Duale Hochschule Sachsen

– Studienrichtung Informationstechnik –

Prototypische Umsetzung einer Anwendung zur Übertragung von Bilddaten durch akustische Signale

Belegarbeit zur Praxisphase des 3. Semesters

vorgelegt am 28. Februar 2025 von

Benjamin Majta

Matrikelnummer: 3005323

Ausbildender Praxispartner:	InQu Solutions GmbH Suhausweg 3, 01099 Dresden
Begutachtung Praxispartner:	Tobias Heinicke
Begutachtung BA Sachsen:	Prof. Dr.-Ing. Tenshi Hara

Zusammenfassung

Diese Arbeit behandelt die Konzeption und prototypische Umsetzung einer Anwendung zur Übertragung von Bilddaten mithilfe von akustischen Signalen. Ziel ist es, eine unübliche Lösung für die Datenübertragung zu implementieren, die auf Tönen basiert und sich damit von traditionellen Übertragungsmethoden auf Grundlage von elektromagnetischen Wellen oder optischen Signalen abhebt. Umfasst werden die theoretische Analyse geeigneter Kodierungs- und Modulationsverfahren, die Implementierung einer Softwarelösung sowie die Evaluierung der Übertragung hinsichtlich Geschwindigkeit, Zuverlässigkeit und Störanfälligkeit.

Inhaltsverzeichnis

Abbildungen

Quellkode

Teil I

Belegarbeit

Kapitel 1

Einleitung

1.1 Unternehmensvorstellung

Die InQu Solutions GmbH ist ein expandierendes Tochterunternehmen der Rosenxt-Gruppe mit Sitz in Dresden. Sie ist ein führender Anbieter von digitalen Lösungen für die Industrie. Im Vordergrund steht diesbezüglich die Entwicklung eines Manufacturing Execution Systems (MES), welches Unternehmen dabei hilft, Fertigungsprozesse zu optimieren, Effektivität und Qualität zu steigern und die Wettbewerbsfähigkeit zu verbessern.

1.2 Zielsetzung

Ziel dieser Arbeit ist die Entwicklung einer prototypischen Anwendung zur Übertragung von Bilddaten über akustische Signale. Im Fokus steht dabei die präzise Rekonstruktion der in einem Audiosignal kodierten Daten mithilfe der Fourier-Transformation. Es soll gezeigt werden, wie Daten in Tonsignale kodiert, übertragen und anschließend wiederhergestellt werden können. Neben der praktischen Umsetzung werden die zugrunde liegenden mathematischen und signaltheoretischen Prinzipien analysiert und angewendet.

1.3 Relevanz des Themas

Die Datenübertragung über akustische Signale ist ein innovativer Ansatz, der in Szenarien ohne klassische Netzwerkinfrastruktur oder bei besonderen technischen Anforderungen von Bedeutung sein kann. Ein Beispiel wäre dafür die Kommunikation von Geräten unter Wasser, wo akustische statt elektromagnetische Signale verwendet werden. Dieses Thema vereint moderne Techniken der Signalverarbeitung und und eine unübliche Form der Datenübertragung und bietet eine praxisnahe Anwendung signaltheoretischer Konzepte.

Kapitel 2

Theoretische Grundlagen

2.1 Grundlagen der Audiosignale

Ein Signal ist der Verlauf einer messbaren physikalischen Größe. Wenn dieser Verlauf in Abhängigkeit von der Zeit betrachtet wird, spricht man von einem Zeitsignal.[1] Audiosignale sind eine spezielle Form von Zeitsignalen, da sie zeitliche Veränderungen von Schwingungen in einem Medium darstellen.

In der Audiotechnik unterscheidet man zwischen analogen und digitalen Audiosignalen.

- Analoge Audiosignale sind kontinuierlich, das heißt, sie können alle Werte innerhalb eines bestimmten Bereichs[2] annehmen[1]. Sie repräsentieren also Schallwellen in einer fließenden Form, so wie sie in der Natur vorkommen.
- Digitale Audiosignale hingegen bestehen aus diskreten Werten. Sie entstehen durch die Abtastung eines analogen Signals, bei der in regelmäßigen Zeitintervallen Messwerte erfasst und gespeichert werden [1]. Ein gängiger Standard für digitale Audiosignale ist eine Abtastrate von 44.100 Werten pro Sekunde (44,1 kHz) [Wik24a].

Um ein digitales Signal wieder in eine analoge Form zu überführen, erfolgt eine Interpolation der gespeicherten Abtastwerte. Dabei werden die fehlenden Werte zwischen den diskreten Punkten rekonstruiert, sodass ein kontinuierlicher Verlauf entsteht, der dem ursprünglichen analogen Signal möglichst nahe kommt. In der Praxis gibt es dafür sogenannte Audio-Konverter oder Wandler, die ein digitales Audiosignal analog ausgeben können [Teu24].

2.2 Informationsübertragung durch Audiosignale

Die Übertragung von Daten mithilfe von Audio basiert auf der Nutzung dieser akustischen Signale zur Kodierung von Informationen. Töne sind Schallwellen, die sich als mechanische Schwingungen in einem Medium (z.B. Luft) ausbreiten. Ein Mikrofon nimmt diese Schwingungen auf, in dem die Schallwellen Spannungsänderungen verursachen und somit repräsentative elektrische Signale erzeugen. Diese Signale können anschließend abgetastet und als diskrete Werte gespeichert (digitalisiert) werden[3].

Zur Datenübertragung müssen die Schallwellen so gestaltet werden, dass sie spezifische Inhalte repräsentieren. Dazu werden beispielsweise gezielte Modulationen bestimmter akustischer Parameter wie Frequenz, Amplitude oder Dauer eines Tons angewendet.

[1] [Mer23] S. 2 - 4.
[2] unendlich viele Werte zwischen einem vom Signal gegebenen Minimalwert und Maximalwert
[3] [Wei08] S. 787.

Für diese Arbeit relevant ist vor allem die Technik des Frequency Shift Keying (FSK). Hierbei werden verschiedene Frequenzen verwendet, um unterschiedliche Zustände darzustellen[4]. Zum Beispiel könnte eine niedrige Frequenz eine 0 und eine höhere eine 1 symbolisieren.

Ein weiteres Modulationsverfahren ist außerdem das Amplitude Shift Keying (ASK), bei dem die Amplitude verändert wird[4]. Ein einfaches Beispiel für solch eine Informationübertragung ist der Morsecode, bei dem unterschiedliche Anordnungen kurzer und langer Tonsignale Buchstaben und Zeichen repräsentieren [Wik25].

Zusätzlich gibt es noch das Phase Shift Keying (PSK), bei dem die Phase eines Trägersignals gezielt verschoben wird[4]. Letzteres setzt allerdings durch seine geringen Unterschiede eine sehr genaue Messung voraus, wodurch es hauptsächlich bei Technologien wie WLAN, RFID oder Bluetooth verwendet wird und bei akustischen Signalen keine Anwendung findet.

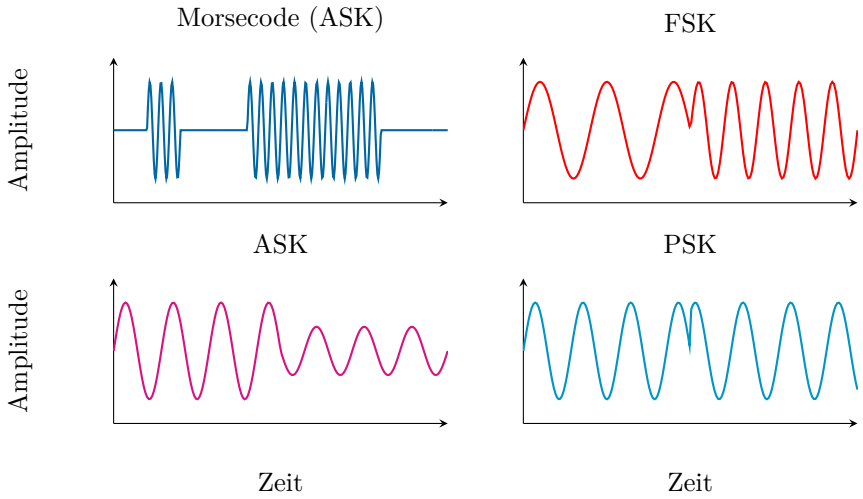


Abbildung 2.2.1: Darstellung verschiedener Modulationstechniken

Die empfangende Seite muss wiederum in der Lage sein, diese Signale zu verarbeiten und zu interpretieren. Hierbei werden die eingebetteten Informationen durch Signalverarbeitungstechniken wie der Fourier-Transformation extrahiert, analysiert und rekonstruiert.

2.3 Fourier-Transformation

Die Fourier-Transformation ist ein mathematisches Verfahren, das ein zeitabhängiges Signal in seine Frequenzkomponenten zerlegt. Anstatt das Signal im Zeitbereich zu betrachten, wird es in den Frequenzbereich überführt. Dadurch lässt sich die spektrale Zusammensetzung eines Signals analysieren[5].

In der Praxis bedeutet dies, dass sich mit der Fourier-Transformation die dominanten Frequenzen eines Zeitsignals identifizieren lassen. Dies ist besonders nützlich für die Verarbeitung von Audiosignalen, da man auf diese Weise effizient Frequenzen erkennen und gezielt modifizieren oder entfernen kann. Diese Methode wird beispielsweise zur Rauschunterdrückung oder Klangverbesserung eingesetzt.

[4] [Rud06] S. 14.
[5] [Mer23] S. 64 - 65.

2.3.1 Spektrale Zusammensetzung eines Signals

Die spektrale Zusammensetzung eines Signals beschreibt alle Frequenzen, aus denen das Signal besteht. Während einfache Signale, wie ein Sinuston, nur eine Frequenz enthalten, setzen sich komplexe Signale, wie Sprache oder Musik, aus vielen verschiedenen Frequenzanteilen zusammen (siehe 2.3.1).

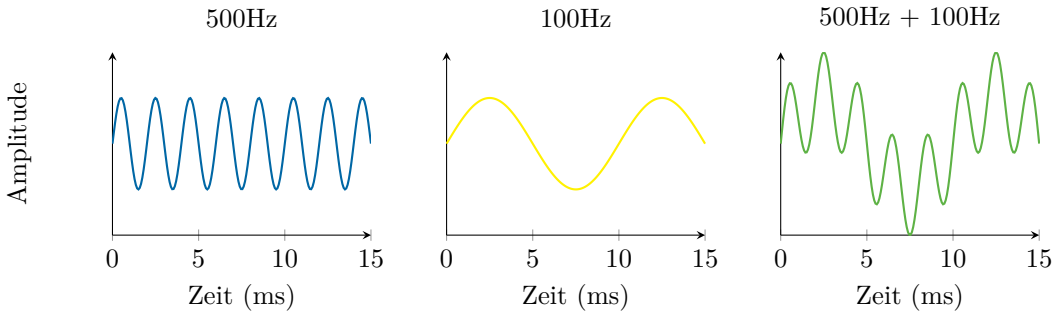


Abbildung 2.3.1: Addition 2 einfacher Signale zu einem komplexen Signal.

2.3.2 Diskrete Fourier-Transformation (DFT)

Während die klassische Fourier-Transformation kontinuierliche Signale verarbeitet, wird die Diskrete Fourier-Transformation DFT speziell für digitale Signale genutzt, die aus einer endlichen Anzahl von Abtastwerten bestehen[6].

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}kn} \tag{2.3.1}$$

- X_k : Wert der Fouriertransformation für eine Frequenz k
- x_n : Wert der Zeitstichproben
- N : Gesamtanzahl der Zeitstichproben
- n : Index der Zeitstichproben
- k : Untersuchte Frequenz

Die Grundlage der DFT basiert auf einer speziellen Darstellung der diskreten Signalpunkte in der komplexen Zahlebene. Dabei wird jeder Punkt des Signals durch eine komplexe Zahl repräsentiert.

- x_n bestimmt den Betrag der komplexen Zahl und entspricht somit der Amplitude des Signals an diesem Punkt.
- $e^{-j\frac{2\pi}{N}kn}$ gibt die Position auf der komplexen Ebene vor, wobei der Winkel von der untersuchten Frequenz k abhängt.

Für jede Frequenz im betrachteten Bereich werden die komplexen Werte der Signalpunkte aufsummiert. Der Algorithmus überprüft, wie sich die Werte in der komplexen Ebene verteilen. Ist der Mittelwert X_k der Summation weit vom Ursprung entfernt, so bedeutet dies, dass diese Frequenz im ursprünglichen Signal besonders stark vertreten ist.

In Abbildung 2.3.2 wird dies anhand eines Signals mit einer 3-Hz-Frequenz visualisiert. Die Berechnung der DFT für verschiedene Frequenzen zeigt, dass nur bei der Analyse mit $k = 3$ eine signifikante Abweichung des Werts X_k vom Ursprung auftritt. Dies bestätigt, dass die Frequenz von 3 Hz im ursprünglichen Signal dominant ist.

[6] [Mer23] S. 132.

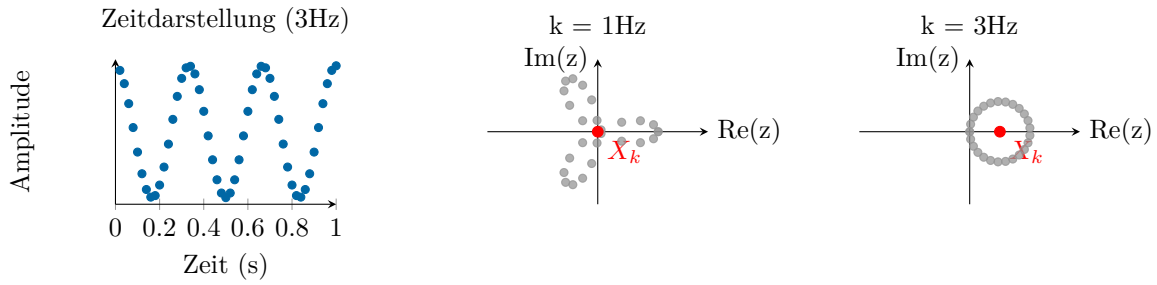


Abbildung 2.3.2: Frequenzanalyse eines Signals mit einer 3Hz Frequenz

2.3.3 Schnelle Fourier-Transformation (FFT)

Die Schnelle Fourier-Transformation (FFT) ist ein optimierter Algorithmus zur Berechnung der DFT[7]. Die klassische DFT ist stets mit einer hoher Rechenkomplexität von $O(N^2)$ verbunden, da für jede mögliche Frequenz alle Werte berechnet und aufsummiert werden müssen. Somit wird diese Methode vor allem für große Datenmengen stark ineffizient. Die FFT reduziert diesen Rechenaufwand erheblich und ermöglicht somit eine schnellere Verarbeitung von Signalen.

Der heutzutage bekannteste FFT-Algorithmus wurde 1965 von James W. Cooley und John W. Tukey veröffentlicht und revolutionierte die digitale Signalverarbeitung, insbesondere in der Audio- und Bildverarbeitung, Telekommunikation und Datenanalyse [Wik24b].

Die Effizienzsteigerung der FFT beruht auf einer rekursiven Zerlegung der DFT, sodass nur kleine DFTs berechnet werden müssen. Da gibt es zum Beispiel die Radix-2-Decimation-in-Time-FFT[7], die für Signallängen $N = 2^k$ optimiert ist. Das Eingangssignal wird hierbei in 2 Teilfolgen, eine mit geraden Indizes und eine mit ungeraden, zerlegt. Die DFT der Gesamtfolge kann dann durch Kombination der DFTs der beiden Teilfolgen berechnet werden. Diese Zerlegung passiert solange, bis nur noch 2 Punkte DFTs verbleiben und diese sind schnell zu berechnen.

Anders ausgedrückt wird dadurch die ursprüngliche DFT-Matrix in eine Produkt aus spärlichen Matrizen faktorisiert. Dadurch sind anstelle von N^2 nur noch $Nlog_2N$ komplexe Additionen notwendig[7].

Im folgenden Kapitel 3 wird die Effizienzsteigerung der Schnellen Fourier-Tranformation am Beispiel ersichtlich.

[7] [Mer23] S. 190 - 194.

Kapitel 3

Prototypische Umsetzung

3.1 Zielsetzung und Anforderungen

Die Anwendung verfolgt das Ziel, ein System zu entwickeln, das Bilder und Audiodateien (im WAV-Format) in binären Mustern kodieren und dekodieren kann. Dabei werden verschiedene Algorithmen genutzt, um Bilder in eine repräsentative Audiosequenz zu übersetzen und umgekehrt aus einer Audiodatei ein Bild zu rekonstruieren.

Zentrale Ziele:

Bild zu Audio Konvertierung Ein Bild wird in ein binäres Bitmuster überführt und anschließend mithilfe spezifischer Frequenzen in eine Wellenform (WAV-Datei) umgewandelt.

Audio zu Bild Konvertierung Eine kodierte Audiodatei wird analysiert, um die darin enthaltenen Frequenzmuster zu extrahieren und in ein Bitmuster zu überführen. Dieses Muster dient als Grundlage für die Rekonstruktion des ursprünglichen Bildes.

Benutzerfreundlichkeit Eine Web-Oberfläche soll die Nutzung der Anwendung erleichtern und eine komfortable Steuerung der Konvertierung ermöglichen.

Funktionale Anforderungen:

Unterstützung gängiger Formate Die Anwendung muss die Verarbeitung von Bildern in JPEG, PNG und Audiodateien in WAV ermöglichen.

Farbtiefen-Kodierung Bilder sollen mit einer Farbtiefe von bis zu 8 Bit kodiert werden können. Je nach gewählter Farbtiefe erfolgt die Übertragung entweder in Graustufen (1–3 Bit) oder in Farbe (4 und 8 Bit)

Effizienz und Genauigkeit Die Umwandlung sollte in beiden Richtungen möglichst schnell und in einer möglichst exakten Darstellung des ursprünglichen Bildes resultieren, sprich mehr als 85% der Pixel entsprechen dem Original.

3.2 Aufbau der Anwendung

3.2.1 Technischer Aufbau

Die Anwendung besteht primär aus einer, im Web bereitgestellten Benutzeroberfläche, welche die Bedienung folgender Funktionen ermöglicht:

- Import von Bilder- und Audiodateien
- Aufnahme von Audio direkt über die Webanwendung
- Steuerung der Konvertierungsparameter, wie Farbtiefe und Bilddimensionen
- Export der erzeugten Dateien

Der Bedienablauf der Anwendung lässt sich in folgende Schritte gliedern:

1. Der Nutzer importiert eine Datei (Bild oder Audio) über das Frontend und wählt die gewünschten Konvertierungsparameter (Kodierung oder Dekodierung, Farbtiefe, Bilddimensionen).
2. Das Backend verarbeitet die Datei gemäß der gewählten Funktion und Parameter.
3. Das Ergebnis wird vom Backend zurück an das Frontend übermittelt und steht dort mit einer Anzeige als Export bereit.

Für die Umsetzung werden folgende Technologien verwendet:

- *Python* als Programmiersprache
- *Flask*, einem Web Framework für Python
- *NumPy*, ein Paket für Python für wissenschaftliche, numerische Berechnungen
- *Pillow*, ein Paket für Python, welches Bildverarbeitungsalgorithmen hinzufügt

Die einzelnen Vorgänge sind wie folgt realisiert:

3.2.2 Kodierung der Bilddaten in Audiosignale

Die Kodierung eines Bildes in ein Audiosignal erfolgt durch eine mehrstufiges Umwandlung der Bilddaten in ein binäres Bitmuster. Um das Bitmuster in der Audiodatei darzustellen wird die Modulationstechnik Frequency Shift Keying benutzt.

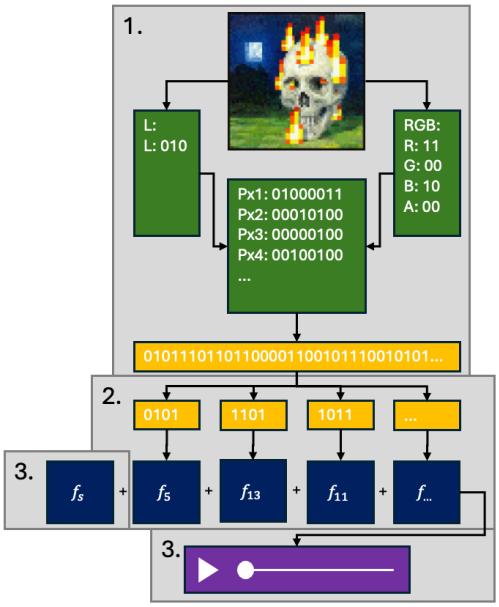


Abbildung 3.2.1: Visualisierung des Bildkodierungsprozesses (Werte fiktiv und dienen ausschließlich Demonstrationszwecken)

1. Konvertierung des Bildes in ein Bitmuster: Das Bild wird in ein binäres Format überführt, wobei die Farbtiefe die Genauigkeit der gespeicherten Informationen bestimmt. Um die Bilddaten interpretieren zu können, wird hierbei von der *Pillow* Bibliothek Gebrauch gemacht. Diese ermöglicht es, die verschiedenen Pixelwerte als Array aus einem Bild herauszulesen und weiter zu verarbeiten. Bei einer Farbtiefe von 1-3 Bit werden Graustufen verwendet. Dabei wird jeder Pixel auf eine festgelegte Anzahl von Graustufen reduziert. Bei 4 oder 8 Bit enthält jeder Pixel Informationen für 4 Farbkanäle: Rot, Grün, Blau und Amber. Bei 4 Bit hat jeder Farbkanal nur 2 Stufen (an / aus), wodurch eine sehr einfache Farbauswahl entsteht. Bei 8 Bit hat jeder Kanal immerhin 4 Stufen, wodurch eine detailliertere Farbdarstellung möglich ist. Das resultierende Bitmuster setzt sich aus den einzelnen Farbwerten aller Pixel in einer fortlaufenden Sequenz zusammen.

2. Gruppierung der Bits: Um die Daten effizienter im Audiosignal darzustellen, werden die Bits in 4er-Gruppen (Symbole) zusammengefasst. Dadurch entstehen 16 mögliche Kombinationen, die jeweils einer bestimmten Frequenz zugeordnet werden. Dieser Vorgang reduziert die Datenmenge, da ein Symbol vier Bits repräsentiert, was einer 1:4-Komprimierung entspricht.

3. Einfügen eines Startmarkers: Dem Audiosignal wird ein spezieller Startmarker vorangestellt, der als Synchronisationspunkt für die spätere Dekodierung dient. Dieser Marker besitzt eine eindeutige Frequenz und festgelegte Dauer, sodass er zuverlässig erkannt werden kann.

4. Synthese des Audiosignals: Ein digital gespeichertes Audiosignal besteht aus einem Array von Datenpunkten n, wobei je nach Abtastrate in einer Sekunde unterschiedlich viele Werte gelesen werden. Für jedes kodierte Symbol wird eine Sinuswelle mit der entsprechenden Frequenz generiert. Die Dauer jedes Symbols wird durch die festgelegte Symboldauer bestimmt. Bei einer Abtastrate von 44100 und einer Symboldauer von $0,03s$ entstehen pro Symbol 1.323 Werte n. Diese werden nacheinander in dem Array gespeichert, welche dann als .WAV Datei gespeichert werden.

3.2.3 Übertragungsmechanismen

Die Übertragung des kodierten Audiosignal erfolgt durch die Speicherung und Weitergabe im WAV-Format oder durch die Aufnahme der Wiedergabe dieses Signals mithilfe eines Mikrofons. Bei der manuellen Aufnahme des wiedergegebenen Signals kann es dabei zu Störungen kommen, da neben dem abgespielten Audiosignal auch Hintergrundgeräusche oder Stimmen aufgenommen werden.

3.2.4 Rekonstruktion der Bilddaten

Die Umwandlung eines kodierten Audiosignals zurück in ein Bild erfolgt primär durch eine Analyse des Frequenzspektrums. Das Ziel ist es, die im Signal enthaltenen binären Daten exakt zu extrahieren und in ein visuelles Format zurückzuführen.

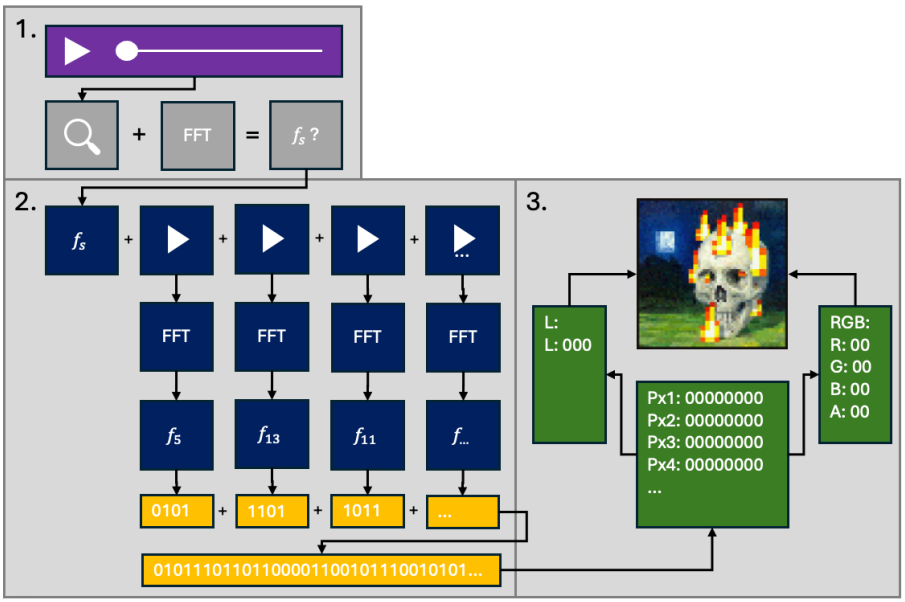


Abbildung 3.2.2: Visualisierung des Bildwiederherstellungsprozesses (Werte fiktiv und dienen ausschließlich Demonstrationszwecken)

1. Identifikation des Startmarkers: Das Audiosignal wird auf den Startmarker analysiert, um den Beginn der kodierten Daten zu ermitteln. Dabei wird das Audiosignal iterativ in Abschnitten von 0,005 Sekunden durchlaufen, bis der Startmarker gefunden wurde. Das Ende des Startmarkers dient als Synchronisationspunkt für die weiteren Informationen.

2. Extraktion der Daten: Nach der Erkennnung des Startmarkers beginnt die eigentliche Dekodierung: Das Signal wird in zeitliche Abschnitte unterteilt, die der Dauer eines Symbols entsprechen. Jeder Abschnitt wird mithilfe der Fourier-Transformation analysiert, um die dominante Frequenz zu ermitteln. Die extrahierten Frequenzen werden den entsprechenden Symbolfrequenzen zugeordnet, welche jeweils einem 4-Bit Wert entsprechen. Bei Abweichungen wird auf die nächst passende Frequenz gerundet um bei Störungen nicht den Prozess abbrechen zu müssen. Die extrahierten Werte werden schrittweise aneinandergereiht, um das ursprüngliche Bitmuster wiederherzustellen.

Zur Berechnung der Fourier-Transformation wurden drei verschiedene Methoden implementiert: eine manuelle Implementierung der FFT (A.1.2) und DFT (A.1.3) und eine Lösung unter Verwendung der FFT Methoden, die von der *Numpy* Bibliothek bereitgestellt werden (A.1.1). Mit den Ergebnissen der Fourier Transformation wird dann die dominante Frequenz berechnet (A.1.4).

3. Wiederherstellung des Bildes: Je nach angegebenen Bilddimensionen und Farbtiefe wird das Bitmuster auf die passende Länge überprüft. Anschließend werden die binären Werte in Pixelwerte umgerechnet und in einem Array gespeichert. Mithilfe der *Pillow* Bibliothek wird aus den einzelnen Pixelwerten das Bild wiederhergestellt. Sollte das Bitmuster durch Störungen unvollständig oder fehlerhaft sein, kann das resultierende Bild Artefakte oder Farbverfälschungen enthalten.

Kapitel 4

Analyse und Ergebnisse

Die Anwendung wurde mithilfe des folgenden Bildes mit 16x16 Pixel getestet:

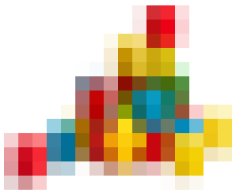
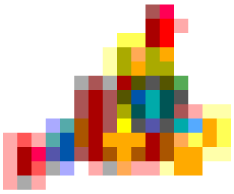


Abbildung 4.0.1: Bild mit 16x16 Pixeln

Bei einer Übertragung mit 3 Bit Farbtiefe werden $\frac{16*16*3}{4} = 192$ Symbole benötigt und bei 8 Bit $\frac{16*16*8}{4} = 512$ Symbole. Bei einer Symboldauer von $0,03s$ und 4 Bit pro Symbol ergibt das eine Datenrate von 133,3 Bit/s. Das entstehende Audio des 3 Bit Bildes ist 6 Sekunden lang und das Audio des 8 Bit Bildes 15 Sekunden. Die Länge des Audios skaliert linear mit der Anzahl an benötigten Symbolen, da pro Symbol eine Frequenz an das Audiosignal angehangen wird. Die Audiodateien wurden nachfolgend mithilfe eines Laptop-Mikrofons aufgenommen und in der Anwendung wieder in ein Bild umgewandelt.

Folgende Bilder waren das Ergebnis der Wiederherstellung:

(a) Rekonstruiertes Bild mit 3-Bit Farbtiefe



(b) Rekonstruiertes Bild mit 8-Bit Farbtiefe

Abbildung 4.0.2: Vergleich der 3-Bit und 8-Bit Versionen nach Wiederherstellung

4.1 Vergleich von DFT, FFT und der FFT der Numpy-Bibliothek

Der entscheidende Unterschied zwischen den verschiedenen Implementierungsarten der Fourier-Transformation liegt in ihrer Laufzeit. Zur Rückgewinnung der ursprünglichen Bitfolge muss die dominante Frequenz für jedes Symbol berechnet werden. Die Ergebnisse des obigen Versuchs mit 8 Bit Farbtiefe zeigen dabei deutliche Unterschiede in der Effizienz der Algorithmen. Um die dominanten Frequenzen zu berechnen benötigt:

- Eigens implementierte FFT: 7 Sekunden für das gesamte Signal
- NumPy FFT: 0,3 Sekunden für das gesamte Signal
- Eigens implementierte DFT: 11 Sekunden pro Symbol → Berechnungszeit von etwa 1,5 Stunden für das gesamte Signal

Neben der reinen Messung lässt sich dieser enorme Laufzeitunterschied zwischen DFT und FFT auch durch ihre jeweilige algorithmische Komplexität erklären. Während die DFT einen Rechenaufwand von $O(N^2)$ benötigt, reduziert die FFT diesen auf $O(NlogN)$.

Für eine Überschlagsrechnung betrachten wir $N \approx 1300$ Werte pro Symbol.

$$\text{DFT:} N^2, \quad \text{FFT:} N \log_2 N \tag{4.1.1}$$

Das Verhältnis der benötigten Operationen beträgt:

$$\frac{N^2}{N \log_2 N} = \frac{N}{\log_2 N} \tag{4.1.2}$$

Für $N = 2000$:

$$\frac{1300}{\log_2 1300} \approx 125 \tag{4.1.3}$$

Die FFT benötigt also bei $N \approx 1300$ etwa 125-mal weniger Operationen. Die oben gemessene Verbesserung beträgt in etwa $\frac{11s*512}{7s} \approx 800$. Diese Abweichung zwischen mathematischen und gemessenen Wert lässt sich auf zusätzliche Faktoren wie Speicherzugriffe und Cache-effekten zurückführen, die pro Operation erfolgen. Das Verhältnis nimmt bei einer höheren Anzahl an Werten immer weiter zu, was die Wichtigkeit der FFT vor allem bei langen Signalanalysen verdeutlicht (siehe 4.1.1). Die FFT legte aufgrund ihrer Effizienzsteigerung den Grundstein für viele moderne Anwendungen in der Signalverarbeitung.

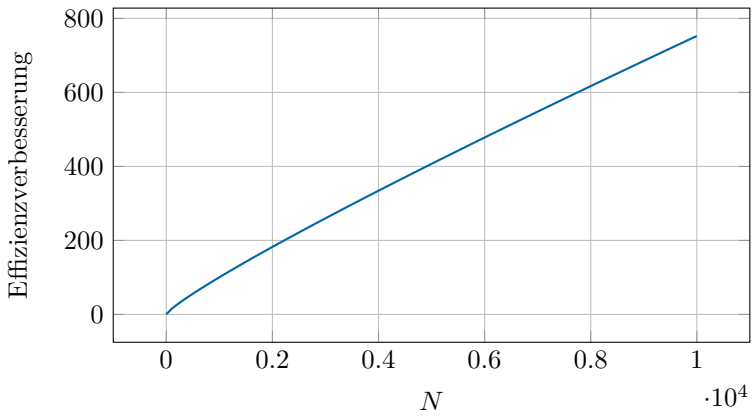


Abbildung 4.1.1: Visualisierung der Effizienzverbesserung zwischen DFT und FFT bei N Werten

4.2 Qualität der Bildrekonstruktion

Die Qualität der Bildrekonstruktion kann nur anhand des Vergleichs zwischen dem bereits in die passende Auflösung und Farbtiefe konvertierten Bild und dem wiederhergestellten Bild beurteilt werden.

Ein direkter Vergleich des vom Nutzer importierten Originalbildes mit der rekonstruierten Version wäre hier nicht aussagekräftig, da die Anwendung nur darauf ausgelegt ist, begrenzte Bildinformationen wiederherzustellen. Beim Import des Bildes wird dieses, wenn nötig, auf maximal 64x64 Pixel skaliert und auf die gewünschte Farbtiefe komprimiert.

Würde man dennoch das importierte Bild ohne Bearbeitung mit der rekonstruierten 8-Bit-Version vergleichen, ergäbe sich lediglich eine Übereinstimmung von $52,73\%$ (135 von 256 Pixeln sind identisch). Dies liegt jedoch an der vorherigen Umwandlung in das 8-Bit-Format und nicht an der eigentlichen Audioübertragung.

Betrachtet man hingegen den direkten Vergleich zwischen dem vor der Audioübertragung komprimierten 8-Bit-Bild und der nach der Audioübertragung rekonstruierten Version, so zeigt sich bei perfekten Aufnahmebedingungen[8] eine Rekonstruktionsqualität von 100% (alle 256 von 256 Pixeln sind identisch).

Um eine Störung zu simulieren wurde innerhalb der Audioaufnahme 2s lang ein fremdes Audiosignal, welches Hintergrundgeräusche simuliert[9] lauter als die Informationsquelle abgespielt. Das daraus resultierende Bild beinhaltet Pixel, deren Symbole von dem Fremdsignal überlagert wurden und somit nicht mehr den Originalinhalt darstellen, da die Implementierung nur die dominante Frequenz beachtet. Diese Umsetzung beeinträchtigt die Qualität der Bildrekonstruktion bei nicht idealen Aufnahmeverhältnissen und bedarf somit einer zukünftigen Optimierung, um eine zuverlässige Qualität sicherzustellen.

4.3 Herausforderungen bei der Umsetzung

Ein wesentliches Problem stellte die präzise Bestimmung des Startsignals dar. Eine fehlerhafte Detektion führt dazu, dass sich nachfolgende Symbole verschieben und die Rekonstruktion des Bildes ungenau oder sogar unbrauchbar wird.

Zudem zeigte sich eine deutliche Ineffizienz bei der Verarbeitung großer Bilder. Da jeder Bildausschnitt in einzelne Symbole umgewandelt wird, steigt die Rechenlast mit der Anzahl der Pixel und der gewählten Farbtiefe schnell an. Besonders die manuelle Implementierung der FFT war hier ein begrenzender Faktor, da sie zwar eine effizientere Berechnung als die DFT ermöglicht, jedoch nicht mit den hochoptimierten FFT-Algorithmen, wie sie in Bibliotheken wie *Numpy* zur Verfügung stehen, mithalten konnte.

[8] Mikrofon direkt neben der Audioquelle, keine bis sehr leise Hintergrundgeräusche, kleiner und hallgedämpfter Raum

[9] : https://www.youtube.com/watch?v=0QKdqm5TX6c ab 25:00 Minuten.

Kapitel 5

Vergleich mit der Unterwasserkommunikation

Kommentar

> *Die Informationen dieses Kapitels stammen aus einem Interview mit Peter Kampmann - Head of Maritime Robotics / Subsea Structures bei Rosenxt [Ros].*

Die Nutzung akustischer Signale zur Datenübertragung spielt eine zentrale Rolle in der Unterwasserkommunikation. Unterwasser sind herkömmliche drahtlose Kommunikationsmethoden aufgrund physikalischer Einschränkungen nicht praktikabel. So werden zum Beispiel elektromagnetische Wellen im Wasser stark absorbiert und haben somit eine zu geringe Reichweite. Dies betrifft vor allem die Datenübermittlung zwischen autonomen Unterwasserfahrzeugen (AUVs), Tauchrobotern und stationären Überwachungssystemen. Das Unternehmen Rosenxt Subsea Structures setzt in diesem Bereich auf Systeme von EvoLogics, welche mit hochentwickelten, spezialisierten Modulationsverfahren eine zuverlässige Kommunikation gewährleisten.

Dabei wird ein Frequenzbereich von 300-350kHz verwendet, die Kommunikation ist also nicht mit herkömmlichen Mikrofonen erfassbar und liegt weit außerhalb des menschlich hörbaren Bereichs von etwa 16 Hz bis 20.000 Hz. Zum Vergleich: die in dieser Arbeit behandelte Anwendung verwendet einen Frequenzbereich von 500-2100Hz.

Auch im professionellen Anwendungsbereich stellen begrenzte Datenraten und hohe Störanfälligkeiten Herausforderungen dar. Rosenxt erreicht bei ihrer Kommunikation eine Datenrate im unteren Kilobit pro Sekunde Bereich, was vor allem im Vergleich mit Kommunikationsmethoden wie WI-FI 6 mit mehreren Gigabit pro Sekunde [Int] sehr gering ist. Unterwasser ist das Signal durch Brechungen, Reflexionen und Absorptionen besonders anfällig für Verzerrungen oder Störungen. Bei Rosenxt wird das unteranderem durch wiederholtes Senden des Signals bis zur vollständigen Ankunft beim Ziel gelöst.

Dies verdeutlicht, dass sowohl in der professionellen Unterwasserkommunikation als auch in experimentellen Ansätzen, wie sie in dieser Arbeit untersucht wurden, die Herausforderungen in Bezug auf Signalstabilität und Datenrate zentrale Aspekte bleiben, die durch optimierte Modulationsverfahren und adaptive Fehlerkorrektur jediglich weiter verbessert werden können.

Kapitel 6

Schlussfolgerung und Ausblick

6.1 Zusammenfassung der Ergebnisse

Die Analyse der Ergebnisse zeigt, dass die Anwendung erfolgreich ihr Ziel erfüllt, ein Bild mithilfe eines Audiosignals zu übertragen.

Die Signalverarbeitung mittels Fourier-Transformation erwies sich als zuverlässige Methode zur Identifikation der übertragenen Symbole, insbesondere durch die Nutzung der optimierten Numpy-FFT.

In Bezug auf die Effizienz der Fourier-Transformation konnte festgestellt werden, dass die Implementierung der FFT eine deutliche Verbesserung gegenüber der direkten DFT darstellt. Bei größeren Bilddimensionen wird die Audiolänge jedoch schnell unbrauchbar lang, was eine zukünftige Optimierung im Bereich Effizienz verpflichtend macht.

6.2 Mögliche Weiterentwicklungen

Basierend auf den gewonnenen Erkenntnissen ergeben sich mehrere Ansätze zur Verbesserung und Weiterentwicklung:

- **Verbesserte Störungserkennung und Filterung** Während der Tests zeigte sich, dass externe Einflüsse wie Rauschen oder Hintergrundgeräusche dazu führen können, dass Frequenzen überlagert werden und die dahinter stehenden Informationen verloren gehen. Durch Implementierung von Rauschunterdrückungsfiltern oder einer Beachtung von nur relevanten Frequenzen könnte man hier positive Veränderungen bewirken.
- **Effizienzsteigerung und weitere Komprimierung**[10] Momentan besteht die einzige Form von Komprimierung darin, dass 4 Bits pro Frequenz verwendet werden. In Zukunft könnte hier eine stärkere Komprimierung der Bilddaten oder eine Anpassung oder auch Zusammenführung von mehreren unterschiedlichen Modulationsmethoden entscheidende Änderungen in der Effizienz der Datenübertragung bewirken.
- **Erweiterung der Einschränkungen auf mehr Farbtiefen und größere Bilddimensionen** Durch eine Steigerung der Effizienz könnte man die Anwendung auch auf mehr Bilddimensionen und Farbtiefen erweitern.

Dieses Projekt zeigt, dass das System in der aktuellen Form erfolgreich arbeitet und die funktionalen Anforderungen erfüllt sind, es jedoch noch Potenzial für Verbesserungen gibt, insbesondere in Bezug auf Effizienz und Skalierbarkeit.

[10] automatische Erkennung optimaler Symbollängen

Quellenverzeichnis

[Int] Intel: *Was ist WLAN 6?* https://www.intel.de/content/www/de/de/gaming/resources/wifi-6.html. – Zugriff am 27. Februar 2025
(zitiert auf Seite 14)

[Mer23] Mertins, Alfred: *Signaltheorie: Grundlagen der Signalbeschreibung, Filterbänke, Wavelets, Zeit-Frequenz-Analyse, Parameter- und Signalschätzung.* Springer Fachmedien Wiesbaden, 2023. http://dx.doi.org/10.1007/978-3-658-41529-7. http://dx.doi.org/10.1007/978-3-658-41529-7. – ISBN 978–3–658–41528–0 978–3–658–41529–7
(zitiert auf Seiten 3, 4, 5 und 6)

[Ros] Rosenxt: *Subsea Structures.* https://www.rosen-nxt.com/en/business-fields/critical-infrastructure-integrity/subsea-structures. – Zugriff am 27. Februar 2025
(zitiert auf Seite 14)

[Rud06] Rudolph, Dietmar: *Digitale Modulationstechniken.* https://web.archive.org/web/20220331112217/http://www.diru-beze.de/funksysteme/skripte/DiFuSy_S06/DiFuSy_Mod_SS06.pdf. Version: 2006. – Zugriff am 20. Februar 2025
(zitiert auf Seite 4)

[Teu24] Teufel: *Digital zu analog – Wandler und Konverter machen es möglich.* https://blog.teufel.de/digital-zu-analog-wandler-und-konverter-machen-es-moeglich/. Version: 2024. – Zugriff am 23. Februar 2025
(zitiert auf Seite 3)

[Wei08] Weinzierl, Stefan (Hrsg.): *Handbuch der Audiotechnik.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2008. http://dx.doi.org/10.1007/978-3-540-34301-1. http://dx.doi.org/10.1007/978-3-540-34301-1. – ISBN 978–3–540–34300–4 978–3–540–34301–1
(zitiert auf Seite 3)

[Wik24a] Wikipedia: *Rainbow Books.* https://de.wikipedia.org/wiki/RainbowBooks. Version: 2024. – Zugriff am 20. Februar 2025
(zitiert auf Seite 3)

[Wik24b] Wikipedia: *Schnelle Fourier-Transformation.* https://de.wikipedia.org/wiki/Schnelle_Fourier-Transformation. Version: 2024. – Zugriff am 25. Februar 2025
(zitiert auf Seite 6)

[Wik25] Wikipedia: *Morsecode.* https://de.wikipedia.org/wiki/Morsecode. Version: 2025. – Zugriff am 15. Februar 2025
(zitiert auf Seite 4)

Hilfsmittel

Zur Erstellung der vorliegenden Belegarbeit wurden die folgenden Hilfsmittel verwendet:

- ChatGPT – https://chatgpt.com,
- Gemeinsame Bibliothek der BA Dresden und der ehs Dresden,
- Google-Scholar – https://scholar.google.com,
- Microsoft Windows 11,
- Apple macOS Sequoia,
- Craft Docs – https://www.craft.do/,
- Excalidraw – https://excalidraw.com/,
- Texifier – https://www.texifier.com/,
- Ich danke Herrn Prof. Dr. Tenshi Hara für die Bereitstellung einer Latex-Vorlage, auf der diese Belegarbeit aufbaut. – https://lern.es/LaTeX-Kurs/,

Teil II

Anhang

Anhang A

Quellkode

A.1 Implementierung der Fourier-Transformation

Quellkode A.1.1: Numpy FFT (Python)

```python
def npft(signal):
    '''
    calculate fourier transform of signal with numpy library
    :param signal: array of signal values
    :return: array of complex values
    '''
    return np.fft.fft(signal)
```

Quellkode A.1.2: Manuelle FFT (Python)

```python
def fft(signal):
    '''
    fast fourier transform on signal
    :param signal: array of signal values
    :return: array of complex values
    '''
    n = len(signal)

    if n <= 1:
        return signal

    even = fft(signal[::2])
    odd = fft(signal[1::2])

    T = [math.e**(-2j * math.pi * k/n) * odd[k] for k in range(n//2)]
    return [even[k] + T[k] for k in range(n//2)] + \
           [even[k] - T[k] for k in range(n//2)]
```

Quellkode A.1.3: Manuelle DFT (Python)

```python
def dft(signal):
    '''
    calculate discrete fourier transform of signal
    :param signal: array of signal values
    :return: array of complex values
    '''
    start_time = time.time()

    N = len(signal)
    result = []
    for k in range(N):
        r = 0
        i = 0
        for n in range(N):
            angle = 2 * np.pi * k * n / N
            r += signal[n] * np.cos(angle)
            i -= signal[n] * np.sin(angle)
        result.append(complex(r, i))

    end_time = time.time()
    print("symbol time cost:", end_time - start_time)

    return np.array(result)
```

Quellkode A.1.4: Dominante Frequenz berechnen (Python)

```python
def get_dominant_frequency(signal, ft_result):
    '''
    calculate dominant frequency from ft result
    :param signal: array of signal values
    :param fft_result: result dft/npft/fft(signal) - array of complex values
    :return: array of complex values
    '''
    n = len(signal)

    freqs = [(i * sample_rate) / n for i in range(n)]
    freqs_pos = freqs[:n//2]

    amps = [abs(x) for x in fft_result]
    amps_pos = amps[:n//2]

    max_index = amps_pos.index(max(amps_pos))
    dominant_frequency = abs(freqs_pos[max_index])

    return dominant_frequency
```

Erklärung an Eides statt

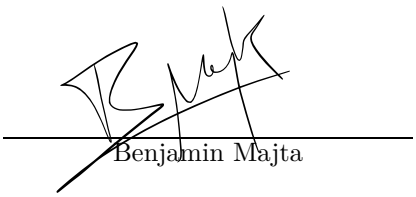
Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen sowie die explizit aufgeführten Hilfsmittel verwendet habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden, ansonsten sind sie mit einem entsprechenden Quellennachweis versehen.

Diese Ausarbeitung ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Dresden, 28. Februar 2025



Benjamin Majta