laborbericht.

Grundlagen datenbasierter Methoden & Softwaredesign (MECH-M-DUAL-1-DBM & MECH-M-DUAL-1-SWD)

Sound Improvment

Master Mechatronik

1. Semester

Lehrveranstaltungsleiter: Peter Kandolf

Jahrgang: M 2025

Verfasser: Til Neubauer, Daniel Brose

15. Februar 2026

inhaltsverzeichnis.

1 Einleitung **3**

2 Gesamtarchitektur **4**

2.1 Zentrale Steuerung [main_gui.py] . . . 4

2.2 Backend Signalverarbeitung . . . 5

2.2.1 AudioEngine [audio_engine.py] . . . 5

2.2.2 Verarbeitungskette [apply_processing()] . . . 5

2.2.3 AudioPlayer [audio_player.py] . . . 5

2.3 Filter zur Rauschunterdrückung . . . 6

2.3.1 Bandpassfilter . . . 6

2.3.2 Spactral Gate . . . 7

2.3.3 Equalizer . . . 9

2.4 Frontend -GUI Architektur . . . 10

2.4.1 Hauptfenster [main_window.py] . . . 10

2.4.2 Header [header.py] . . . 10

2.4.3 Main Content Container [main_content.py] . . . 10

2.4.4 Input Frame [input_frame.py] . . . 10

2.4.5 Pipeline Frame [pipeline_frame.py] . . . 10

2.4.6 Output Frame [output_frame.py] . . . 10

2.4.7 Gestaltungskonzept . . . 10

2.5 Bedienungsanleitung . . . 11

2.6 Docker . . . 12

abbildungsverzeichnis.

1 Geamtstrucktur . . . 4

2 Bandpassfilter . . . 6

3 Noise Floor . . . 7

4 Rauschen aus Tonsignal herausfiltern 1 . . . 7

5 Rauschen aus Tonsignal herausfiltern 2 . . . 8

6 Rauschen aus Tonsignal herausfiltern 3 . . . 8

7 Equalizer Funktion . . . 9

tabellenverzeichnis.

1 einleitung

Im Rahmen des Projekts „Audio Signal Processing 2“ wurde eine modulare Desktop Anwendung zur Analyse, Wiedergabe und Verarbeitung von Audiosignalen entwickelt.
Die Software ermöglicht:

- Import von Audio-Dateien (WAV)
- Darstellung von Zeit- und Frequenzbereich (Waveform & FFT)
- Echtzeitnahe Signalverarbeitung
- Anwendung mehrerer Digitaler Filter:
  - Bandpass (Butterworth)
  - Spectral Gate (Rauschreduktion)
  - 10 Band Equalizer
- Vergleich von Input- und Output-Signal
- Playback mit Timeline-Scrubber
- Export des verarbeiteten Signals

Ziel war die saubere Trennung von GUI (Frontend) und Signalverarbeitung (Backend) sowie eine klar strukturierte und erweiterbare Architektur.

2 gesamtarchitektur

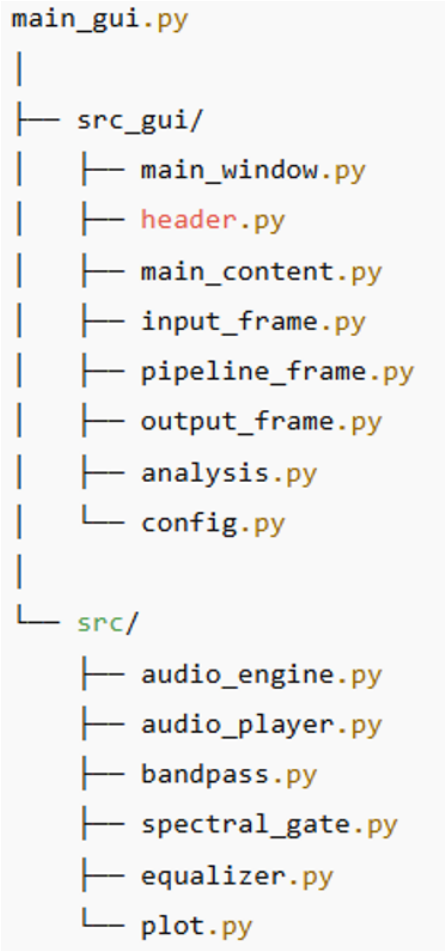

Abbildung 1: Geamtstrucktur

2.1 ZENTRALE STEUERUNG [MAIN_GUI.PY]

Die main_gui.py ist das Herzstück des Frontends. In ihr wird sowohl das Frontend korrekt aufgerufen, sowie auch das Backend eingebunden. Diese Datei bildet den Einstiegspunkt der Anwendung. Ihre Aufgaben sind wie Folgt:

- Erstellung des Hauptfensters
- Instanziierung der [AudioEngine]
- Aufruf und Aufbau aller GUI-Koponenten
- Kopplung von Timeline-Slider und Engine
- Zyklisches Update der aufgerufenen Fester

2.2 BACKEND SIGNALVERARBEITUNG

2.2.1 AudioEngine [audio_engine.py]

Die AudioEngine ist das Herzstück der Anwendung. Ihre Aufgabe ist:

- Verwaltung von Input- und Output-Player
- Speicherung von Signalzuständen
- Verwaltung der Filter-States
- Verarbeitungskette
- Aktualisierung des Output-Signals

2.2.2 Verarbeitungskette [apply_processing()]

Die Aufgaben der Verarbeitungskette sind:

- Input-Signal kopieren
- Anwendung der Filter (falls angewählt)
- Output-Player aktualisieren
- Callback zur GUI auslösen

2.2.3 AudioPlayer [audio_player.py]

Der Audioplayer ist für die Audioausgabe zuständig. Die Funktionen beinhalten:

- Audio laden
- Playback über (sounddevice)
- Stream-Callback für kontinuierliche Ausgabe
- Seek-Funktion
- Zeitrückgabe

Es werden verwendet pydub für MP3, sowie soundfile für WAV und FLAC Dateien

2.3 FILTER ZUR RAUSCHUNTERDRÜCKUNG

2.3.1 Bandpassfilter

Eine herkömmliche Methode zur Verbesserung von Audiosignalen ist der Bandpassfilter. Bei einem Bandpassfilter kann man angeben in welchem Frequenzbereich sich das Signal befinden soll, alle Frequenzen, die außerhalb dieses Bereiches liegen werden abgeschnitten. Alle Funktionen, die mit dem Bandpassfilter in Verbindung stehen sind in der Datei banpass.py abgespeichert.

Technische Umsetzung

Um den Bandpassfilter zu realisieren, wurden die Bibliothek scipy verwendet. Der Bandpassfilter beruht hauptsächlich auf den Funktionen butter und lfilter von scipy.signal.

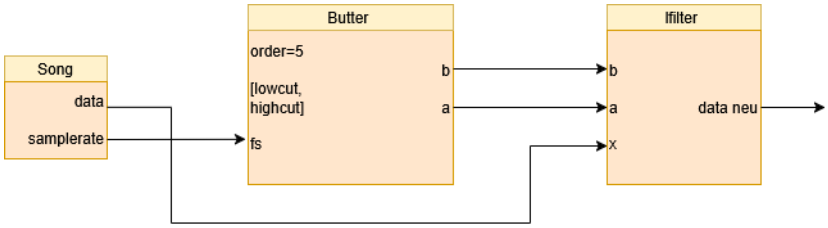

Abbildung 2: Bandpassfilter

Die Funktion lfilter basiert auf der Methode eines IIL Filters und setzt den Bandpassfilter um. Die Funktion butter Liefert die Koeffizient für lfilter. Butter werden die obere und untere grenze des Bandpassfilters sowie der Samplerate des Liedes übergeben. Die Rückgabe Werte von butter werden der lfilter Funktion zusammen mit dem Tonsignal übergeben. Das Ergebnis ist ein Bandpassfilter, der nicht scharf an den Frequenzgrenzen abschneidet, sondern einen glatten Übergang bietet.

2.3.2 Spactral Gate

Spactral Gate ist der Hauptfilter der das Rauschen eines Tonsignals unterdrücken soll. Die Funktionen des Filters sind in der Datei spectral_gate.py gespeichert. Der Filter besteht aus zwei Schritten. Zuerst wird eine Maske des Rauschsignals erzeugt. Dann kann diese Maske auf das Tonsignal mit rauschen angewandt werden, und Frequenzen herausgefiltert werden, die dem Rauschen sehr ähnlich sind. Der Filter benötigt ein Tonsignal, das möglichst nur Rauschen enthält. Im Falle der Rauschunterdrückung bei alten Schallplatten ist dies vorteilhaft, da bevor ein Lied startet, normalerweise die Schallplatte sich schon dreht und ein charakteristisches Rauschen der Schallplatte zu hören ist. Dieses könnte zum Erstellen der Rauschmaske verwendet werden.

Technische Umsetzung: Maske vom Rauschens erstellen

Die Funktion noise_profile() kümmert sich um das erstellen der Maske des Rauschens bzw. des Noise Floors.

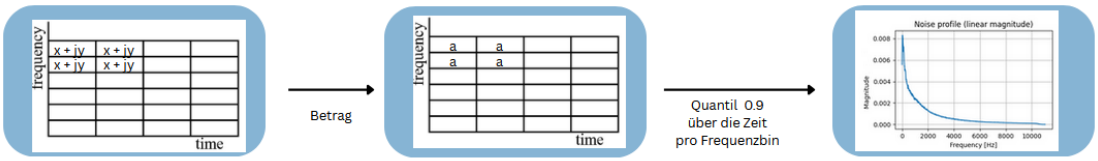

Abbildung 3: Noise Floor

Es wird eine Short-Time Fourier Transformation auf das Rauschsignal angewandt. Hierbei wird die Bibliothek scipy verwendet. Da uns die Stärke der vorkommenden Frequenzen interessiert, wird im Anschluss der Betrag der einzelnen Frequenz Bins gebildet. Um einen typischen Wert des Rauschens pro Frequenzbin zu erhalten, wird pro Frequenzbin über die Zeit das Quantil von 0.9 genommen.
Das Ergebnis der noise_floor() Funktion ist die Typische Stärke des Rauschsignals pro Frequenzbin.

Technische Umsetzung: Rauschen aus Tonsignal herausfiltern

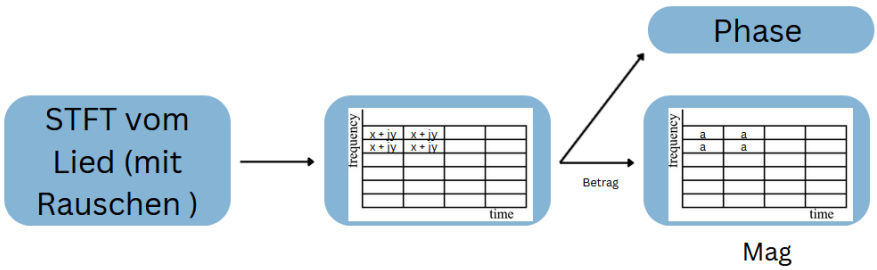

Abbildung 4: Rauschen aus Tonsignal herausfiltern 1

Zuerst wird eine Short-Time Fourier Transformation (STFT) auf das Tonsignal angewandt. Im Anschluss wird der Betrag genommen, da uns die Stärke der einzelnen Bins interessiert und in der Variablen Mag gespeichert. Jedoch wird auch die Phase jedes Bins abgespeichert. Diese wird benötigt, um vom Frequenzbereich in den Zeitbereich zurückzurechnen.

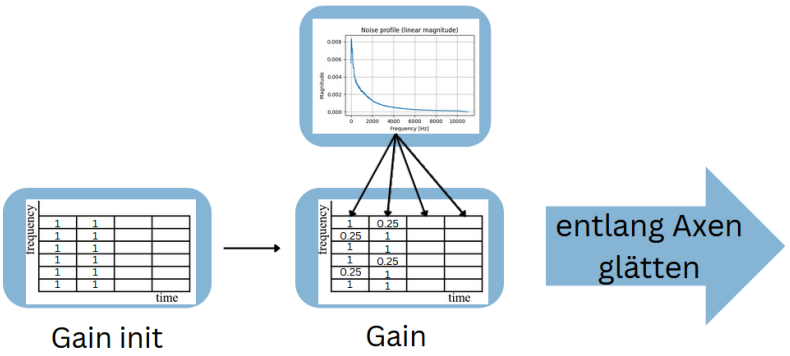

Abbildung 5: Rauschen aus Tonsignal herausfiltern 2

Im Anschluss wird Gain initialisier,t der das gleiche Format wie das Tonsignal nach der STFT hat. Es werden alle Werte auf 1 gesetzt. Um das Rauschen zu minimieren wird nun über jeden Time-Bin jeder Frequenz Bin mit dem Frequenz Bin des Noise Floors verglichen. Ist der Wert kleiner wie im Noise Floor, wird der der Wert im Gain auf 0.25 gesetzt. Im Anschluss wird der Gain entlang der Zeit- und der Frequenz-Achse geglättet.

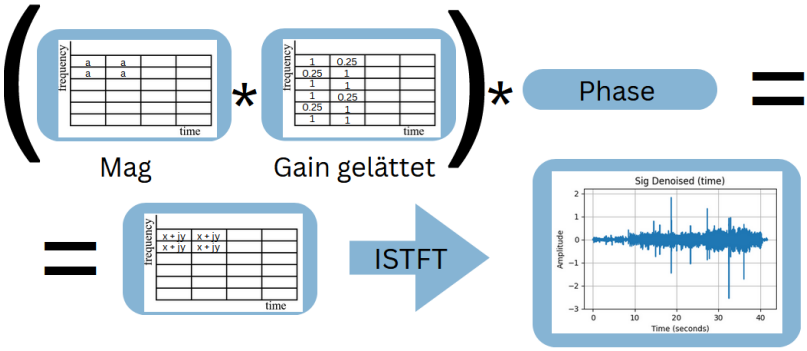

Abbildung 6: Rauschen aus Tonsignal herausfiltern 3

Um wieder in den Zeitbereich zu kommen und das Rauschen zu unterdrücken werden Mag und Gain multipliziert und im Anschluss noch mit der zuvor abgespeicherten Phase multipliziert. Von diesem Ergebnis kann man die Inverse der Short-Time Fourier Transformation bilden und erhält wieder ein Zeitsignal. Dieses Zeitsignal hat nun deutlich weniger Rauschen.

2.3.3 Equalizer

Zur Grundfunktionalität jedes Audiobearbeitungsprogramms zählt ein Equalizer. Bei einem Equalizer kann angegeben werden, welche Frequenzen(Bereiche) verstärkt oder gedämpft werden sollen. Alle Funktionen, die mit dem Equalizer in Verbindung stehen, sind in der Datei equalizer.py gespeichert.

Technische Umsetzung eq_fkt()

Der Equalizer besteht aus zwei Funktionen. Die Funktion eq_fkt() erhält als Input die Verstärkung bzw. Dämpfung für die einzelnen Kanäle. Es gibt 10 Inputkanäle: 31Hz, 62Hz, 125Hz, 250Hz, 500Hz, 1kHz, 2kHz, 4kHz, 8kHz, 16kHz. Der Input erfolgt in dB. Die Funktion erstellt eine Gleichmäßige Verlauf aus den Inputvariablen. Hierbei wird die Funktion PchipInterpolator von scipy.interpolate verwendet.

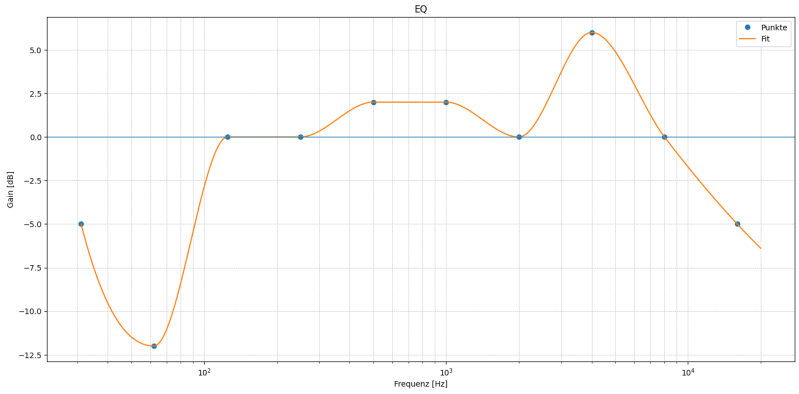

Abbildung 7: Equalizer Funktion

Die Abbildung 7 zeigt das Ergebnis der Funktion eq_fkt() mit den Inputvariablen:

```
eq_fkt(e32=-5, e64=-12, e125=0, e250=0, e500=2, e1k=2, e2k=0, e4k=6, e8k=0, e16k=-5)
```

Wie zusehen ist wird aus den Eingabepunkten ein gleichmäßiger Verlauf mit einer Logarithmischen Skala auf der X-Achse generiert.
Der Rückgabewert der Funktion sind die x-Werte mit np.logspace() und den dazugehörigen y-Werten der interpolierten Funktion.

Technische Umsetzung apply_eq()

Die Funktion apply_eq() setzt den Equalizer um. Als Inputvariablen bekommt sie das Tonsignal mit der Samplrate so wie die rückgabewerte der eq_fkt() Funktion.
Das Tonsignal wir mittels np.fft.rfft() in den Frequenzbereich transformiert. Es wird die Real FFT verwendet da die „Intensität“ der einzelnen Frequenzen relevant ist. Damit es zu keinen Fehlern zwischen den Frequenz Bins der FFT und der Verstärkungsfunktion die mittels eq_fkt() berechnet wird kommen kann wird erneut interpoliert. Die Verstärkung oder Dämpfung wird mittels einer Maske im Frequenzbereich angewandt. Im Anschluss wird mit np.fft.irfft() wieder in den Zeitbereich zurückgerechnet.

2.4 FRONTEND -GUI ARCHITEKTUR

Das Frontend ist komplett mit TKinter umgesetzt. TKinter ist eine Standard-GUI-Bibliothek in Python. Es ist Widget-basiert und Event-getrieben. Es unterstützt den Aufbau über grid, pack und place. Für diese GUI wurde für das Hauptlayout hauptsächlich grid verwendet. Für interne Widgets meistens pack.

2.4.1 Hauptfenster [main_window.py]

Das Hauptfenster erstellt das Hauptfenster mit drei Zeilen. Den Header, den Main Content Container und den Analyse Container.

2.4.2 Header [header.py]

Der Header beinhaltet den Titel

2.4.3 Main Content Container [main_content.py]

Der Maincontent erstellt ein Frame mit drei gleichgroßen Spalten für den Input, die Pipeline und den Output

2.4.4 Input Frame [input_frame.py]

Der Inputframe enthält alle Funktionen zum Einlesen einer Audiodatei, sowie Labels für die Metadaten und auch die Plots des unveränderten Signals. Ebenso besteht die Möglichkeit zum Abspielen des originalen Soundtracks.

2.4.5 Pipeline Frame [pipeline_frame.py]

Das Pipelineframe enthält alle Anwendungsmöglichkeiten der Soundbearbeitung. Über ein Dropdown Menü kann man zwischen den Filtern Bandpass, SpectralGate und Equalizer auswählen und diese individuell konfigurieren und aktivieren.

2.4.6 Output Frame [output_frame.py]

Das Outputframe ist für die Ausgabe des verarbeiteten Signals zuständig. Es ist aufgebaut wie der Inputframe. Es bietet die Möglichkeit das neue Audiosignal abzuspielen und zu speichern. Ebenso werden die Pots des neuen Signals angezeigt. Allerdings ist dies erst möglich, nachdem ein Input geladen ist und mindestens ein Filter aktiviert wurde.

2.4.7 Gestaltungskonzept

Die Gestaltung ist dunkel gewählt mit Farblicher Hinterlegung bei angewählten Buttons. Die Konfigurationen werden zentral in der Konfigurationsdatei [config.py] verwaltet.

2.5 BEDIENUNGSANLEITUNG

1. Die GUI wird über *pdm run main_gui.py*
2. Audio importieren über den „Import Audio“ Button
3. Filter wählen über das Dropdown im Pipeline-Bereich
4. Parameter einstellen (Frequenzen & EQ-Slider)
5. Filter Aktivieren über den „Apply“ Button
6. Wiedergabe des Inputs oder Outputs über die „Play / Pause“ Buttons
7. Timeline verwenden. Über die „Slidebar“ an die interessanten Stellen scrollen
8. Export der Audiodatei über den „Save Audio“ Button

2.6 DOCKER

Von dem Projekt wurde ein Docker Image erstellt. Da bei dem Projekt die Bibliothek tkinter und verschiedene Audiobibliotheken verwendet wurden, war das erstellen eines Docker Image etwas komplizierter. Damit die GUI so wie die Tonausgabe funktionieren musste auf eine kombination von Docker mit Ubuntu zurückgegriffen werden. Eine detaillierte Anweisung zum Starten des Dockers in StartDocker.md enthalten. Das Docker Image ist in der mygui_latest.tar enthalten.

Technische Umsetzung

Damit nicht alle Dateien aus dem Projektordner im Docker erscheinen wurde eine .dockerignore Datei erstelle, die zum Großteil der .gitignore Datei ähnelt.
Es wurde das Dockerfile erstellt das die Abläufe für das Docker Image enthält. Hierbei mussten die Bibliotheken tkinter und sounddevice extra behandelt werden. Des weiteren werden in dem Dockerfile die verwendeten Bibliotheken des Projektes behandelt, die in diesem Projekt von PDM gemanagt werden. Zuletzt wird die GUI über *pdm run* gestartet.