Aplikacja okienkowa do analizy cech sygnału audio w dziedzinie czasu

Jan Opala

March 2025

Spis treści

1 Wstęp

Aplikacja okienkowa Audio Analyzer służy do wyświetlania cech sygnału dźwiękowego na poziomie ramki (wykresy głośności, Short Time Energy, Zero Crossing Rate, określa częstotliwość tonu podstawowego (na podstawie funkcji autokorelacji i AMDF) i wyświetla statystyki dotyczące głośności na poziomie klipu. Co więcej, program pozwala na analizę kilku plików dźwiękowych jednocześnie, zmianę analizowanych parametrów w czasie rzeczywistym, zapisywanie cech dźwięku do pliku o rozszerzeniu `.csv` oraz o wykrywanie fragmentów: ciszy, dźwięcznych, bezdźwięcznych, mowy i muzyki.

2 Implementacja

2.1 Aspekty techniczne

Projekt został napisany w języku Python w wersji 3.9.5. Cała aplikacja składa się wyłącznie z jednego pliku `App.py`.

W projekcie wykorzystano następujące biblioteki:

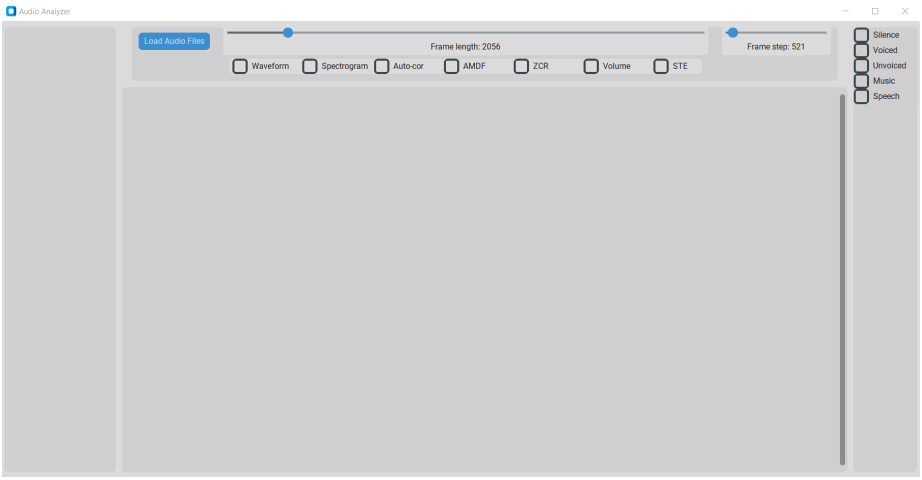
- **Customtkinter** – biblioteka służąca do przygotowania interfejsu graficznego i ogólnego wyglądu programu
- **Librosa** – biblioteka używana do wyciągnięcia surowych danych do dalszej analizy z sygnałów dźwiękowych i do przygotowania wykresów (przebieg czasowy pliku, spektrogram)
- **Tkinter** – podstawowa wersja Tkintera wykorzystana w celu przesyłania plików z dysku
- **PIL** – biblioteka wymagana do wyświetlania wykresów za pomocą odczytywania tymczasowo zapisywanych plików `.png`
- **Matplotlib** – biblioteka używana do tworzenia wykresów
- **NumPy** – biblioteka służąca do operacji na wektorach w celu uzyskiwania różnych cech przez przekształcanie amplitudy
- **os** – umożliwia tymczasowe zapisywanie wykresów w pamięci podręcznej
- **csv** - umożliwia zapis cech dźwięku w formacie `.csv`
- **sounddevice** – biblioteka umożliwiająca odtwarzanie nagrań w aplikacji

2.2 Interfejs aplikacji

Interfejs graficzny aplikacji został zbudowany w bibliotece `customtkinter` i składa się z następujących paneli

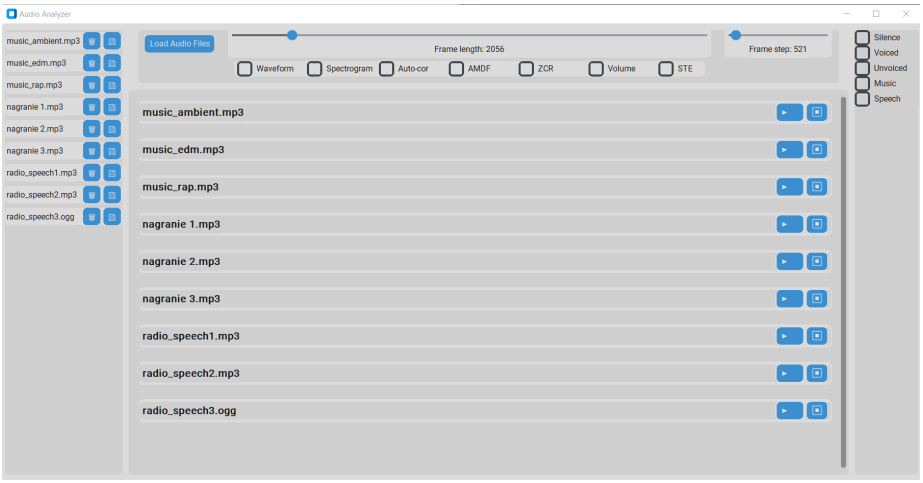
- LeftPanel – panel na którym wyświetlane są kafelki informujące o aktualnie analizowanych nagraniach. Z tego widoku niechciane pliki mogą zostać skasowane lub można eksportować cechy dźwięku do pliku o rozszerzeniu `.csv`
- RightPanel – panel zawierający checkboxy do wybrania które cechy użytkownik chce, żeby były zaznaczone na wykresieprzebiegu czasowego (cisza, fragmenty dźwięczne/bezdźwięczne, mowa, muzyka)
- TopPanel – panel w którym znajdują się suwaki i checkboxy pozwalające wybierać które cechy dźwięku użytkownik chce analizować. Dodatkowo znajduje się przycisk pozwalający wgrywanie plików audio do programu
- **AudioPanel** – każdy z obiektów tej klasy zawiera: nazwę analizowanego nagrania, przyciski umożliwiające odsłuchanie/wyłączenie nagrania, a także opcjonalnie wykresy przedstawiające cechy dźwięku na poziomie ramki, jak również (w postaci informacji tekstowej) cechy głośności na poziomie klipu
- **AudioViewer** – panel możliwy do przewijania góra-dół na którym wyświetlane są obiekty AudioPanel, które są aktualizowane wraz z każdą zaznaczoną opcją. W obiekcie tej klasy inicjalizowane są wszystkie pozostałe panele wymienione wcześniej

Powyżej opisany interfejs wygląda następująco:



Rysunek 1: Wygląd interfejsu aplikacji po jej uruchomieniu

Po załadowaniu do programu wielu plików możemy zobaczyć kafelkową strukturę LeftPanelu oraz różne AudioPanele widoczne na środku (wykresy wyłączone):



Rysunek 2: Wygląd interfejsu aplikacji po załadowaniu wielu plików dźwiękowych

3 Podstawowe funkcjonalności

Podstawowa funkcjonalność aplikacji może zostać opisana w następujący sposób:

1. Użytkownik uruchamia program
2. Użytkownik zaznacza które cechy chciałby analizować (domyślnie ustawione: 'Waveform' i 'Spectrogram'). Może również zmienić wielkość i odstęp między ramkami
3. Użytkownik wgrywa jeden lub więcej plików dźwiękowych do aplikacji
4. Wyświetlane są analizowane cechy. Użytkownik może wcisnąć przycisk w celu odtworzenia konkretnego nagrania.
5. Checkboxy po prawej stronie mogą zostać zaznaczone w celu wskazania na wykresie przebiegu czasowego charakterystycznych fragmentów
6. Zbędne nagrania mogą zostać usunięte na lewym panelu
7. Jeśli użytkownik chce pozyskać zbiór danych z cechami danego nagrania (np. w celu analizy statystycznej, zbudowania modoelu uczenia maszynowego) może pobrać plik o rozszerzeniu `.csv` dla każdego nagrania

Powyższe funkcjonalności zostały zaimplementowane w następujący sposób:

3.1 Wgrywanie plików dźwiękowych

Po wciśnięciu przycisku 'Load Audio Files' użytkownik może przesłać jeden lub więcej plików dźwiękowych. Zostało to zaimplementowane w poniższym kodzie: W pierwszej kolejności wciśnięty zostaje przycisk:

```
self.load_button = ctk.CTkButton(self, text="Load Audio
    Files", command=self.on_update_callback)
```

W klasie `AudioAnalyzerApp` w której wywoływana jest cała aplikacja wykonana zostaje linijka:

```
self.audio_viewer.top_panel.load_button.configure(command=
    self.audio_viewer.load_audio_files)
```

Z powyższego wynika, że `on_update_callback` zostaje nadpisane metodą `load_audio_files`, która wygląda następująco:

```
    def load_audio_files(self):
        filepaths = filedialog.askopenfilenames(filetypes=[(
            "Audio Files", "*.wav *.mp3 *.flac *.ogg")])
        if not filepaths:
            return
        for filepath in filepaths:
            try:
                y, sr = librosa.load(filepath)
                panel = AudioPanel(self, filepath, y, sr)
                self.audio_panels.append(panel)
                self.left_panel.add_tile(filepath, panel.
                    export_data, panel.remove)
```

```
            except Exception as e:
                messagebox.showerror("Error", f"Failed to
                    load {os.path.basename(filepath)}: {e}")
```

W powyższym kodzie zaimplementowano następującą funkcjonalność:

1. Program pobiera ścieżkę od użytkownika w której znajdują się pliki dźwiękowe (przy pomocy `filedialog` z biblioteki `Tkinter`
2. Za pomocą biblioteki `Librosa` wyciągnięte zostały tablice NumPy array `y` jako sygnał dźwiękowy i `sr` częstotliwość próbkowania (sampling rate)
3. Tworzony jest panel jako obiekt klasy AudioPanel wykorzystując jako paramentry nazwę pliku, sygnał i sampling rate
4. Powstały panel zostaje dodany do listy paneli
5. Kroki 2-4 zostają powtórzone jeśli wgrywanych plików jest więcej niż jeden

3.2 Wyświetlanie wykresów cech dźwięku

Żeby przedstawić cechę dźwięku na wykresie w pierwszej kolejności należy wybraną funkcję zastosować do sygnału według ramki. Do tego posłuży nam metoda `apply_function`:

```
    def apply_function(self, func):
        frame_length = self.parent.top_panel.
            get_frame_length()
        frame_step = self.parent.top_panel.get_frame_step()
        results = []
        for start in range(0, len(self.y) - frame_length +
            1, frame_step):
            frame = self.y[start:start + frame_length]
            result = func(frame)
            results.append(np.mean(result))
        return np.array(results)
```

Powyższa metoda:

1. Pobiera wartości długości ramki i odstępu
2. Dla każdej ramki o wskazanej długości i co odstęp wybierany jest wycinek sygnały `y`
3. Dla każdego wycinka stosuje się funkcję. Następnie oblicza się średnią wartość funkcji w tym wycinku, co następnie zostaje przyłączone do listy `results`
4. Metoda zwraca strukturę NumPy array powstałą z listy `results`

Po uzyskaniu tablicy wartości danej cechy sygnału implementowane jest wyświetlanie wykresu:

```
    def display_plot(self, plot_func, title=""):
        fig, ax = plt.subplots(figsize=(8, 2))
        plot_func(ax)
        ax.set_title(title)
        if "Volume" in title:
            ax.set_ylabel("Volume")
        elif "Zero Crossing Rate" in title:
            ax.set_ylabel("Zero Crossing Rate")
        elif "Short-Time Energy" in title:
            ax.set_ylabel("Short Time Energy")
        elif "AMDF" in title:
            ax.set_ylabel("AMDF")
        elif "Autocorrelation" in title:
            ax.set_ylabel("Autocorrelation")
        else:
            ax.set_ylabel("Value")
        ax.set_xlabel("Frame Index")
        buf = io.BytesIO()
        plt.savefig(buf, format='png')
        plt.close(fig)
        buf.seek(0)
        img = Image.open(buf).resize((800, 200), Image.
            Resampling.LANCZOS)
        photo = ImageTk.PhotoImage(img)
        canvas = ctk.CTkCanvas(self.panel, width=800, height
            =200)
        canvas.create_image(0, 0, anchor='nw', image=photo)
        canvas.image = photo
        canvas.pack(pady=5)
```

Powyższa metoda tworzy wykres, nadaje odpowiednie etykiety i przy pomocy biblioteki `io` zapisuje obraz wykresu w pamięci podręcznej by następnie go umieści w *canvie*.

Powyższa metoda wykorzystywana jest w metodzie `update_contents` która w czasie rzeczywistym wprowadza nowe wykresy w zależności od checkboxów zaznaczonych w gónym panelu:

```
    def update_contents(self):
        for widget in self.panel.winfo_children():
            if isinstance(widget, ctk.CTkCanvas):
                widget.destroy()

        y, sr = self.y, self.sr

        if self.parent.top_panel.show_waveform():
            def plot_waveform_with_overlay(ax):
                librosa.display.waveshow(y, sr=sr, ax=ax)
                filters = self.parent.right_panel.
                    get_filters()
```

```
frame_length = self.parent.top_panel.
    get_frame_length()
frame_step = self.parent.top_panel.
    get_frame_step()
volume_vals, zcr_vals, time_stamps = [], [],
     []
for start in range(0, len(y) - frame_length
    + 1, frame_step):
    frame = y[start:start + frame_length]
    volume_vals.append(volume(frame)[0])
    zcr_vals.append(zero_crossing_rate(frame
        )[0])
    time_stamps.append(start / sr)
vol_thresh = np.percentile(volume_vals, 25)
zcr_thresh = np.percentile(zcr_vals, 25)
zcr_high = np.percentile(zcr_vals, 75)
for t, v, z in zip(time_stamps, volume_vals,
     zcr_vals):
    if filters["silence"] and v < vol_thresh
         and z < zcr_thresh:
        ax.axvspan(t, t + frame_step / sr,
            color='blue', alpha=0.3)
    if filters["unvoiced"] and v <
        vol_thresh and z > zcr_high:
        ax.axvspan(t, t + frame_step / sr,
            color='green', alpha=0.3)
    if filters["voiced"] and v > vol_thresh
        and z < zcr_high:
        ax.axvspan(t, t + frame_step / sr,
            color='red', alpha=0.3)
    if filters["music"] and v > vol_thresh
        and z > zcr_high:
        ax.axvspan(t, t + frame_step / sr,
            color='yellow', alpha=0.3)
    if filters["speech"] and v > vol_thresh
        and z < zcr_high:
        ax.axvspan(t, t + frame_step / sr,
            color='purple', alpha=0.3)
legend_patches = []
if filters["silence"]:
    legend_patches.append(mpatches.Patch(
        color='blue', alpha=0.3, label='
        Silence'))
if filters["unvoiced"]:
    legend_patches.append(mpatches.Patch(
        color='green', alpha=0.3, label='
        Unvoiced'))
if filters["voiced"]:
    legend_patches.append(mpatches.Patch(
        color='red', alpha=0.3, label='Voiced
```

```
                ’))
            if filters["music"]:
                legend_patches.append(mpatches.Patch(
                    color=’yellow’, alpha=0.3, label=’
                    Music’))
            if filters["speech"]:
                legend_patches.append(mpatches.Patch(
                    color=’purple’, alpha=0.3, label=’
                    Speech’))

            if legend_patches:
                ax.legend(handles=legend_patches, loc=’
                    upper right’)

        self.display_plot(plot_waveform_with_overlay, "
            Waveform")

    if self.parent.top_panel.show_spectrogram():
        S = librosa.stft(y)
        S_db = librosa.amplitude_to_db(np.abs(S), ref=np
            .max)
        fig, ax = plt.subplots(figsize=(8, 2))
        librosa.display.specshow(S_db, sr=sr, x_axis=’
            time’, y_axis=’hz’, ax=ax)
        ax.set_title("Spectrogram")
        ax.set_xlabel("Time (s)")
        ax.set_ylabel("Frequency (Hz)")

        buf = io.BytesIO()
        plt.savefig(buf, format=’png’, bbox_inches=’
            tight’)
        plt.close(fig)
        buf.seek(0)
        img = Image.open(buf).resize((800, 200), Image.
            Resampling.LANCZOS)
        photo = ImageTk.PhotoImage(img)

        self.spectrogram_image = photo
        self.spectrogram_canvas = ctk.CTkCanvas(self.
            panel, width=800, height=200)
        self.spectrogram_canvas.create_image(0, 0,
            anchor=’nw’, image=photo)
        self.spectrogram_canvas.image = photo
        self.spectrogram_canvas.pack(pady=5)

    for show_flag, func, title in [
        (self.parent.top_panel.show_autocorrelation(),
            autocorrelation, "Autocorrelation"),
        (self.parent.top_panel.show_amdf(), amdf, "AMDF"
            ),
```

```
            (self.parent.top_panel.show_zcr(),
                zero_crossing_rate, "Zero Crossing Rate"),
            (self.parent.top_panel.show_volume(), volume, "
                Volume"),
            (self.parent.top_panel.show_ste(), ste, "Short-
                Time Energy")
        ]:
            if show_flag:
                values = self.apply_function(func)
                self.display_plot(lambda ax: ax.plot(values)
                    , title)
                if title == "Volume":
                    self.display_volume_stats(values)
```

W powyższym kodzie:

1. Usuwane zostają dotychczasowe wykresy w panelu (nie są już potrzebną, bo powstaną nowe)
2. Jeśli użytkownik zaznaczył 'Waveform' to wyświetlany jest wykres przebiegu czasowego z opcjonalnymi wykrytymi fragmentami (więcej informacji o tym w sekcji Rozpoznawanie fragmentów)
3. Jeśli użytkownik zaznaczył 'Spectrogram' generowany jest spektrogram przy pomocy funkcji wbudowanej z biblioteki `Librosa`
4. Dla każdego pozostałego checkboxu: jeśli dana cecha została zaznaczona to generowany jest wykres (zawierający funkcję od sygnału)
5. Jeśli zaznaczono głośność ('Volume') to wyświetlane są również cechy na poziomie klipu

3.3 Odtwarzanie plików dźwiękowych

Odtwarzanie plików dźwiękowych zostało zaimplementowane przy pomocy biblioteki `sounddevice`. Użytkownik na każdym panelu z klasy AudioPanel może wcisnąć przycisk odtwarzający lub przerywający nagranie:

```
play_btn = ctk.CTkButton(btn_frame, text="      ", width=30,
    command=self.play_audio)
        play_btn.pack(side="left", padx=2)
        stop_btn = ctk.CTkButton(btn_frame, text="   ź   ",
            width=30, command=self.pause_audio)
        stop_btn.pack(side="left", padx=2)
```

Powyższe przyciski wykonują metody `play_audio` oraz `stop_audio` zaimplementowane w następujący sposób:

```
    def play_audio(self):
        try:
            if self.is_playing:
                return
            self.is_playing = True
            self.stream = sd.OutputStream(
                samplerate=self.sr,
                channels=1,
                callback=self.audio_callback,
                finished_callback=self.on_stream_finished
            )
            self.stream.start()
        except Exception as e:
            messagebox.showerror("Unable to play", e)

    def audio_callback(self, outdata, frames, time, status):
        if status:
            print(status)
        end = self.playback_pos + frames
        chunk = self.y[self.playback_pos:end]
        if len(chunk) < frames:
            outdata[:len(chunk), 0] = chunk
            outdata[len(chunk):] = 0
            raise sd.CallbackStop()
        else:
            outdata[:, 0] = chunk
        self.playback_pos = end

    def pause_audio(self):
        if self.is_playing and hasattr(self, 'stream'):
            self.stream.stop()
            self.is_playing = False

    def on_stream_finished(self):
        self.is_playing = False
        self.playback_pos = 0
```

Powyższe metody:

- `play_audio` przerywa odtwarzany dźwięk, ustawia wartość zmiennej zerojedynkowej `is_playing` na True i odtwarza nagranie z panelu na którym jest przycisk poprzez utworzenie `OutputStream`
- `audio_callback` służy do dostarczania dźwięków do bufora powstałego w poprzedniej metodzie
- `pause_audio` służy do przerwania streamu i zmiany wartości zmiennej zerojedynkowej `is_playing` na False
- `on_stream_finished` po zakończeniu nagrania zmienia wartość zmiennej

zerojedynkowej `is_playing` na False i przesuwa wskaźnik na początek tak, aby uzytkownik mógł ponownie puścić dany dźwięk od początku

3.4 Usuwanie analizowanych nagrań

Usuwanie analizowanych nagrań z widoku odbywa się poprzez kliknięcie przycisku z koszem na śmieci, który znajduje się przy każdym kafelku na lewym panelu:

```
remove_btn = ctk.CTkButton(frame, text="      ", width=10,
    command=on_remove)
        remove_btn.pack(side="right", padx=2)
```

Wciśnięcie tego przycisku wywołuje ponowne przeładowanie plików audio (`load_audio_files`) na skutek czego usunięte zostają panele z przestrzeni AudioViewer:

```
    def remove(self):
        self.panel.destroy()
        self.parent.left_panel.remove_tile(self.filepath)
        self.parent.audio_panels.remove(self)
```

Usunięty zostaje również kafelek:

```
    def remove_tile(self, filepath):
        if filepath in self.tiles:
            self.tiles[filepath].destroy()
            del self.tiles[filepath]
```

3.5 Eksport do `.csv`

Analogicznie jak przy usuwaniu, przy kafelku na lewym panelu może zostać zaznaczony przycisk zapisywania:

```
export_btn = ctk.CTkButton(frame, text="      ", width=10,
    command=on_export)
        export_btn.pack(side="right", padx=2)
```

Podobnie jak przycisk usuwania, zostaje on wywołany przy przeładowaniu plików dźwiękowych (ma to miejsce po kliknięciu tego przycisku). Zostaje wykonana następująca metoda, służąca do eksportu danych:

```
    def export_data(self):
        save_path = filedialog.asksaveasfilename(
            defaultextension=".csv", filetypes=[("CSV files",
             "*.csv")])
        if not save_path:
            return
        fl = self.parent.top_panel.get_frame_length()
        fs = self.parent.top_panel.get_frame_step()
        with open(save_path, 'w', newline='') as f:
```

```
            writer = csv.writer(f)
            writer.writerow(["Frame", "Start", "Volume", "
                STE", "ZCR", "Autocorr", "AMDF"])
            for i, start in enumerate(range(0, len(self.y) -
                 fl + 1, fs)):
                frame = self.y[start:start + fl]
                writer.writerow([
                    i, start,
                    volume(frame)[0],
                    ste(frame)[0],
                    zero_crossing_rate(frame)[0],
                    np.mean(autocorrelation(frame)),
                    np.mean(amdf(frame))
                ])
```

Implementacja powyższego kodu może zostać opisana następująco:

1. Uzytkownik wybiera ścieżkę, gdzie ma zostać zapisany plik `.csv`
2. Długość i odstęp ramki zostają pobrane z suwaków. Następnie dla każdej funkcji/cechy dźwięku obliczana jest wartość dla ramki (ten sam algorytm jak przy wykresach)
3. Każda kolejna obserwacja wyznaczona w powyżej opisany sposób zostaje wpisana jako wiersz zbioru danych przy pomocy biblioteki `csv`

4 Metody analizy cech dźwięku

W tej sekcji omówiona zostanie implementacja podstawowych cech dźwięku. Tak jak wcześniej zostało to opisane, używana jest metoda `apply_function`, która zastosowuje daną funkcję do sygnału dźwiekowego i wyznacza uśrednioną wartość dla analizowanej ramki, Poniżej zostaną krótko omówione funkcje użyte do wydobywanie cech dźwięku.

4.1 Częstotliwość tonu podstawowego

Częstotliwość tonu podstawowego może być rozumiana jako podstawowa częstotliwość sygnału dźwiękowego o strukturze harmonicznej. Są dwa popularne sposoby uzyskania takiej częstotliwości:

- **Autokorelacja**, za pomocą której wyszukiwany jest pewien wzór podobieństwa, tzn jak podobny jest dany sygnał do sygnału przesuniętego o pewien odstęp
- **AMDF** natomiast mierzy średnią bezwzględną różnicę między sygnałem o sygnałem oddalonym o dany krok

Obie te metody zostały zaimplementowane w programie:

4.1.1 Autokorelacja

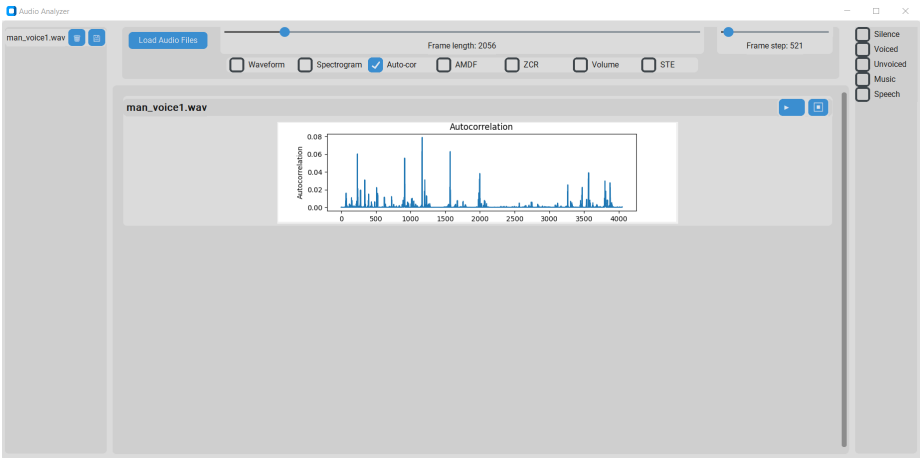
Autokorelację definiujemy wzorem:

$$R_n(l) = \sum_{i=0}^{N-l-1} s_n(i) \cdot s_n(i+l)$$

i analogicznie zostaje zaimplementowana w programie:

```
def autocorrelation(s):
    N = len(s)
    return np.array([np.sum(s[i:] * s[:N-i]) for i in range(
        N)])
```

Wykres autokorelacji dla próbki męskiego głosu powstały w programie prezentuje się następująco:



Rysunek 3: Wygląd interfejsu aplikacji po zastosowaniu autokorelacji na nagraniu dźwiękowym

4.1.2 AMDF

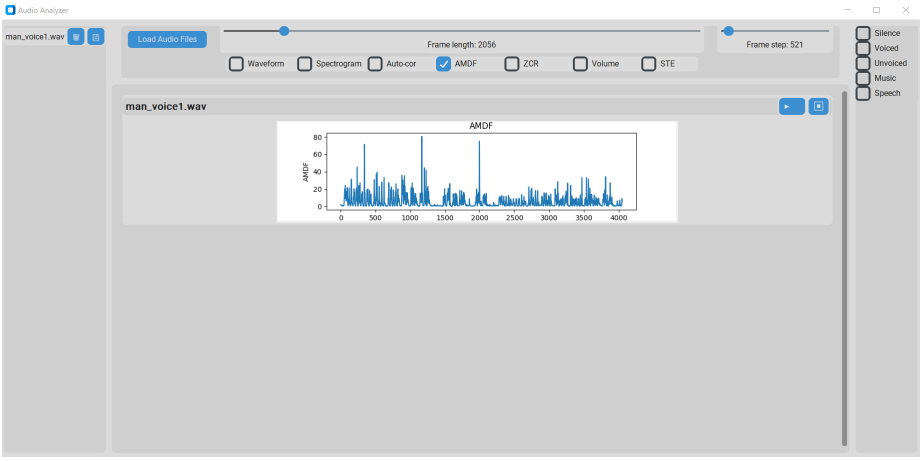
AMDF (Average Magnitude Difference Function) definiujemy jako:

$$A_n(l) = \sum_{i=0}^{N-l-1} |s_n(i) - s_n(i+l)|$$

AMDF w aplikacji zostało zaimplementowane przy pomocy funkcji:

```
def amdf(s):
    N = len(s)
    return np.array([np.sum(np.abs(s[i:] - s[:N - i])) for i
        in range(N)])
```

Wywołanie wykresu z powyższą funkcją prezentuje się następująco (nie jest zaskakujące, że ogólny kształt obu metod określania częstotliwości tonu podstawowego daje jest podobny i peaki osiągane są w tych samych miejscach)



Rysunek 4: Wygląd interfejsu aplikacji po zastosowaniu AMDF na nagraniu dźwiękowym

4.2 Cechy dźwięku na poziomie ramki

W podobny sposób jak algorytmy znajdowania częstotliwości tonu podstawowego, pewne podstawowe cechy dźwięku mogą być uzyskane z ramek za pomocą określonych funkcji:

4.2.1 Zero Crossing Rate

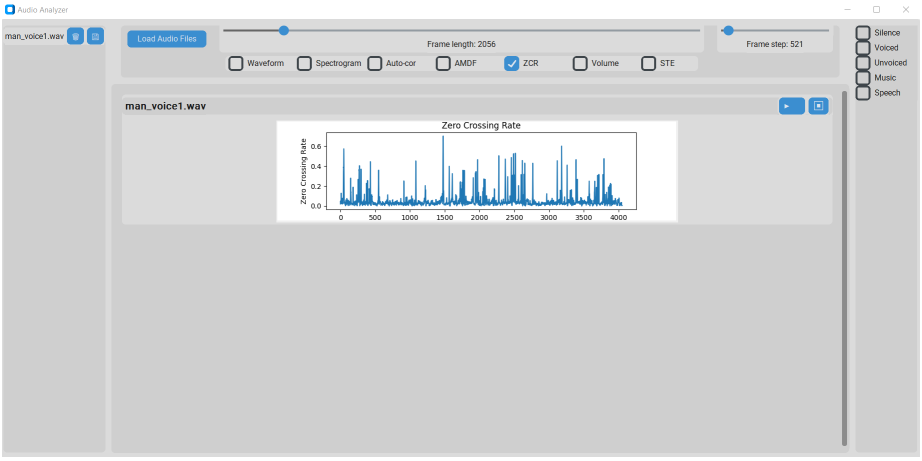
Zero Crossing Rate to miara tego jak często sygnał przekracza zero i jest definiowana za pomocą:

$$\mathrm{ZCR}(n) = \frac{1}{2N} \sum_{i=1}^{N-1} |\mathrm{sign}(s_n(i)) - \mathrm{sign}(s_n(i-1))|$$

ZCR zostało zaimplementowane w aplikacji:

```
def zero_crossing_rate(s):
    return np.array([((s[:-1] * s[1:]) < 0).sum() / len(s)])
```

Wywołanie ZCR na nagraniu męskiego głosu wygląda następująco:



Rysunek 5: Wygląd interfejsu aplikacji po zastosowaniu ZCR na nagraniu dźwiękowym

4.2.2 Głośność

Głośnośc rozumiana jest jaka miara intensywności sygnału dźwiękowego. Definiowana jest w następujący sposób:

$$\text{Volume}(n) = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1} s_n(i)^2}$$

Głośność została zaimplementowana w programie:

```
def volume(s):
    return np.array([np.sqrt(np.mean(s**2))])
```

Wykres głośności próbki nagrania męskiego głosu prezentuje się następująco:

4.2.3 Short Time Energy

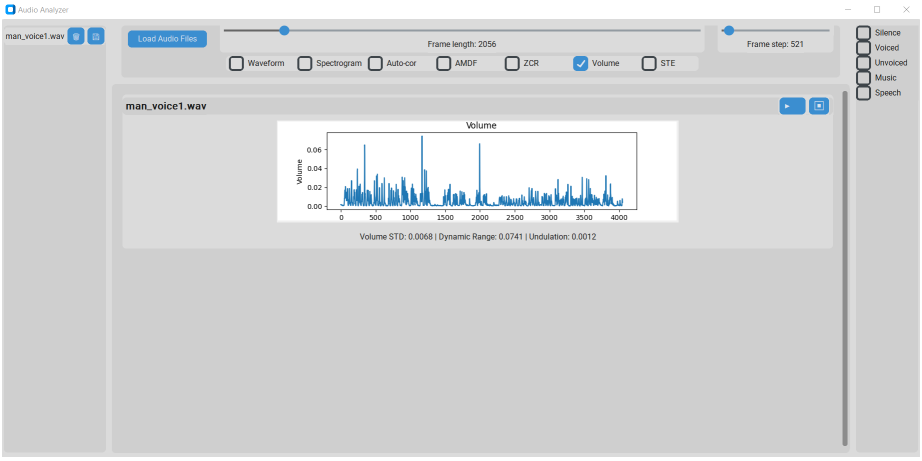
Short Time Energy jest miarą energii w krótkim czasie. Matematyczna definicja STE to:

$$\text{STE}(n) = \frac{1}{N}\sum_{i=0}^{N-1} s_n(i)^2$$

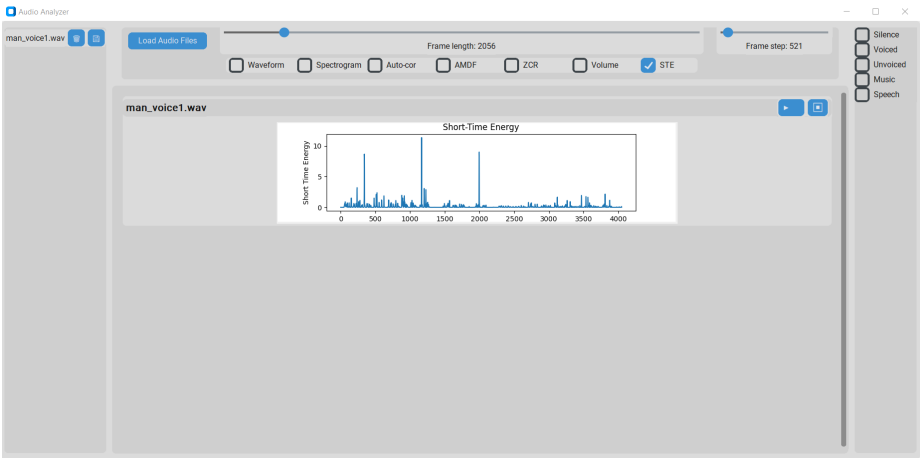
Implementacja STE w programie:

```
def ste(s):
    return np.array([np.sum(s**2)])
```

Wywołanie STE na wykresie:



Rysunek 6: Wygląd interfejsu aplikacji po zastosowaniu funkcji głośności na nagraniu dźwiękowym



Rysunek 7: Wygląd interfejsu aplikacji po zastosowaniu funkcji STE na nagraniu dźwiękowym

4.3 Cechy głośności na poziomie klipu

Tak jak widoczne było to na wykresie głośności, pod wykresem pojawiły się statystyki dotyczące głośności na poziomie klipu. Konkretnie:

- **vts** – odchylenie standardowe głośności w klipie
- **vdr** – różnica między największą a najmniejszą wartością głośności (rozstęp)

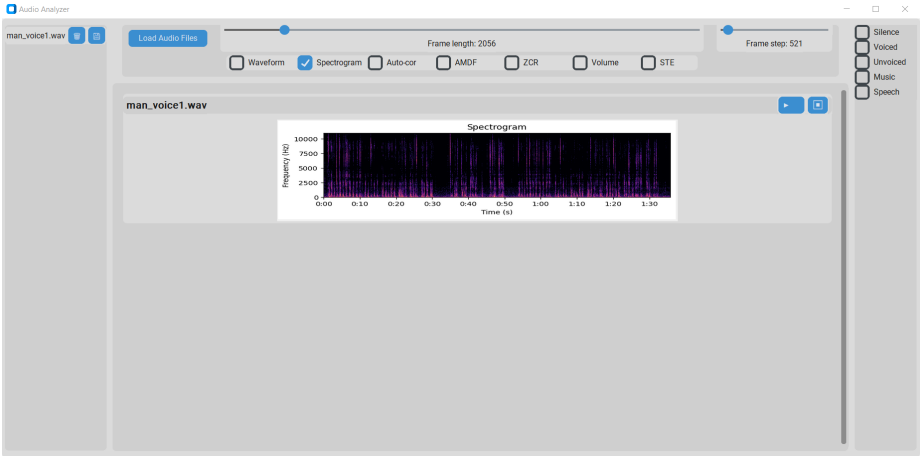
- **vund** – falistość głośności, czyli średnia bezwzględna róznica między kolejnymi wartościami głośności

Obliczone i wyświetlone zostały one za pomocą implementacji:

```
    def display_volume_stats(self, values):
        vstd = np.std(values)
        vdr = np.max(values) - np.min(values)
        vund = np.mean(np.abs(np.diff(values)))
        stats = f"Volume STD: {vstd:.4f} | Dynamic Range: {
            vdr:.4f} | Undulation: {vund:.4f}"
        ctk.CTkLabel(self.panel, text=stats).pack(pady=5)
```

4.4 Spektrogram

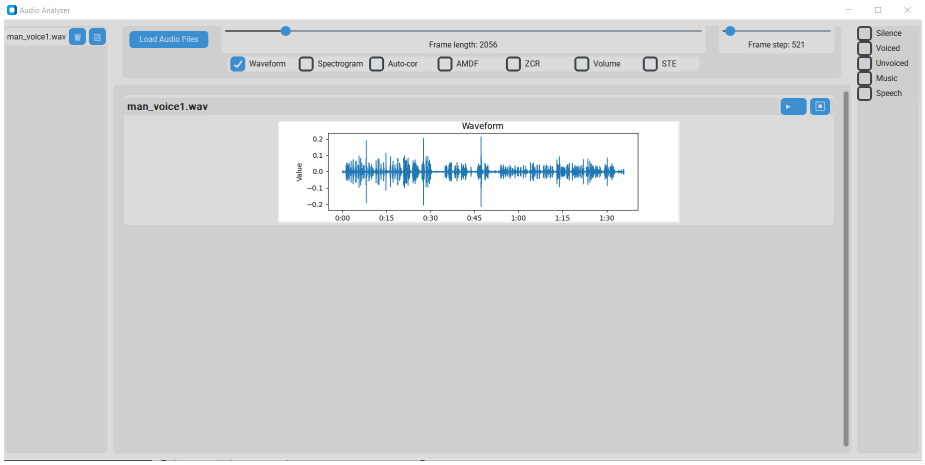
Program po wybraniu odpowiedniego checkboxu w panelu górnym przgotowuje spektrogram dla pliku dźwiękowego. W tym celu wykorzystywane są **wbudowane funkcje pakietu Librosa**. Implementacja została przedstawiona w sekcji Wyświetlanie wykresów cech dźwięku.



Rysunek 8: Wygląd interfejsu aplikacji po zastosowaniu spektrogramu nagrania dźwiękowym

4.5 Wykres przebiegu czasowego

Podobnie jak spektrogram, wykres przebiegu czasowego zostaje wytworzony przy pomocy funkcji wbudowanej z biblioteki `Librosa` przy pomocy `y` i `sr`. Dla testowanego wcześniej nagrania przebieg prezentuje się następująco:

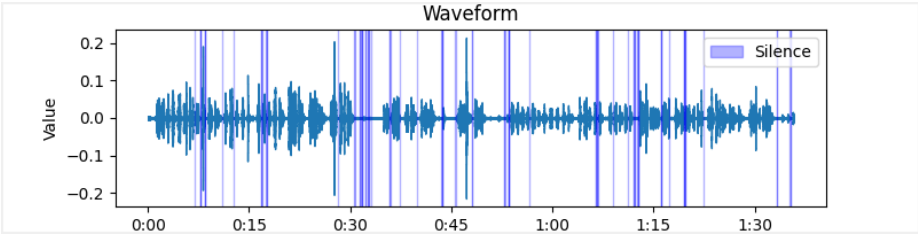


Rysunek 9: Wygląd interfejsu aplikacji po zastosowaniu wykresu przebiegu czasowego nagrania dźwiękowym

4.5.1 Rozpoznawanie fragmentów

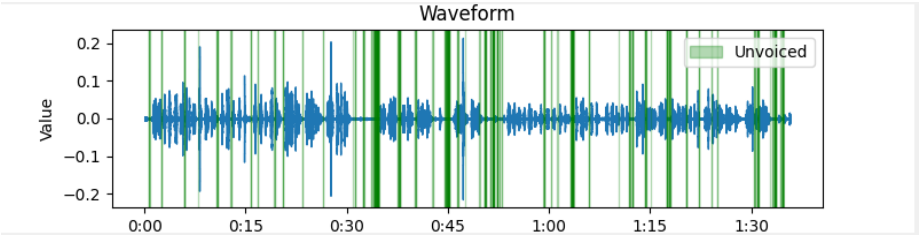
Na wyżej pokazany wykres mogą zostać naniesione kolorowe obszary, które symbolizują fragmenty. Są one wyszukiwane według następujących kryteriów heurystycznych (**ze względu na bardzo długi czas oczekiwania na wykonanie się operacji autokorelacji i AMDF zdecydowałem się jako prototyp zbudować proste zasady w oparciu o głośność i ZCR**):

$$\text{Volume}(n) < \theta_v \wedge \text{ZCR}(n) < \theta_{\text{zcr}} \Rightarrow \text{Silence}$$



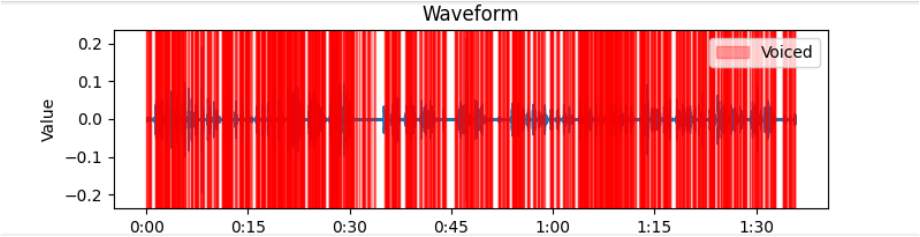
Rysunek 10: Wykres przebiegu czasowego nagrania dźwiękowego z rozpoznaną ciszą

$$\text{Volume}(n) < \theta_v \wedge \text{ZCR}(n) > \theta_{\text{zcr\_high}} \Rightarrow \text{Unvoiced}$$



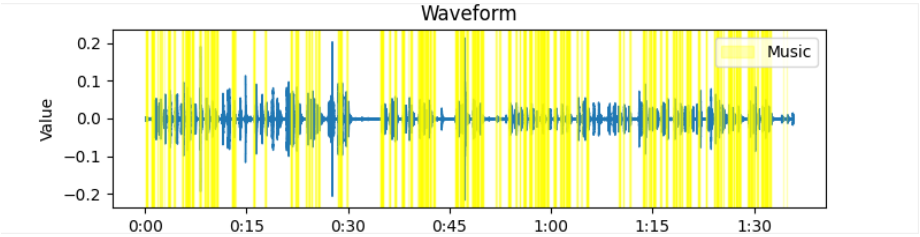
Rysunek 11: Wykres przebiegu czasowego nagrania dźwiękowego z rozpoznanymi fragmentami bezdźwięcznymi

$$\text{Volume}(n) > \theta_v \wedge \text{ZCR}(n) < \theta_{\text{zcr\_high}} \Rightarrow \text{Voiced}$$



Rysunek 12: Wykres przebiegu czasowego nagrania dźwiękowego z rozpoznanymi fragmentami dźwięcznymi

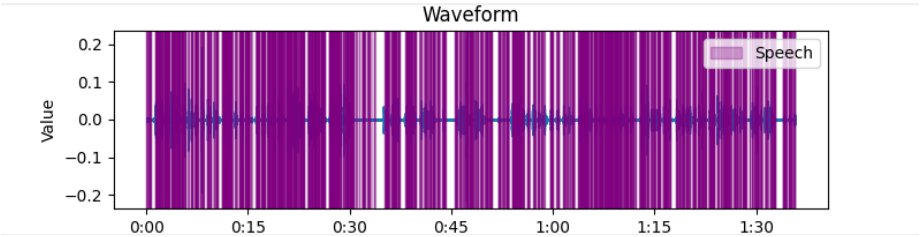
$$\text{Volume}(n) > \theta_v \wedge \text{ZCR}(n) > \theta_{\text{zcr\_high}} \Rightarrow \text{Music}$$



Rysunek 13: Wykres przebiegu czasowego nagrania dźwiękowego z rozpoznanymi fragmentami z muzyką

$$\text{Volume}(n) > \theta_v \wedge \text{ZCR}(n) < \theta_{\text{zcr\_high}} \Rightarrow \text{Speech}$$

gdzie progi zostały zdefiniowane kolejno jako 25 i 75.



Rysunek 14: Wykres przebiegu czasowego nagrania dźwiękowego z rozpoznanymi fragmentami z mową

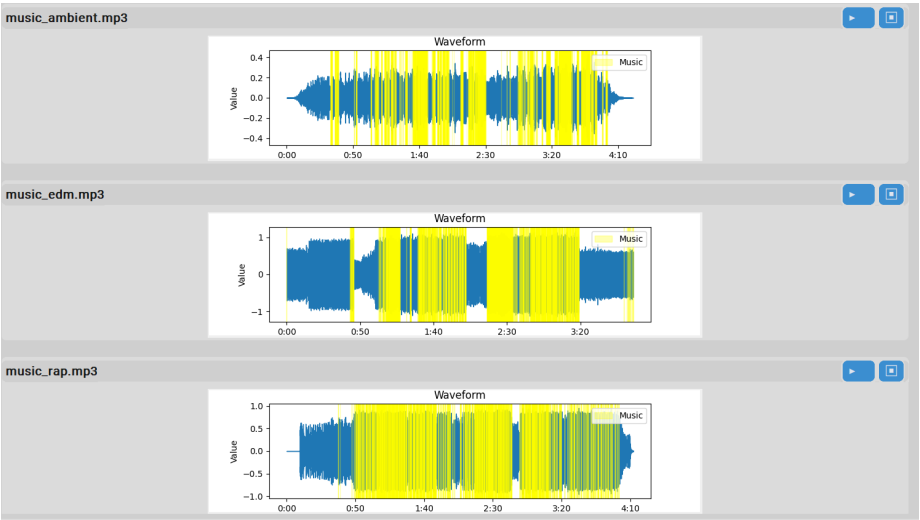
5 Przykłady zastosowania

5.1 Analiza różnych gatunków muzycznych

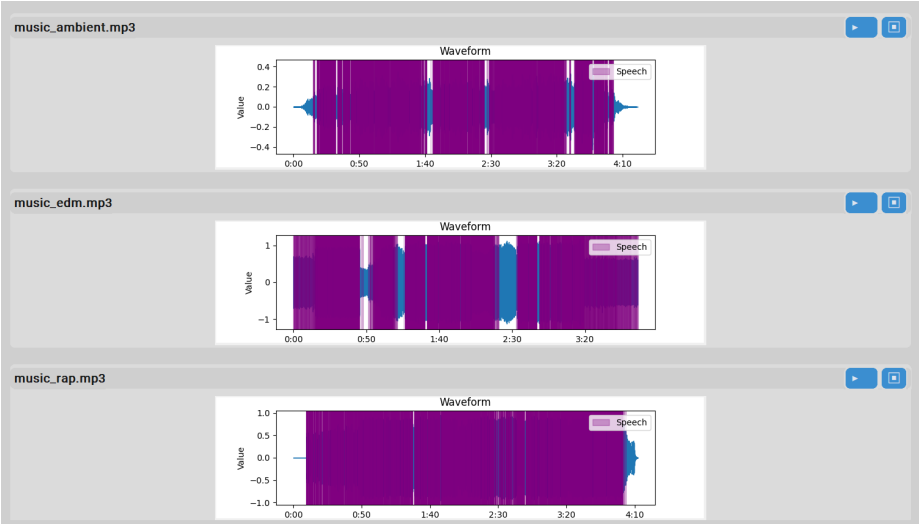
Wybrane zostały trzy piosenki z różnych gatunków muzycznych:

- **Rap** GrubSon – Na szczycie
- **Ambient** Brian Eno – An Ending
- **EDM** DVBBS & Borgeous – Tsunami

Przetestowane je pod kątem rozpoznania fragmentów z muzyką i z mową:



Rysunek 15: Wykres przebiegu czasowego piosenek różnych gatunków muzycznych z rozpoznanymi fragmentami z muzyką

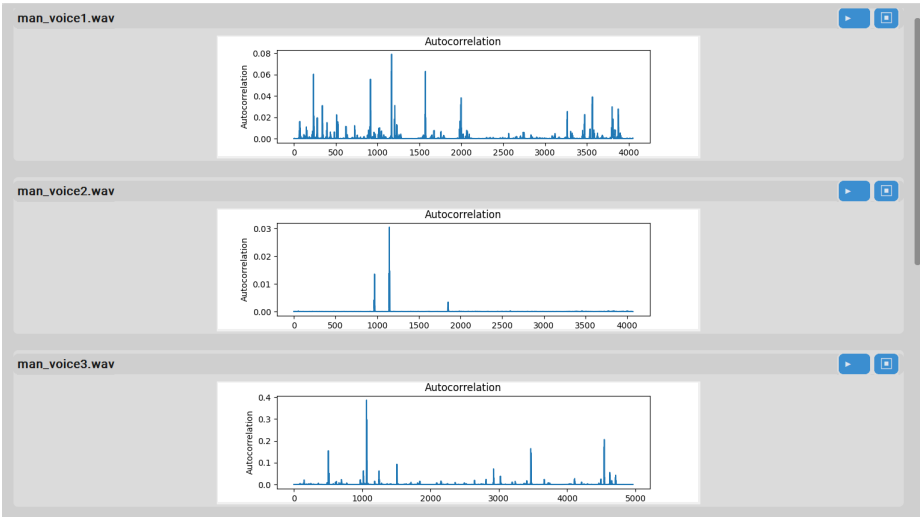


Rysunek 16: Wykres przebiegu czasowego piosenek różnych gatunków muzycznych z rozpoznanymi fragmentami z mową

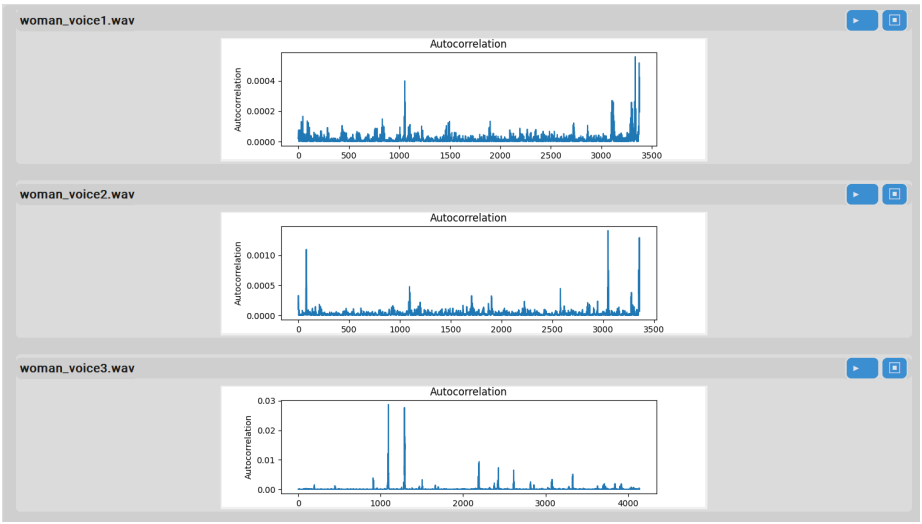
Jak widać fragmenty z mową najchętniej są przypisywane piosence rapowej (co nie jest zaskakujące, bowiem jako jedyna zawiera głos wokalisty). W piosence EDM zrozumiały jest brak rozpoznanej muzyki na początku (wolny wstęp, tzw. "buil-up"). Podobnie w piosence ambient niski stopień przypisania do muzyki wskazuje na ciche fragmenty, heurystycznie nieuznawane za muzykę.

5.2 Głos męski vs. żeński

W celu zauważenia ciekawych wzorów dla głosu męskiego i żeńskiego próbka 3 nagrań żeńskiego i 3 męskiego głosu zostały porównane pod względem częstotliwości tonu podstawowego metodą autokorelacji:



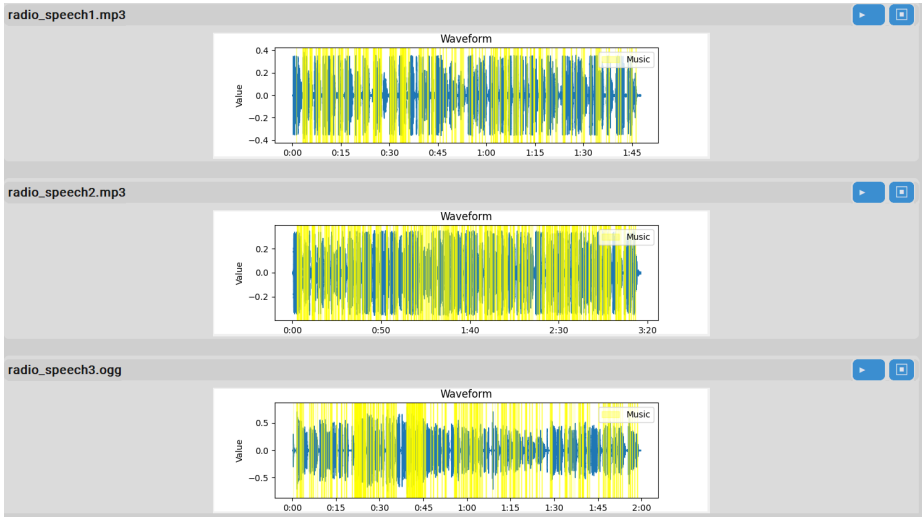
Rysunek 17: Wykresy autokorelacji dla nagrań z męskim głosem



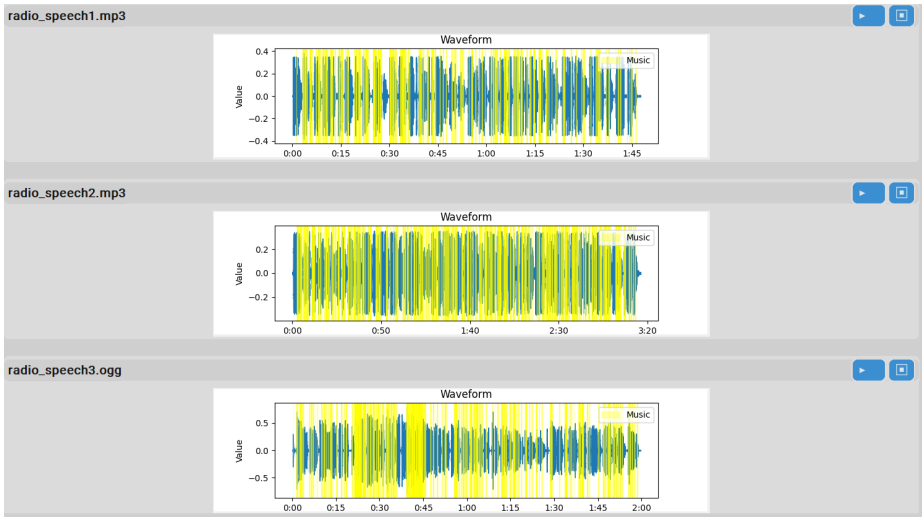
Rysunek 18: Wykresy autokorelacji dla nagrań z damskim głosem

5.3 Muzyka vs. mowa

W celu porównania mowy z muzyką próbka trzech nagrań mowy pochodzących z radia (źródło 1, źródło 2) została przeanalizowana pod kątem rozpoznawania mowy i muzyki: z powyższego wynika, że mowa jest również często wykrywana jak przy piosenkach, ale muzyka istotnie rzadziej – te fragmenty zawierają liczne



Rysunek 19: Wykres przebiegu czasowego nagrań mowy z radia z rozpoznanymi fragmentami z muzyką



Rysunek 20: Wykres przebiegu czasowego nagrań mowy z radia z rozpoznanymi fragmentami z mową

przerwy, co wskazuje na potencjalny punkt zaczepienia do sporządzenia reguł rozpoznawania pliku dźwiękowego jako muzykę lub mowę.

6 Wnioski

Jednym z zauważalnych wniosków podczas testowania różnych funkcji programu jest długi czas oczekiwania na wykonanie algorytmów znajdowania częstotliwości tonu podstawowego, w szczególności dla długich nagrań.

Ze względu na wysoką złożonośc obliczeniową powyższych operacji, niemożliwe było utworzenie kryeteriów rozpoznawania momentów w oparciu o te metody (zostawia to pole do potencjalnego rozszerzenia). Innym potencjalnym sposobem rozbudowania projektu byłoby manipulowanie progami według których rozpoznawane są fragmenty.