Bachelorarbeit

Integration einer Sprachsteuerungsfunktion in Mobile Apps

Rubén Nuñez

Herbstsemester 2023

Bachelorarbeit an der Hochschule Luzern – Informatik

Titel: Integration einer Sprachsteuerungsfunktion in Mobile Apps

Studentin/Student: Ruben Nuñez

Studiengang: BSc Informatik

Jahr: 2023

Betreuungsperson: Dr. Florian Herzog

Expertin/Experte: xxx

Auftraggeberin/Auftraggeber: Stefan Reinhard, Bitforge AG

Codierung / Klassifizierung der Arbeit:
☒ Öffentlich (Normalfall)
☐ Vertraulich

Eidesstattliche Erklärung Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe angefertigt habe, alle verwendeten Quellen, Literatur und andere Hilfsmittel angegeben habe, wörtlich oder inhaltlich entnommene Stellen als solche kenntlich gemacht habe, das Vertraulichkeitsinteresse des Auftraggebers wahren und die Urheberrechtsbestimmungen der Hochschule Luzern respektieren werde.

Ort / Datum, Unterschrift _____

Abgabe der Arbeit auf der Portfolio Datenbank:
Bestätigungsvisum Studentin/Student
Ich bestätige, dass ich die Bachelorarbeit korrekt gemäss Merkblatt auf der Portfolio Datenbank abgelegt habe. Die Verantwortlichkeit sowie die Berechtigungen habe ich abgegeben, so dass ich keine Änderungen mehr vornehmen kann oder weitere Dateien hochladen kann.

Ort / Datum, Unterschrift _____

Verdankung gibt ein separiertes Kapitel dazu

Ausschliesslich bei Abgabe in gedruckter Form: Eingangsvisum durch das Sekretariat auszufüllen

Rotkreuz, den _____ Visum: _____

Abstract

Das Problem dieser Arbeit ist im wesentlichen die Erkennung von Triggerwörtern innerhalb des Kontext einer App. Grundsätzlich ist es unüblich, dass mobile Apps eine integrierte Sprachsteuerungsfunktion anbieten.

Inhaltsverzeichnis

1 Problem, Fragestellung, Vision

Das Kernproblem dieser Bachelorarbeit ist die Entwicklung einer integrierten Spracherkennungsfunktion die bestimmte Triggerwörter innerhalb einer mobilen App erkennt. Obwohl Sprachsteuerungstechnologien ein erhebliches Potenzial aufweisen und Assistenten wie Siri oder Alexa weit verbreitet sind, bieten mobile Apps selten eine integrierte Spracherkennung. Dies führt zu einer Lücke, da solche Assistenten nicht spezifisch für App-Kontexte optimiert sind. Diese Arbeit verfolgt das Ziel, diese Lücke zu schliessen und eine integrierte Spracherkennungsfunktion zu entwickeln, die Triggerwörter in einer App effektiv erkennt.

1.1 Fragestellung

Die Fragestellung dieser Arbeit lautet: *Wie kann eine integrierte Sprachsteuerung für eine Mobile Apps entwickelt werden, die speziell das Erkennen von Triggerwörtern ermöglicht, indem Methoden des Machine Learnings genutzt werden?*.

1.2 Vision

Das Ziel sowie die Vision der Arbeit ist es zum einen, eine Grundlage zu schaffen, um ein Triggerwort oder eine Sequenz von Triggerwörtern in der akustischen Sprache erkennen zu können. Dabei werden Methoden und Werkzeuge aus dem Bereich des Machine Learnings verwendet. Zum anderen soll diese Erkenntnis in eine mobile Plattform wie iOS oder Android integriert werden. Für den Rahmen dieser Arbeit genügt die Integration in eine der genannten Plattformen. Weiterhin werden das Thema Datenschutz und die ethischen Aspekte berücksichtigt.

2 Grundlagen

Um diese Arbeit fundiert anzugehen, ist ein Verständnis der Grundlagen in den Bereichen Audioverarbeitung und Machine Learning essenziell. Daher wird in diesem Kapitel ein Überblick über die wichtigsten Themen gegeben. ...

2.1 Audio

In der digitalen Welt repräsentiert Audio Schallwellen, die durch eine Reihe von numerischen Werten dargestellt werden Somberg et al., 2019, p.9. beschreibt Audio als: „Fundamentally, audio can be modeled as waves in an elastic medium. In our normal everyday experience, the elastic medium is air, and the waves are air pressure waves.“ Audiosignale werden durch die Funktion $A(t)$ repräsentiert, wobei t die Zeit und $A(t)$ die Amplitude zum Zeitpunkt t angibt. Die Amplitude ist die Stärke des Signals und die Zeit repräsentiert die Position des Signals in der Zeit. Diese Betrachtung ist vor allem in der Elektrotechnik von Bedeutung, da die Amplitude als Spannung angesehen werden kann. Grundsätzlich ist Audio ein kontinuierliches Signal. In der digitalen Welt können wir jedoch nur diskrete Werte darstellen. Daher wird das kontinuierliche Signal in diskrete Werte umgewandelt. Dieser Vorgang wird als *Sampling* bezeichnet (Tarr, 2018, Chapter 3.1).

2.1.1 Sampling

Ein früher Ansatz zur digitalen Darstellung von analogen Signalen war die Pulse-Code-Modulation (PCM). Dieses Verfahren wurde bereits in den 1930er Jahren von Alec H. Reeves entwickelt, parallel zum Aufkommen der digitalen Telekommunikation (Deloraine und Reeves, 1965, p. 57). Im Grundsatz wird es heute noch in modernen Computersystemen nach dem gleichen Verfahren angewendet.

Es folgt eine formelle Definition von Sampling. Ein kontinuierliches Signal $A(t)$ wird in bestimmten Zeitintervallen T_s gesampelt. Diese Zeitintervalle werden auch als Sampling-Periode bezeichnet. Die Sampling-Rate $F_s = \frac{1}{T_s}$ gibt die Anzahl der Samples pro Sekunde an. Angenommen wir haben ein Signal mit einer Sampling-Periode von $T_s = 0.001$. Um nun die Sampling-Rate zu berechnen, müssen wir den Kehrwert der Sampling-Periode berechnen. $F_s = \frac{1}{0.001} = 1000$. Somit erhalten wir eine Sampling-Rate von 1000 Samples pro Sekunde. Nun typische Sampling-Raten sind 44100 Hz oder 48000 Hz. Bei Sampling-Raten wird die Einheit *Hertz* verwendet. Ein Hertz entspricht einer Frequenz von einem Sample pro Sekunde. Ein weiter wichtiger Begriff ist die *Nyquist-Frequenz*. Die Nyquist-Frequenz F_n ist die Hälfte der Sampling-Rate. Also $F_n = \frac{F_s}{2}$. Die Idee hinter der Nyquist-Frequenz ist, dass die Sampling-Rate mindestens doppelt so hoch sein muss wie die höchste Frequenz des Signals. Wenn diese Eigenschaft erfüllt ist, kann das Signal ohne Informationsverlust rekonstruiert werden (Tarr, 2018, Chapter 3.1). Mehr dazu folgt im Unterkapitel *Fourier-Analyse*.

Weiter ist es wichtig zu verstehen, dass ein Sample ein diskreter Wert ist. Und dieser wird in digitalen Systemen durch eine bestimmte Anzahl von Bits dargestellt. Die Anzahl der Bits wird als *Bit-Depth* bezeichnet. Die Bit-Depth bestimmt die Auflösung des Signals. Typische Bit-Depth Werte sind 16 oder 24 Bit (Somberg et al., 2019, p.10).

2.1.2 Frames, Channels, Buffers

Ebendfalls wichtig ist das Verständnis von Frames, Channels und Buffers. Da diese Arbeit sich mit Audio-Systemen beschäftigt, ist es wichtig, die Begriffe *Frame*, *Channel* und *Buffer* zu verstehen. Fangen wir mit dem Begriff *Channel* an. Ein Channel kann als ein einzelnes Audio-Signal angesehen werden. Ein Mono-Signal hat genau nur einen Channel. Ein Stereo-Signal hat zwei Channels. Ein Surround-Signal hat mehr als zwei Channels. usw. Nun zum Begriff *Frame*. Ein Frame entspricht einem Sample pro Channel. Weiter sind Frames in Buffers organisiert. Ein Buffer ist eine Sammlung von Frames. Typischerweise werden Buffers in Grössen von 64, 128, 256, 512 oder 1024 Frames organisiert.

Die Abbildung 1 zeigt die Beziehung zwischen Frames, Channels und Buffers. Die Abbildung wurde basierend auf (Somberg et al., 2019, p.10) erstellt und verdeutlicht die Beziehung zwischen Frames, Channels und Buffers.

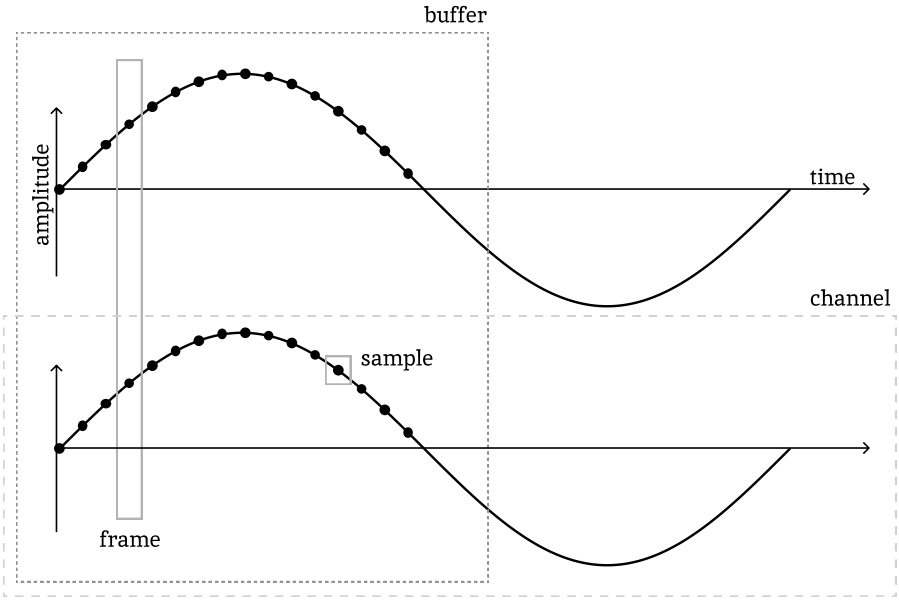

Abbildung 1: Frames, Channels und Buffers

2.1.3 Buffers im Detail

Ein Buffer im Kontext von Audio ist eine aufeinanderfolgende Sammlung von Frames. Die bereits angesprochene Grösse eines Buffers bestimmt im wesentlichen die Latenzzeit des Systems. Kleine Buffer-Grössen haben eine geringe Latenzzeit, während grosse Buffer-Grössen eine hohe Latenzzeit haben (Somberg et al., 2019, p.10). Der Trade-Off ist dass kleine Buffer-Grössen zu einer höheren CPU-Auslastung führen, während bei grossen Buffer-Grössen das nicht der Fall ist. Das liegt daran, dass bei kleinen Buffer-Grössen die CPU häufiger aufgerufen wird, um die Buffers zu verarbeiten.

Nun betrachten wir die mögliche Anordnung eines Buffers, wie in den folgenden Abbildungen dargestellt. Es gibt zwei Möglichkeiten, wie Buffers angeordnet werden können: *Interleaved* und *Non-Interleaved*. Bei der *Interleaved*-Anordnung werden die Samples der einzelnen Channels nacheinander in sequentieller Reihenfolge in den Buffer geschrieben. Im Gegensatz dazu werden bei der *Non-Interleaved*-Variante die Samples eines Channels nacheinander in den Buffer geschrieben, bevor die Samples des nächsten Channels hinzugefügt werden. Dieser Vorgang wird für jeden Channel wiederholt. Die Abbildung 2 zeigt die Unterschiede zwischen den beiden Anordnungen. Jede Zelle der Tabelle entspricht einem Sample. L und R stehen exemplarisch für die Channels Left und Right. Die erste Zeile entspricht der *Interleaved*-Anordnung und die zweite Zeile der *Non-Interleaved*-Anordnung. Die Abbildung wurde basierend auf (Somberg et al., 2019, p.11) erstellt.

L	R	L	R	L	R	L	R
L	L	L	L	R	R	R	R

Abbildung 2: Frames in Interleaved und Non-interleaved Buffers

Mit diesem Wissen kennen wir nun die Unterschiede zwischen den beiden Anordnungen. Für die Anwendung ist es wichtig zu verstehen, mit welcher Anordnung die verwendete API arbeitet.

2.1.4 Einblick in Audio APIs

Audio APIs sind im Bereich der Audioverarbeitung von essentieller Bedeutung. Sie bieten eine Schnittstelle, welche den Zugriff auf vielfältige Audiofunktionen erlaubt. Ohne solche APIs stünden Entwickler vor der Mammutaufgabe, eigenständige Schnittstellen und Treiber für jedes Projekt neu zu konzipieren. Einige der in dieser Arbeit verwendeten Schlüssel-APIs werden in diesem Abschnitt näher beleuchtet.

In der Erarbeitungsphase dieser Arbeit kristallisierten sich zwei primäre Anwendungsgebiete heraus. Erstens die intensive Auseinandersetzung mit Audioverarbeitung in Python, um tieferes Verständnis für die Materie zu entwickeln. Zweitens die Notwendigkeit, eine Audio API in eine mobile Applikation zu integrieren. Im Kontext dieser Arbeit werden sie als *Audio API für Analyse* und *Audio API für Integration* bezeichnet.

Im Bereich der *Analyse* fiel die Wahl auf folgende APIs:

- **PyAudio:** Eine verbreitete Schnittstelle in Python zur Audioverarbeitung.
- **SoundDevice:** Eine vielseitige Python-Bibliothek für vielschichtige Audioverarbeitungsaufgaben.
- **librosa:** Eine Bibliothek, die speziell auf die Analyse von Audiosignalen ausgerichtet ist.

Im Kontext der *Integration* standen folgende APIs im Fokus:

- **AVAudioEngine:** Eine leistungsfähige Schnittstelle primär für die Plattformen iOS und macOS.
- **AudioTrack:** Eine spezialisierte API für Audioanwendungen auf Android-Geräten.

Die folgenden Abschnitte werden tiefer auf die jeweiligen Eigenschaften und Möglichkeiten dieser APIs eingehen, insbesondere im Hinblick darauf, wie sie sich für die Aufnahme, Wiedergabe und Echtzeitverarbeitung von Audiodaten eignen.

2.1.5 Audio API für Analyse

Python zeichnet sich durch eine beeindruckende Auswahl an Bibliotheken für datenanalytische Aufgaben aus, zu denen auch NumPy, SciPy, Pandas und Matplotlib gehören. In dieser Arbeit wurde zunächst **PyAudio** in Erwägung gezogen. PyAudio ist als Schnittstelle zur PortAudio-Bibliothek bekannt, die plattformübergreifende Audioverarbeitungsfunktionen bereitstellt. Trotz ihrer intuitiven Funktionen für Aufnahme und Wiedergabe wurde PyAudio letztlich aufgrund von Inkompatibilitäten mit der gewählten Entwicklungsumgebung verworfen.

2.1.6 Integration und Anwendung von Audio APIs

Die effektive Nutzung einer Audio API erfordert ein Verständnis ihrer Struktur und Funktionsweise. Beispielsweise ist es wichtig zu wissen, wie Buffers und Channels in der jeweiligen API gehandhabt werden. In der Regel bieten APIs Funktionen zum Initialisieren von Audio-Streams, zum Steuern von Audio-Parametern wie Abtastrate und Bitrate und zum Verarbeiten von Audio-Daten in Echtzeit. Die korrekte Integration und Anwendung dieser Funktionen ist entscheidend für die Erstellung qualitativ hochwertiger Audio-Anwendungen.

2.1.7 Zusammenfassung

Die Audioverarbeitung ist ein faszinierendes und komplexes Gebiet, das fundierte Kenntnisse in vielen verschiedenen Bereichen erfordert. Von der tiefen mathematischen Theorie der Fourier-Analyse bis hin zur praktischen Anwendung von Audio APIs gibt es viele Aspekte zu berücksichtigen. Diese Arbeit zielt darauf ab, einen umfassenden Überblick über die Schlüsselkonzepte und Techniken in diesem Bereich zu geben und als Fundament für zukünftige Forschung und Anwendung zu dienen.

2.2 Fourier-Analyse

Die Fourier-Analyse befasst sich mit der Zerlegung von Funktionen in Frequenzkomponenten. Die Fourier-Analyse ist ein wichtiges Konzept in der Signalverarbeitung und findet breite Anwendung in der Audioverarbeitung. Daher ist ein Grundverständnis für diese Arbeit relevant.

2.2.1 Fourier-Transformation

Die Fourier-Transformation ist ein zentrales Werkzeug der Fourier-Analyse. Sie ermöglicht die Zerlegung von Funktionen in ihre Frequenzkomponenten und die Rekonstruktion von Funktionen aus diesen Komponenten. Dies wird als Fourier-Analyse und Fourier-Synthese bezeichnet. Dieses Konzept wird auch von Prof. Dr. Weitz in seinem Video zu Fourier-Analyse erläutert (Weitz, 2023, 2:20). Mathematisch ausgedrückt wird die kontinuierliche Fourier-Transformation eines Signals $f(t)$ wie folgt definiert (Hansen, 2014, Chapter 5):

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t}\, dt$$

$F(\omega)$ ist die Fourier-Transformation von $f(t)$ (Weitz, 2023, 49:27). Als kleines Rechenbeispiel betrachten wir die Fourier-Transformation der Rechteckfunktion $\text{rect}(x)$, die wie folgt definiert ist:

$$\text{rect}(x) = \begin{cases} 1 & \text{für } -1 \leq x \leq 1 \\ 0 & \text{sonst} \end{cases}$$

Die Fourier-Transformation der Funktion $\text{rect}(x)$ kann wie folgt berechnet werden:

$$\begin{aligned} F(\omega) &= \int_{-\infty}^{\infty} \text{rect}(x) e^{-i\omega x}\, dx \\ &= \int_{-1}^{1} e^{-i\omega x}\, dx \\ &= \frac{1}{-i\omega} \left[e^{-i\omega x} \right]_{-1}^{1} \\ &= \frac{1}{-i\omega} \left(e^{-i\omega} - e^{i\omega} \right) \\ &= \frac{1}{-i\omega} \left(\cos(\omega) - i\sin(\omega) - \cos(\omega) - i\sin(\omega) \right) \\ &= \frac{1}{-i\omega} \left(-2i\sin(\omega) \right) \\ &= \frac{2\sin(\omega)}{\omega} \end{aligned}$$

Somit ist die Fourier-Transformation der Rechteckfunktion $\text{rect}(x)$ gleich $\dfrac{2\sin(\omega)}{\omega}$.

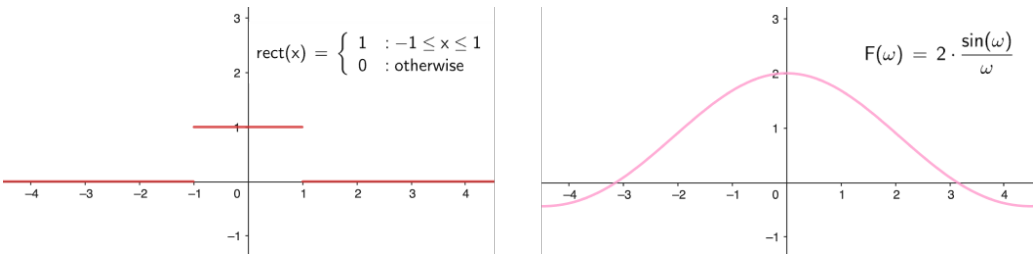

Abbildung 3: Rechteckfunktion und ihre Fourier-Transformation

Die Abbildung 3 stellt die $\text{rect}(x)$ Funktion und ihre Fourier-Transformation, die als $\text{sinc}(\omega)$ bezeichnet wird, dar. Die Nullstellen $\pm\pi, \pm 2\pi, \pm 3\pi, \ldots$ der $\text{sinc}(\omega)$ Funktion deuten darauf hin, dass die $\text{rect}(x)$

Funktion bei diesen Frequenzen keine Energie besitzt. Die primäre Energie der Funktion liegt bei $\omega = 0$. Beispiel adaptiert von (Hansen, 2014, Chapter 5 - Example 5.1).

2.2.2 Diskrete Fourier-Transformation

Die diskrete Fourier-Transformation (DFT) stellt eine diskrete Variante der kontinuierlichen Fourier-Transformation dar und wird speziell auf diskrete Signale angewendet. In digitalen Systemen sind Signale typischerweise diskret und bestehen aus einzelnen Samples, weshalb die DFT besonders relevant für solche Anwendungen ist. Die mathematische Definition der DFT ist (Hansen, 2014, Chapter 3):

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

Zur Veranschaulichung betrachten wir ein Code-Beispiel. Wir haben eine Funktion $f(t)$ und unterteilen diese in N Samples. Die DFT berechnet nun die Frequenzkomponenten des Signals. Die Abbildung 4 zeigt ein Beispiel für ein Signal $f(t)$ mit $N = 5$ Samples.

$$f(t) = 1.5\cos(t) + 0.25\sin(t) + 2\sin(2t) + \sin(3t)$$

.

```python
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return 1.5 * np.cos(x) + 0.25 * np.sin(x) + 2 * np.sin(2*x) + np.sin(3*x)

N_SAMPLES = 5

x_curve = np.linspace(0, 2*np.pi, 100) # 100 Punkte zwischen 0 und 2pi
x_points = np.linspace(0, 2*np.pi, N_SAMPLES) # 5 Punkte zwischen 0 und 2pi

plt.plot(x_curve, f(x_curve))
plt.plot(x_points, f(x_points), 'o') # Plotte die 5 Punkte
```

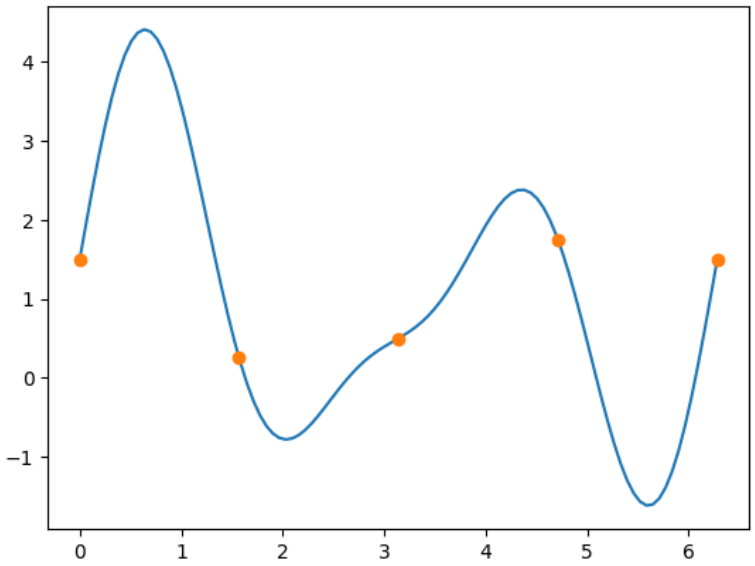

Abbildung 4: Funktion $f(x)$ mit 5 Samples

Nun berechnen wir die DFT der 5 Samples. Dazu verwenden wir die `fft` Funktion aus der `numpy` Bibliothek. Das Resultat ist ein Array mit N komplexen Zahlen. Die Abbildung 5 zeigt die Funktion $f(x)$ und die DFT der 5 Samples.

```
fhat = np.fft.fft(f(x_points), N_SAMPLES)
```

	0	1	2	3	4
$f(x)$	1.5	0.25	0.5	1.75	1.5
dft	$5.50 + 0.00i$	$0.22 + 1.92i$	$0.78 - 0.45i$	$0.78 + 0.45i$	$0.22 - 1.92i$

Abbildung 5: $f(x)$ und die DFT der 5 Samples

Mit den Frequenzkomponenten der DFT können Signale manipuliert werden, etwa durch Filtern bestimmter Frequenzbereiche. Um das ursprüngliche Signal wiederzuerlangen, wenden wir die inverse DFT an. Die Formel der inversen DFT, welche das Signal rekonstruiert, lautet (Hansen, 2014, Chapter 3):

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

```
reconstructed_manual = np.zeros_like(x_points, dtype=np.complex128)

dt = x_points[1] - x_points[0] # Abstand zwischen zwei Punkten
T = N_SAMPLES * dt # Periode des Signals

for n in range(N_SAMPLES):
    # Rekonstruiert Signal mit Fourier-Koeffizienten, neg. Freq. bei n > N_SAMPLES/2.
    freq = n / (2*np.pi) if n <= N_SAMPLES//2 else (n - N_SAMPLES) / (2*np.pi)
    reconstructed_manual += fhat[n] * np.exp(1j * 2 * np.pi * freq * x)

reconstructed_manual = (reconstructed_manual / N_SAMPLES).real
reconstructed = np.fft.ifft(fhat).real # Rekonstruiertes Signal mit ifft
```

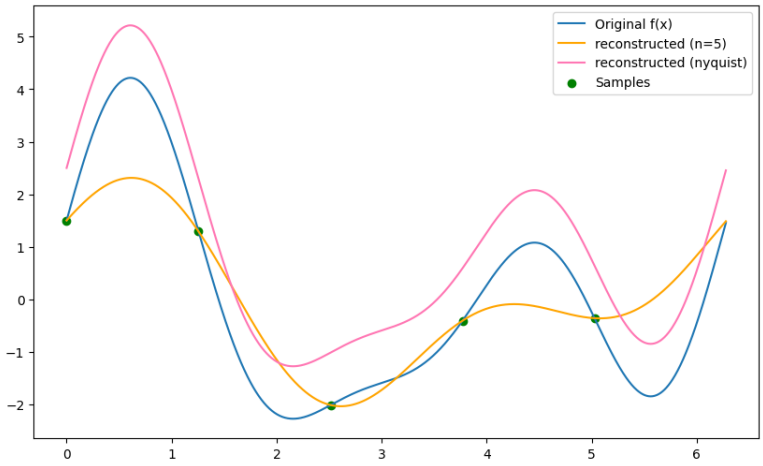

Abbildung 6: Rekonstruktion des Signals $f(x)$

Die Abbildung 6 zeigt die ursprüngliche Funktion $f(x)$ und die rekonstruierte Funktionen `reconstructed (n=5)` und `reconstructed (n=nyquist)`. Die Annäherung der mit der inversen DFT rekonstruierten Funktion an die ursprüngliche Funktion ist bei $n = 5$ deutlich sichtbar. Bei $n = nyquist$ ist die Annäherung nahezu perfekt, wenn nicht sogar perfekt. Die Sampling-Rate bei der Nyquist angenäherten Funktion ist das Doppelte der höchsten Frequenz des Signals. Das ist in diesem Fall $2 \cdot 3 + 1 = 7$.

2.2.3 Aliasing

Aliasing tritt auf, wenn ein Signal bei einer nicht ausreichend hohen Samplingrate digital erfasst wird, wodurch Frequenzen des Signals fehlinterpretiert werden können. Als allgemeines Beispiel wenn ein Sinussignal mit einer Frequenz von 1200 Hz betrachtet wird und dieses mit einer Samplingrate von nur 1000 Hz aufgenommen wurde, könnte das digitalisierte Signal so aussehen, als ob das ursprüngliche Signal eine Frequenz von 200 Hz hätte. Das ist, als ob man ein sich schnell drehendes Rad filmt und auf dem Video wirkt es, als würde es sich langsamer oder sogar rückwärts drehen. Um solche Fehler zu verhindern, sollte die Samplingrate stets mindestens das Doppelte der höchsten Frequenz des Signals betragen, ein Grundsatz, der als Nyquist-Kriterium bekannt ist. (Weitz, 2023).

2.3 Spektrogramm

Mit einem Verständnis der Grundlagen der Fourier-Analyse können wir die Bedeutung des Spektrogramms erfassen. Ein Spektrogramm bietet eine visuelle Darstellung der verschiedenen Frequenzen, die in einem Signal über die Zeit hinweg vorhanden sind. Ein Spektrogramm wird wie folgt definiert:

> "A spectrogram is a three-dimensional visualization of a signal's amplitude over frequency and time. Many audio signals are comprised of multiple frequencies occurring simultaneously, with these frequencies often changing over time." (Tarr, 2018, Chapter 15.2.1)

Im Bereich des Machine Learning, insbesondere bei der Spracherkennung, nimmt das Spektrogramm eine zentrale Position ein. Die Fourier-Transformation eines Audiosignals in seine Frequenzkomponenten resultiert in einem Verlust der zeitlichen Informationen durch die Anwendung der FFT. Für Aufgaben wie die Spracherkennung ist es jedoch von grundlegender Bedeutung, nicht nur die im Signal vorhandenen Frequenzen zu identifizieren, sondern auch den Zeitpunkt ihres Auftretens zu bestimmen. Hier schafft das Spektrogramm Abhilfe, da es die zeitliche Abfolge der Frequenzen sichtbar macht. Diese Fähigkeit ist insbesondere für das Erkennen der Sequenz gesprochener Wörter in einem Satz von Bedeutung (Chaudhary, 2020). Somit verknüpft das Spektrogramm zeitliche und frequenzbezogene Informationen, was es zu einem wichtigen Instrument für die Spracherkennung und andere Machine Learning-Anwendungen macht. Abbildung 7 zeigt ein Beispiel eines Spektrogramms, das im Rahmen dieser Arbeit entwickelt wurde. Das Spektrogramm wurde unter Verwendung der Python-Bibliotheken `PyQt5` für die Echtzeit-Visualisierung und `soundevice` für den Zugriff auf die Audio-Hardware generiert.

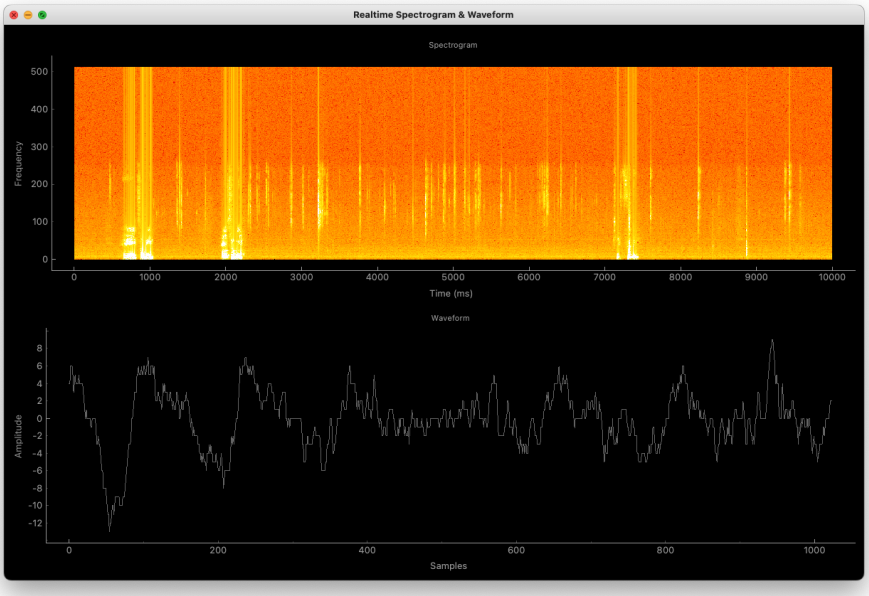

Abbildung 7: Spektrogramm

3 Stand der Forschung

Die Forschung im Bereich der Spracherkennung ist ein aktives Gebiet. Eine Suche auf Google Scholar nach dem Begriff *Speech Recognition* ergibt um die 3 Millionen Ergebnisse. Die Jahre 2010 bis 2020 haben laut (Hannun, 2021) einen grossen Fortschritt in der Spracherkennung erlebt. Dieser Fortschritt ist vor allem auf die Verwendung von Deep Learning zurückzuführen. In der Allgemeinheit ist Spracherkennung vor allem durch die Sprachassistenten von Apple, Google und Amazon bekannt. Apple veröffentlichte Siri im Jahr 2011 mit dem Release von iOS 5.

3.1 Zeitliche Entwicklung der Spracherkennung

TODO: - Abbildung mit einer Timeline diverser Sprachassistenten und deren Release-Daten aus (Hannun, 2021)

3.2 Komparative Analyse von Sprachassistenten

TODO: - (Matarneh et al., 2017) bietet eine komparative Analyse von Sprachassistenten. Die Analyse "Voice control implementation may be conditionally divided into parts: speech, recognition, translation, and execution of commands (Fig. 1)."

3.3 Funktionsweise von Siri

TODO: Offenlegung der Funktionsweise von Siri. Wie funktioniert Siri? (Siri-Team, 2017) und (Apple, 2023)

- Untersuchung und Darstellung der Funktionsweise von Siri.
- Nutzung der Quellen Siri-Team, 2017 und Apple, 2023 zur detaillierten Erklärung von Siris Arbeitsweise.
- Überprüfen der Links für zusätzliche Informationen:
  - https://machinelearning.apple.com/research/hey-siri
  - https://machinelearning.apple.com/research/voice-trigger

4 Ideen und Konzepte

Dieses Kapitel beschreibt die Ideen und Konzepte, die für die Umsetzung der Arbeit verwendet werden. Es wird auch auf die verwendeten Technologien eingegangen.

Die Grob Idee ist es im wesentlichen, ein eigenes Modell zu trainieren, welches Triggerwörter erkennt. Dazu wird ein Datensatz erstellt, welcher die Triggerwörter, sowie andere Wörter enthält.

In einem ersten Schritt wird ein eigenes Modell trainiert, welches Triggerwörter erkennt. Es gibt aber auch die Möglichkeit, ein bereits vortrainiertes Modell zu verwenden. Dieses Kapitel

....

5 Methoden

Das Problem dieser Arbeit ist im wesentlichen die Erkennung von Triggerwörtern innerhalb des Kontext einer App. Grundsätzlich ist es unüblich, dass mobile Apps eine integrierte Sprachsteuerungsfunktion anbieten.

6 Realisierung

Das Problem dieser Arbeit ist im wesentlichen die Erkennung von Triggerwörtern innerhalb des Kontext einer App. Grundsätzlich ist es unüblich, dass mobile Apps eine integrierte Sprachsteuerungsfunktion anbieten.

7 Evaluation und Validation

Das Problem dieser Arbeit ist im wesentlichen die Erkennung von Triggerwörtern innerhalb des Kontext einer App. Grundsätzlich ist es unüblich, dass mobile Apps eine integrierte Sprachsteuerungsfunktion anbieten.

8 Ausblick

Das Problem dieser Arbeit ist im wesentlichen die Erkennung von Triggerwörtern innerhalb des Kontext einer App. Grundsätzlich ist es unüblich, dass mobile Apps eine integrierte Sprachsteuerungsfunktion anbieten.

9 Anhang

9.1 Projektmanagement

Das Projektmanagement spielt eine zentrale Rolle in der Vorbereitungsphase meiner Bachelorarbeit und bildet die Grundlage für den Erfolg des gesamten Vorhabens. Dabei geht es nicht nur um die reine Planung, sondern auch um eine effiziente Steuerung und kontinuierliche Kontrolle aller Arbeitspakete und derer Ergebnisse. Die besondere Herausforderung meiner Arbeit liegt darin, das umfangreiche Themengebiet, das für diese Bachelorarbeit relevant ist, innerhalb des engen Zeitrahmens von 14 Wochen sinnvoll und fundiert zu bearbeiten. Das Themengebiet umfasst diverse Bereiche der Informatik. Darunter fallen Audioverarbeitung, maschinelles Lernen, Softwareentwicklung und auch einiges an mathematischem Hintergrundwissen. Daher wurde ein agiles Vorgehensmodell gewählt. Dies bedeutet, dass sowohl die Planung als auch die Umsetzung in iterative Zyklen unterteilt sind. Während es zu Beginn eine grobe Struktur und Zielsetzung gibt, ermöglicht diese Herangehensweise Flexibilität in der Durchführung. Dadurch können Veränderungen oder unerwartete Ereignisse leichter integriert und die Bachelorarbeit fortlaufend optimiert werden.

9.2 Grobplanung

Die Grobplanung zeigt die wichtigsten Meilensteine des Projekts auf. Ausserdem werden die Themengebiete, die für die Bachelorarbeit relevant sind, aufgezeigt.

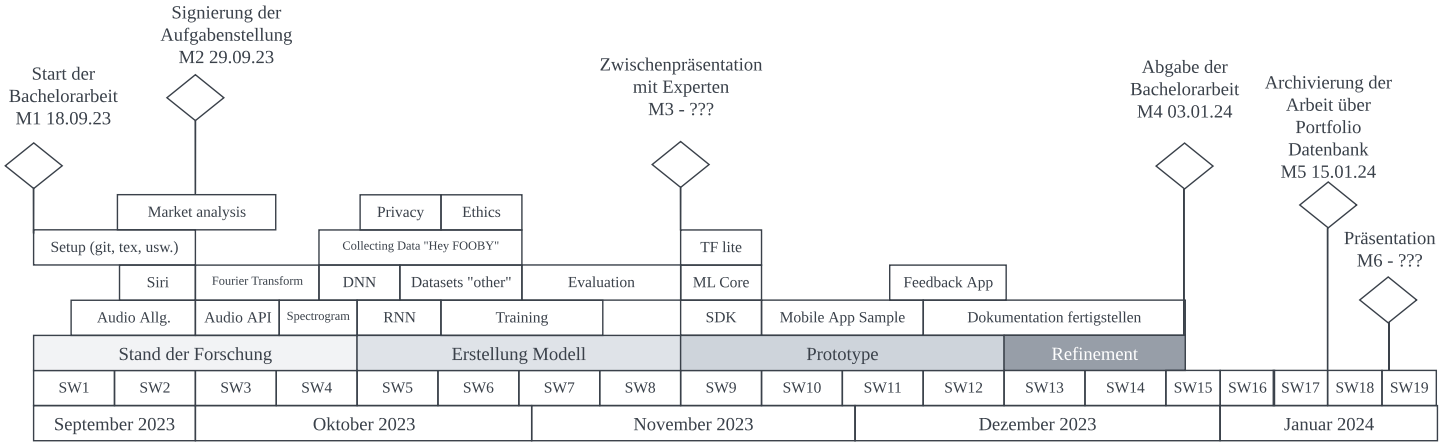

Abbildung 8: Grobplanung

9.2.1 Produkt Backlog

In der Vorbereitungsphase kann ein anfängliches Produkt Backlog als einfache Tabelle dargestellt werden. Ein Beispiel für eine solche Tabelle ist in Abbildung 5 dargestellt.

9.2.2 Risikomanagement

Risikomanagement dient dem Zweck, mögliche Probleme vorwegzunehmen. Die Verwendung von Checklisten, Brainstorming mit den Anspruchsgruppen und die von Erfahrungen aus früheren Projekten sind mögliche Strategien zur Identifikation möglicher Risiken.

Abbildung 9: Tabelle für das anfängliche Product Backlog

Tabelle 1: Beispiel-Tabelle für Risikomanagement

Kopf 1	Kopf 2	Kopf 3
Wert 1	Wert 2	Wert 3
Wert 4	Wert 5	Wert 6

Tabelle 2: Eine einfache Tabelle

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

Apple, M. L. J. (2023, August). *Voice Trigger System for Siri* [Zugriffsdatum: 10. September 2023]. https://machinelearning.apple.com/research/voice-trigger

Chaudhary, K. (2020). *Understanding Audio data, Fourier Transform, FFT and Spectrogram features for a Speech Recognition System* [Zugriff am: 06.10.2023]. https://dropsofai.com/understanding-audio-data-fourier-transform-fft-and-spectrogram-features-for-a-speech-recognition-system/

Deloraine, E. M., & Reeves, A. H. (1965). The 25th anniversary of pulse code modulation. *IEEE Spectrum*, *2*(5), 56–63. https://doi.org/10.1109/MSPEC.1965.5212943

Hannun, A. (2021). The History of Speech Recognition to the Year 2030.

Hansen, E. W. (2014, September). *Fourier Transforms: Principles and Applications*. Wiley.

Matarneh, R., Maksymova, S., Lyashenko, V. V., & Belova, N. V. (2017). Speech Recognition Systems: A Comparative Review [Submission Date: 13-10-2017, Acceptance Date: 27-10-2017]. *OSR Journal of Computer Engineering (IOSR-JCE)*, *19*(5), 71–79. https://doi.org/10.9790/0661-1905047179

Siri-Team. (2017, Oktober). *Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant* [Zugriffsdatum: 10. September 2023]. https://machinelearning.apple.com/research/hey-siri

Somberg, G., Davidson, G., & Doumler, T. (2019). A Standard Audio API for C++: Motivation, Scope, and Basic Design ["C++ is there to deal with hardware at a low level, and to abstract away from it with zero overhead." – Bjarne Stroustrup, Cpp.chat Episode #44]. *Programming Language C++*.

Tarr, E. (2018). *Hack audio : : an introduction to computer programming and digital signal processing in MATLAB* (1st edition). Routledge.

Weitz, P. D. E. (2023). *Fourier-Analysis in 100 Minuten* [Zugriff am: 06.10.2023]. YouTube. https://www.youtube.com/watch?v=zXd743X6I0w

Aufgabenstellung

Integration von Sprachsteuerungstechnologien in Mobile Apps, insbesondere zur Erkennung von Triggerwörtern.

Projektteam

- Student:in: Rubén Nuñez
- Betreuer:in: Herzog
- Firma: Bitforge AG

Auftraggeber

- Firma: Bitforge AG
- Ansprechperson: Stefan Reinhard
- Funktion: Head of Mobile
- Adresse: Zeughausstrasse 39, 8004 Zürich
- Telefon: +41 55 211 02 41
- E-Mail: stefan.reinhard@bitforge.ch
- Website: www.bitforge.ch

Sonstige Bemerkungen

Grundkenntnisse in Machine Learning, speziell im Bereich der Spracherkennung, sowie Erfahrung mit entsprechenden APIs sind erforderlich.