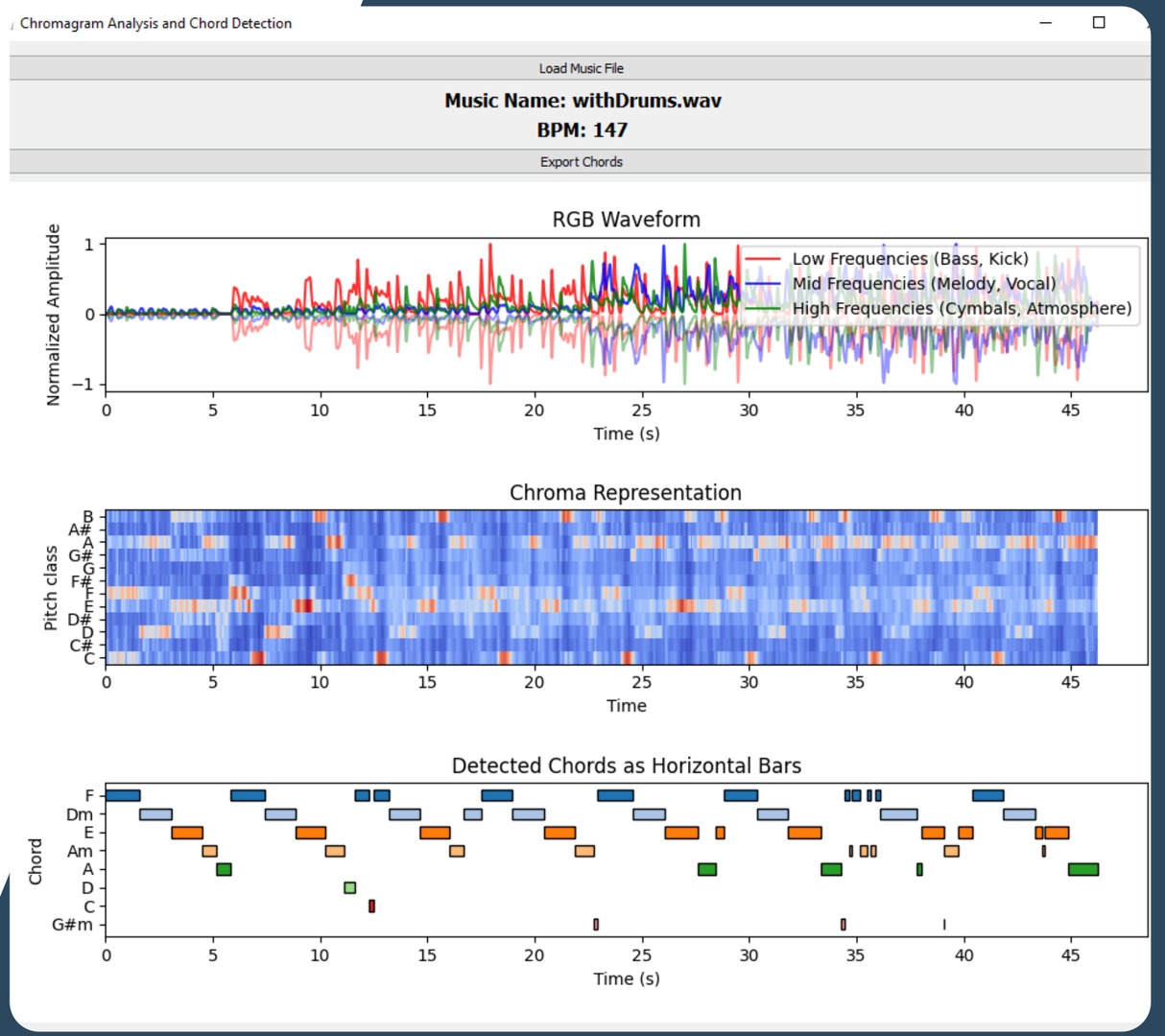
CHORD RECOGNITION USING CHROMAGRAM ANALYSIS AND SIGNAL PROCESSING

GROUP-9

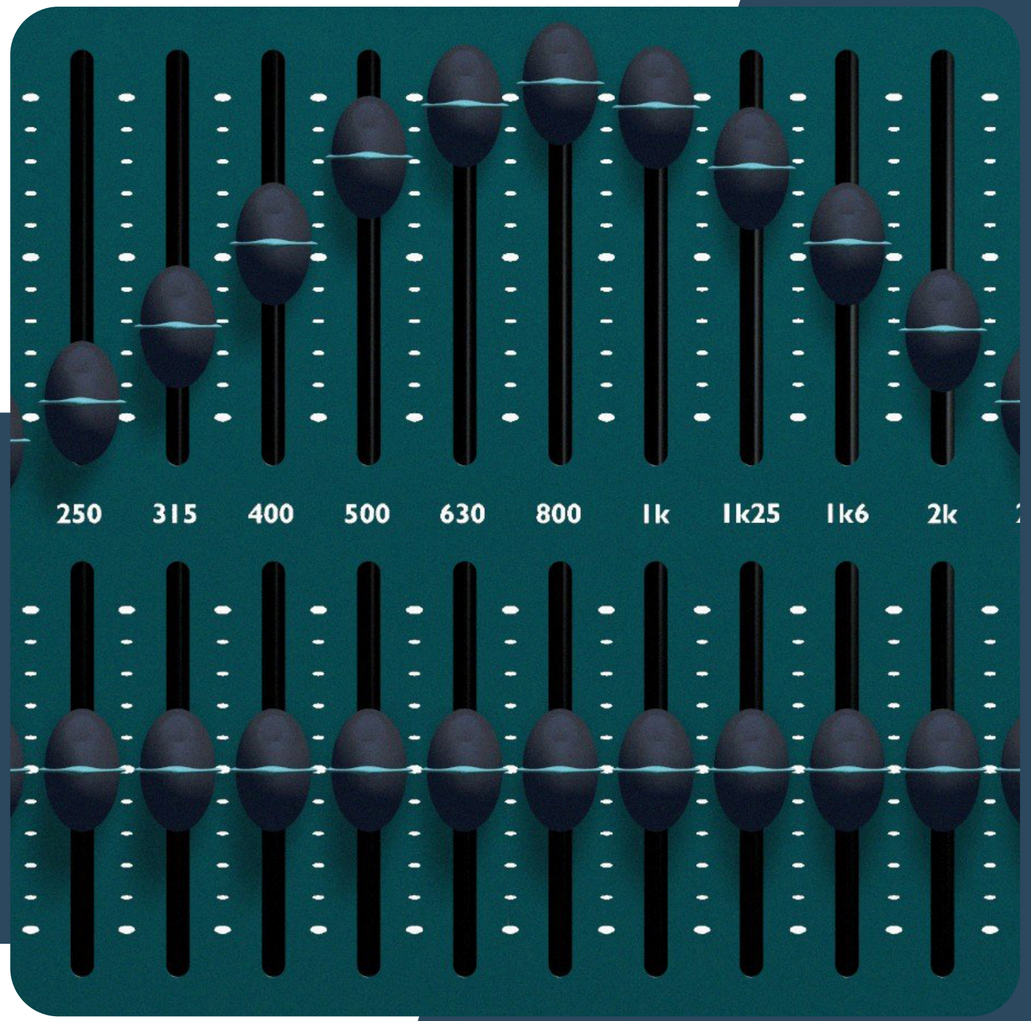
Introduction

- **Definition of chord**
  - A group of (typically three or more) notes sounded together, as a basis of harmony
- **Importance of Music Analysis**
  - Applications: Automatic transcription, karaoke systems, music recommendation
- **Focus on Chord Recognition**
  - Understanding musical structure
- **Motivation**
  - Growing demand for tools like Chordify
  - Personal interest in the music industry

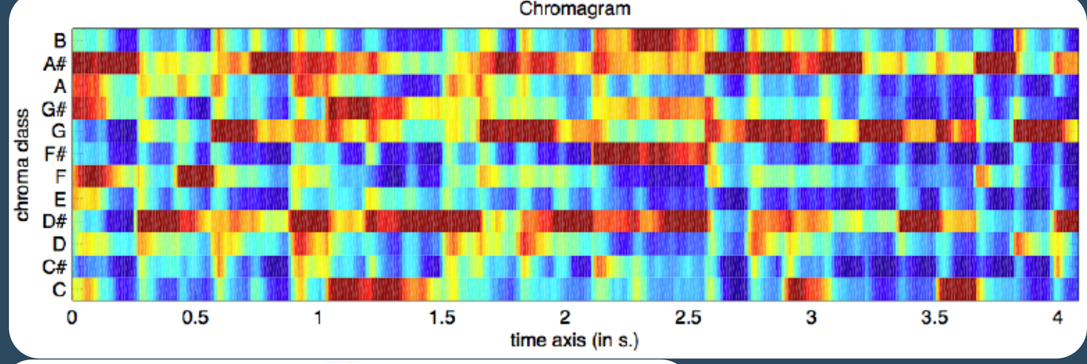


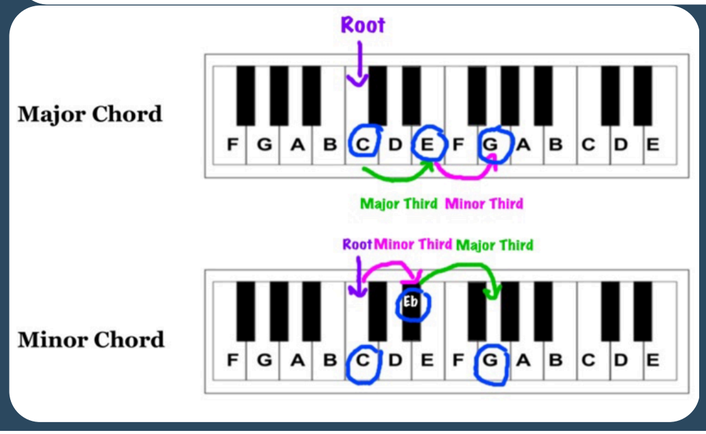
Signal Processing & CQT

- Preprocessing Audio:
  - Resampling to 44.1 kHz, converting to mono
- Constant-Q Transform (CQT):
  - Generates chroma features
  - Advantages over STFT for musical signals
- Chromagram Enhancement:
  - Normalization and smoothing to detect chords accurately



Methodology
Overview





1st Main Step

Signal Processing & Chromagram Generation

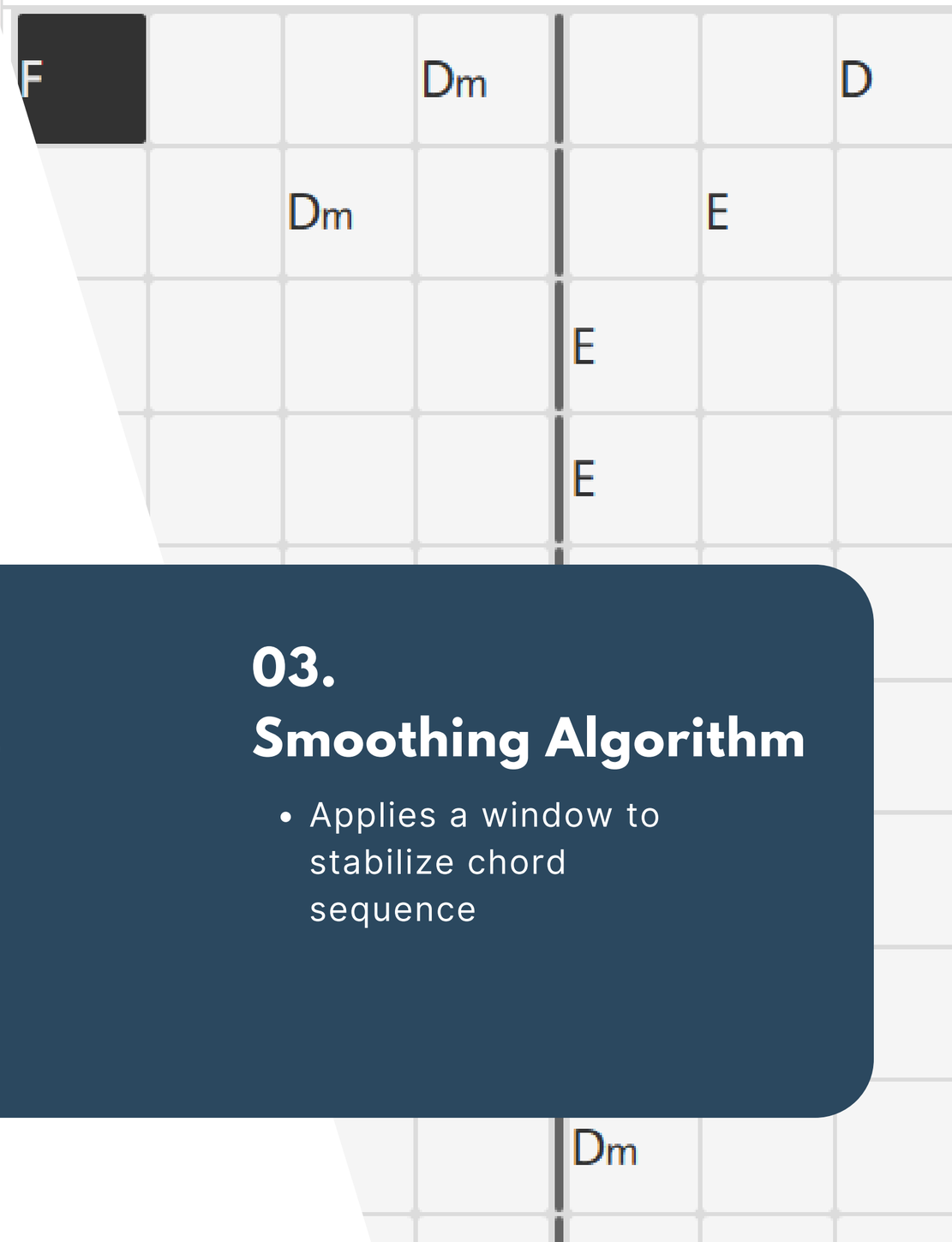
2nd Main Step

Chord Detection via Templates

Tools & Libraries

Python, Librosa, Matplotlib, PyQt5

Chord Detection



01. Chord Templates

- 12 major and minor chords as 12-element vectors

02. Detection Process

- Compare chromagram frames with templates using dot product
- Assign chord with highest similarity

03. Smoothing Algorithm

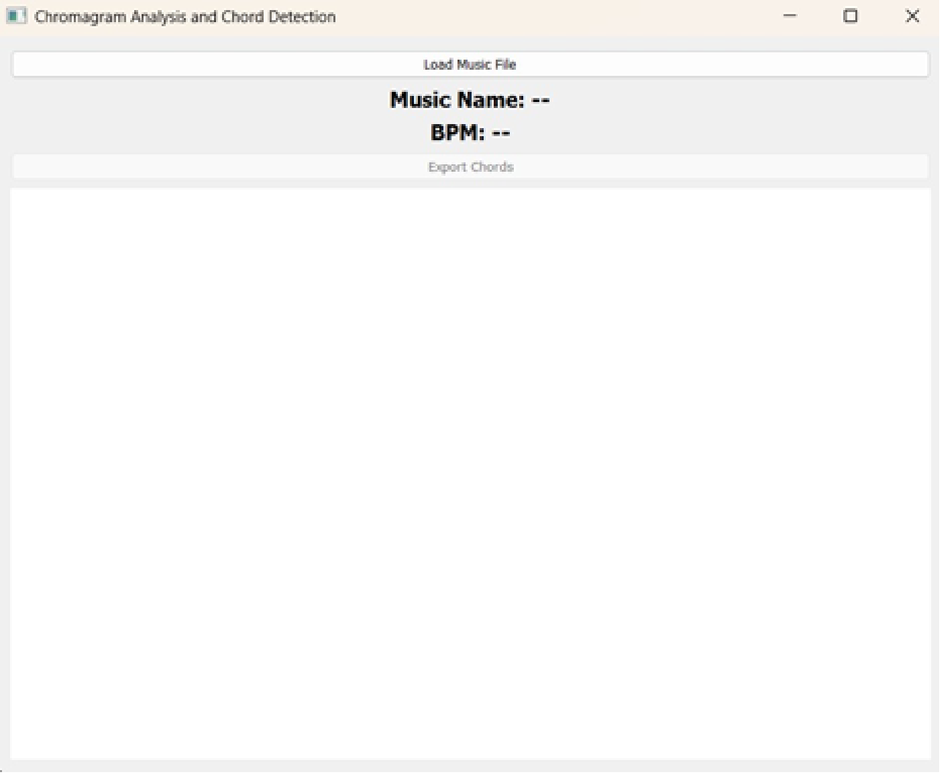
- Applies a window to stabilize chord sequence

Implementation & Visualization

BPM: 71		
Start Time	End Time	Chord
0	1.428027	Am
1.428027	3.738413	Em
3.738413	4.89941	F
4.89941	7.128526	G
7.128526	8.266304	Am
8.266304	10.94821	Em
10.94821	11.94667	Am
11.94667	14.00163	A
14.00163	15.13941	Am
15.13941	17.43819	Em
17.43819	18.54113	F
18.54113	20.67737	G
20.67737	21.94286	Am
21.94286	24.28807	Em
24.28807	25.39102	F
25.39102	27.62014	G
27.62014	28.81596	Am
28.81596	31.1844	Em
31.1844	32.3454	F
32.3454	34.42358	G
34.42358	35.61941	Am
35.61941	37.94141	Em
37.94141	39.14884	F
39.14884	41.31991	G
41.31991	43.03819	Am

- Figure 3: Exported chords in CSV

- Implementation Steps
  - Load and preprocess audio files
  - Compute chromagram using Librosa
  - Detect chords and apply smoothing
  - Visualize with Matplotlib
- GUI with PyQt5:
  - User-friendly interface for loading files and exporting results
- Visuals:
  - Figure 1: Project main menu
  - Figure 2: Processed music file
  - Figure 3: Exported chords in CSV

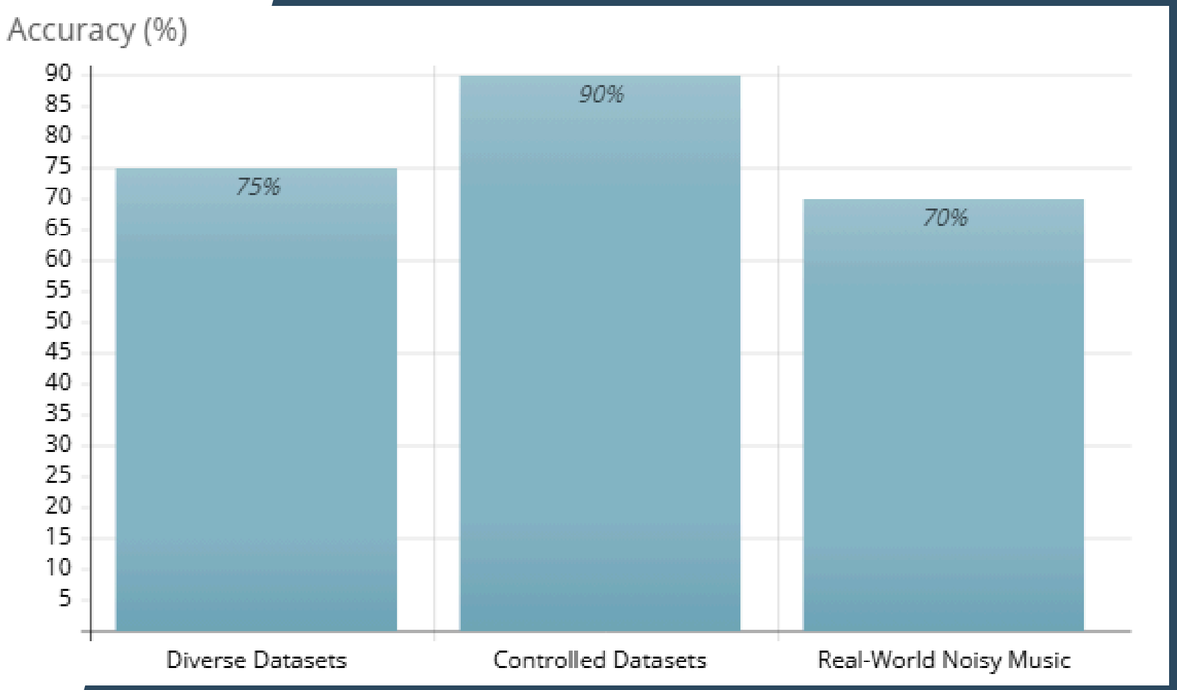


◦ Figure 1: Project main menu

◦ Figure 2: Processed music file

Results & Discussion

- Accuracy Rates
  - 65–75% on diverse datasets
  - Up to 85–90% on controlled datasets
  - 50–70% on real-world noisy music

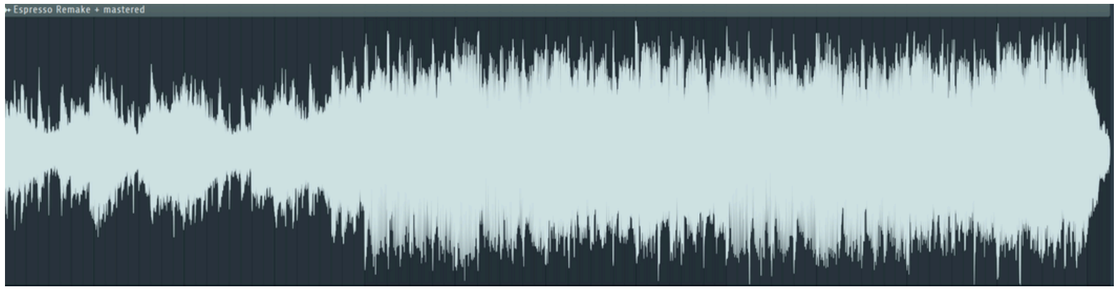


- Comparison:
  - Advanced ML algorithms achieve ~95% accuracy
  - Achieved high accuracy with basic signal processing

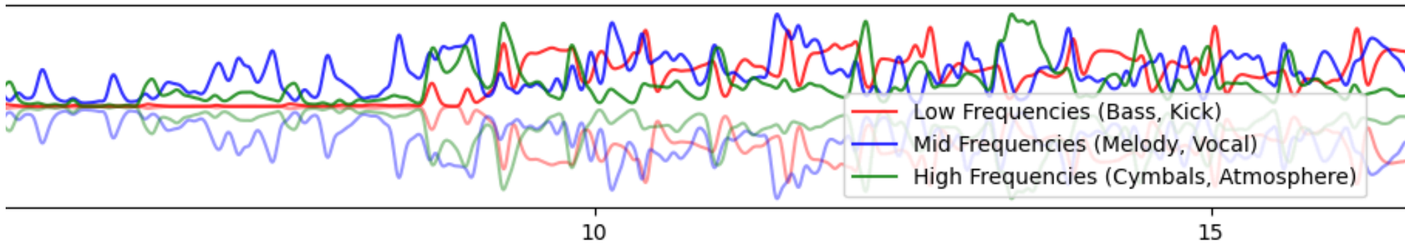
- Challenges:
  - Noise and complex music structures


hani
YouTube

Traditional Waveform



Our Special Waveform



SERUM(VST)

FL Studio (DAW)

Voxengo SPAN

YouTube **ESPRESSO REMAKE**

References

- Brian McFee et al., ”librosa: Audio and Music Signal Analysis in Python,” Proceedings of the 14th Python in Science Conference, 2015.
- librosa Documentation
- matplotlib Documentation
- PyQt5 Documentation
- M. Mueller, Fundamentals of Music Processing Audio, Analysis, Algorithms, Applications. Springer Verlag, 2015.
- Caio Miyashiro, Hidden Markov Models for Chord Recognition, Pydata 2019 Conference


```
}
in enumerate(notes):
_template = np.zeros(12)
intv in major_intervals:
major_template[(i + intv) % 12] = 1
ds[f"{n}"] = major_template

or_template = np.zeros(12)
 intv in minor_intervals:
  minor_template[(i + intv) % 12] = 1
ords[f"{n}m"] = minor_template
n chords


detection function
d_detection(chroma, smoothing_window):
rds = get_chord_templates()
rd_names = list(chords.keys())
ord_templates = np.array([chords[ch] for ch in chord_names])
ord_templates = chord_templates / np.linalg.norm(chord_templates, a

etected_chords = []
or frame in chroma.T:
    similarities = chord_templates.dot(frame)
    best_chord_index = np.argmax(similarities)
    detected_chords.append(chord_names[best_chord_index])

# Smooth chord sequence to remove rapid fluctuations
if len(detected_chords) > 0:
    smoothed_chords = []
    for i in range(len(detected_chords)):
        start = max(0, i - smoothing_window)
        end = min(len(detected_chords), i + smoothing_window + 1)
        window_chords = detected_chords[start:end]
        # Choose the most common chord in the window
```