. Wählt man $f_{mod,1} = 8\text{kHz}$ und $f_{mod,2} = 12\text{kHz}$ so liegen diese Modulationsfrequenzen bereits zu nah aneinander und die Frequenzbänder der beiden modulierten Signale überlappen sich wie in Abbildung 1.22 zu sehen ist. Anschließend werden

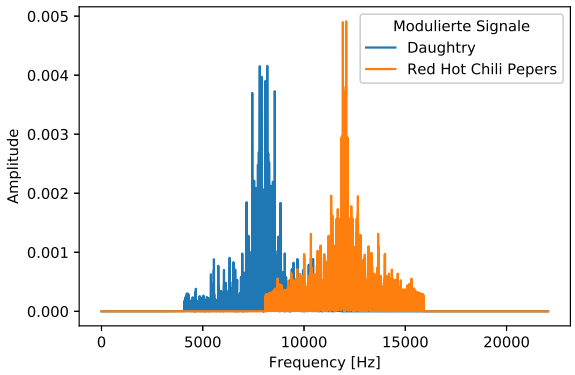

Abbildung 1.22: Überlappender Frequenzbereich der modulierten Signale: Die Modulationsfrequenzen sind mit $f_{mod,1} = 8\text{kHz}$ und $f_{mod,2} = 15\text{kHz}$ zu nah aneinander gewählt.

die modulierten Signale zu einem einzigen Signal (Trägersignal) addiert und dieses anschließend demoduliert. Jedes demodulierte Signal wird dann einer Tiefpassfilterung, mit derselben Cutoff-Frequenz wie die Eingangssignale vor der Modulation, unterzogen. Abhängig von der Wahl der Cutoff-Frequenz dieser Filterung und den Modulationsfrequenzen treten unterschiedliche Qualitäten der Signalrekonstruktion auf.

Auswertung der rekonstruierten Signale

In Abbildung 1.23 und 1.24 ist das Amplitudenspektrum der rekonstruierten (demodulierten und tiefpassgefilterten) Signale zu sehen, einmal für nicht überlappende und einmal für überlappende Frequenzbänder der modulierten Signale. Hört man sich die rekonstruierten Audio-Dateien in Abbildung 1.23 an, so ist die Signalrekonstruktion ziemlich gut und die Audio Dateien sind sehr klar und deutlich zu erkennen und in guter Qualität anzuhören.
Benutzt man allerdings Modulationsfrequenzen, sodass sich die Frequenzbereiche der modulierten Signale überlappen, so sind bei beiden rekonstruierten Signalen in Abbildung 1.24 höhere Frequenzanteile zu erkennen, die in der ursprünglichen Audiodatei nicht enthalten waren. Diese Störungsfrequenzen hört man beim Anhören der Dateien auch sehr gut als hohes, periodisches, metallisches "Piepenünd "Klirren"heraus. Dies stört den Hörgenuss und die Soundqualität enorm. Diese Störung kann man somit nicht nur am Amplitudenspektrum erkennen sondern ist auch sehr gut hörbar.

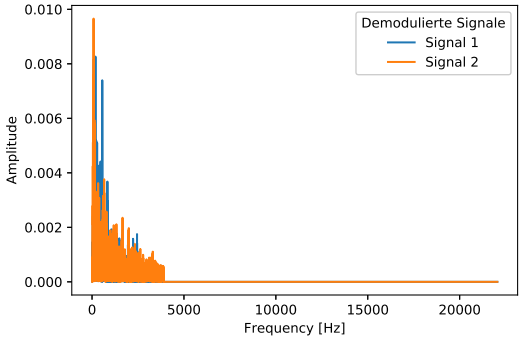

Abbildung 1.23: Gutes Ergebnis aufgrund Frequenzbänder der modulierten Signale, die sich nicht überlappen.

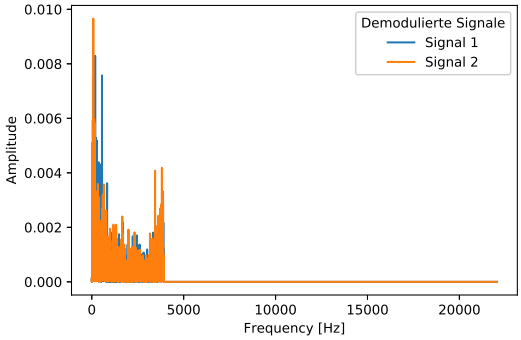

Abbildung 1.24: Schlechtes Ergebnis aufgrund überlappende Frequenzbänder.

Die Berechnungen wurden mit eher hohen Filterordnungen (≈ 1024) gemacht. Wählt man sehr kleine Filterordnungen wie zB. $n = 8, 16, 32$ so cuttet der Filter nicht mehr "hartßondern cuttet "weich"was dazu führt, dass sich die Frequenzbänder der modulierten Signale wieder überlappen können und sich dadurch die Soundqualität der rekonstruierten Signale wieder verschlechtert.