AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA ELEKTRONIKI

PRACA DYPLOMOWA INŻYNIERSKA

Biblioteka do detekcji mowy w czasie rzeczywistym dla rozwiązań wbudowanych

Voice activity detection library for embedded real-time applications

Autor: *Sebastian Zarębski*
Kierunek studiów: *Elektronika i Telekomunikacja*
Opiekun pracy: *dr inż. Jakub Gałka*

Kraków, 2016

Spis treści

1. Wstęp

1.1. Cel i zakres pracy

Rozwój telekomunikacji i technologii operujących na mowie ludzkiej przyczynił się do jednoczesnego wzrostu zapotrzebowania na rozwiązania je wspierające, w tym mechanizmy przeznaczone do wykrywania aktywności mowy ludzkiej (ang. *Voice Activity Detection*, VAD). Ich zastosowanie wpłynęło, między innymi, na zwiększenie efektywności algorytmów rozpoznawania mowy [1] oraz zmniejszyło zapotrzebowanie energetyczne telefonów działających w sieciach komórkowych GSM (kodek AMR używa modułu VAD w celu kontroli działania transmisji przerywanej DTX, ograniczając tym samym wypromieniowany sygnał z telefonu w chwili, kiedy rozpozna chwilowy brak aktywności któregoś z rozmówców) [2].

Celem niniejszej pracy jest zaprojektowanie i implementacja stworzonej w języku C++ biblioteki nazwanej dalej **vadlibrary** i realizującej wykrywanie aktywności mowy ludzkiej. Całość operuje na plikach audio w formacie Wave.

Wbudowaną właściwością biblioteki jest jej prostota w implementacji (z perspektywy wykorzystującej jej użytkownika). Większość jej wewnętrznych mechanizmów jest dla użytkownika niedostępna, przewidziano jednak możliwość wpływu na generowany wynik analizy plików poprzez wymuszenie innych niż domyślne parametrów wejściowych wewnętrznych funkcji (szerokość histerezy stanów oraz próg detekcji jej obecności w strumieniu audio).

1.2. Struktura pracy

Niniejsza praca stanowi opis procesu działania biblioteki opisanej w rozdziale poprzednim. Na całość składa się łącznie sześć rozdziałów.

Rozdział pierwszy zawiera ogólną informację na temat zawartości i budowy czytanej pracy.

Rozdział drugi zawiera wiedzę teoretyczną stojącą za zagadnieniem wykrywania aktywności mowy ludzkiej w strumieniach audio. Opisane w nim informacje składają się na pełen obraz implementowanego przeze mnie algorytmu. Podrozdział 2.2 opisuje przebieg podejmowania ostatecznej decyzji odnośnie wykrycia głosu ludzkiego wewnątrz badanego pliku.

Rozdział trzeci jest dokładnym opisem budowy, interfejsu biblioteki i procedury jej działania. Opisane zostały tutaj parametry konfiguracji, którymi sterować może użytkownik (podrozdział 3.1). Następnie, w podrozdziale 3.2, znaleźć można dokładne informacje o samym algorytmie, na którym opiera się biblioteka i procedurach w niej realizowanych. Mowa tu o wczytywaniu plików, logice raportowania związanych z tym błędów oraz o ważniejszych metodach służących do analizy pobranych danych.

W kolejnym rozdziale przedstawiono sposób implementacji i wykorzystania biblioteki do pracy. Zawarte tu informacje informują o poprawnej metodzie korzystania z parametrów biblioteki.

Przedostani rozdział jest podsumowaniem prac, które wykonałem realizując zadany temat. Wnioski i osobiste spostrzeżenia poprzedzone są prezentacją otrzymanych wyników.

Ostatnie rozdział składa się na bibliografię użytą wewnątrz dokumentu.

2. Zagadnienie wykrywania aktywności mowy ludzkiej

Istotnym problemem w rozpoznawania mowy ludzkiej jest wcześniejsze określenie jej obecności w badanym sygnale. Rozdział mowy od muzyki, szumu otoczenia, czy ciszy ma swoje zastosowania w konkretnej dziedzinie przetwarzania mowy, którą jest wykrywanie aktywności mowy ludzkiej. Spełnienie tych celów wymagało wybrania (jako parametrów) wielu cech, zarówno z dziedziny czasu jak i częstotliwości [3]. Najczęściej spotykane i wykorzystane przeze mnie w pracy to 4 Hz-owa modulacja energii (wyznaczona na podstawie parametru MED), entropia modulacji oraz gęstość przejść przez zero.

Uzyskane dzięki tej bibliotece dane mogą posłużyć zarówno do wykrywania takiej aktywności jak i fragmentów ciszy.

2.1. Składowe algorytmu wykrywania aktywności mowy

Gęstość przejść przez zero (ang. *Zero-crossing ratio***,** dalej ZCR) jest jednym z podstawowych sposobów wykorzystywanych do parametryzacji oraz wykrywania mowy ludzkiej [10]. Działanie to polega na wyznaczeniu w badanym sygnale akustycznym średniej chwilowej liczby przejść przez zero dla określonej ramki tego sygnału [4] [5] na podstawie wzoru (1).

$$ZCR = \frac{1}{2}\sum_{i=0}^{M} |sign(i) - sign(i+1)| \tag{1}$$

Gdzie *sign()* to funkcja signum oznaczająca znak, zaś *x(i)* jest dyskretną wartością sygnału dźwiękowego. M jest liczbą badanych próbek.

W pracy [5] autorzy określili minimalny próg na 25 przejść z dodaną sumą wartości średniej przejść w trakcie ciszy. Na podstawie testów, które przeprowadziłem, stwierdzam, że wartość ta jest dla mojego rozwiązania zbyt wysoka i funkcjonalność tej metody opieram o empiryczne poznany parametr. Wartość progowa ZCR określona jest więc na podstawie przeprowadzonych obliczeń

dla kilku pierwszych ramek sygnału, które uznawane są powszechnie za szum tła.

Na podstawie propozycji złożonej przez pracowników Akademii Górniczo-Hutniczej: mgr inż. Stanisław Kacprzak oraz prof. dr hab. inż. Mariusza Ziółko [3], w algorytm włączony zostaje opracowany przez nich parametr **MED** (ang. *Minimum Energy Density*). Charakteryzuje się on bardzo dobrą skutecznością w determinacji obecności mowy w sygnale i jednoczesną prostotą obliczeń. Mowa, jako wyjątkowo charakterystyczny sygnał akustyczny, posiada specyficznie wysoki poziom **energii modulacji w częstotliwości 4Hz**. Oznacza to, że w trwającym jedną sekundę oknie wystąpią cztery minima energii. Zgodnie z tym założeniem wymagane jest okno klasyfikacji o długości co najmniej 250 milisekund. Parametr MED definiowany jest wówczas (dla k-tego okna klasyfikacyjnego) jako:

$$MED(k) = min\{probE(n) : (k-1)\cdot M + 1 \leqslant n \leqslant k \cdot M\}, \tag{2}$$

gdzie M określa ilość ramek wewnątrz tego okna. Zaś poprzez *probE(n)* rozumiemy znormalizowaną krótko-czasową energię pojedynczej ramki i wyznaczamy ją równaniem:

$$probE(n) = \frac{E(n)}{\sum_{k=1}^{N} E(k)} \tag{3}$$

Minimalnej wartości *E(n)* (definiującej rozdzielczość klasyfikacji) szukamy w oknie klasyfikacji, którego długość powinna być krótsza od okna normalizacji. W trakcie fazy treningu znajdowana jest wartość progowa (ang. *treshold*). Wówczas, jeśli parametr MED znajdzie się poniżej tego progu, to dana ramka zostaje oznaczona jako zawierająca mowę. Decyzja ta jest określona poprzez równanie (4).

$$class(k) = \begin{cases} speech & if \ \exists n : E(n) < \lambda, \ where \ (k-1)\cdot M + 1 \leqslant n \leqslant k \cdot M \\ music & otherwise \end{cases} \tag{4}$$

gdzie:

$$\lambda = threshold \cdot \sum_{n=1}^{N} E(n) \tag{5}$$

Pamiętać należy jednak, że pomimo wysokiej skuteczności w rozpoznawaniu treści (na poziomie 96.4%), autorzy przypominają o fakcie, że testy przeprowadzone zostały tylko na nagraniach zawierających mowę lub muzykę. Badanie pominięto dla ścieżek zawierających jednocześnie mowę na tle muzyki, bowiem ocena taka staje się już subiektywna. Dodatkowo oryginalna i cytowana praca zawiera zastosowanie do podziału mowa-muzyka, biblioteka prezentowana tutaj pomija jednak jego dwoistość skupiając się jedynie na wykrywaniu aktywności głosu ludzkiego.

Trzecim wykorzystanym parametrem jest **entropia modulacji** badanego sygnału. Na podstawie twierdzenia Shannona określić możemy entropię jako średnią ilość informacji przypadającą na pojedynczy symbol i opisać wzorem (6).

$$H(S) = -\sum_{i=1}^{N} P(s(i)) \cdot \log_2(P(s(i))) \tag{6}$$

Gdzie *S = [s(1), ... , s(n)]* jest wektorem źródłowym o długości N symboli (opisując tym samym liczbę wszystkich zdarzeń w danej przestrzeni) zaś *P(s(i))* opisuje prawdopodobieństwo pojawienia się symbolu *i*. Entropia przyjmuje wartość maksymalną, kiedy pojawienie się wszystkich symboli jest równoprawdopodobne.

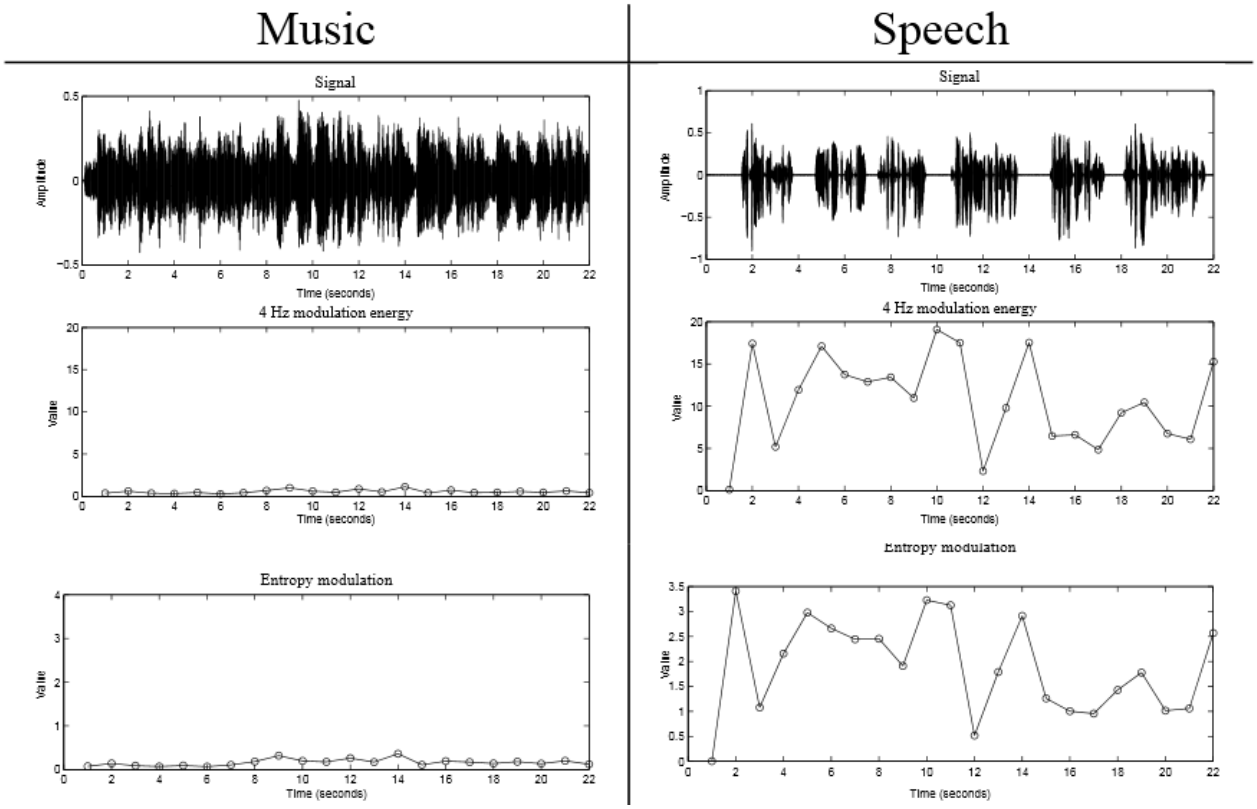
Zastosowanie pomiaru entropii w problemie detekcji mowy ludzkiej opiera się na założeniu, że spektrum sygnału jest bardziej zorganizowane (co wskazuje na niższą wartość entropii) podczas fragmentów zawierających mowę ludzką niż szum [8]. Pomiar entropii, w badanym zagadnieniu, określa więc wzór (7):

$$H(|Y(w,t)|^2) = -\sum_{w=1}^{\Omega} P(H(|Y(w,t)|^2)) \cdot \log_2(P(H(|Y(w,t)|^2))) \tag{7}$$

Gdzie $P(H(|Y(w,t)|^2)) = \dfrac{|Y(w,t)|^2}{\sum_{w=1}^{\Omega} |Y(w,t)|^2}$ (8) opisuje prawdopodobieństwo

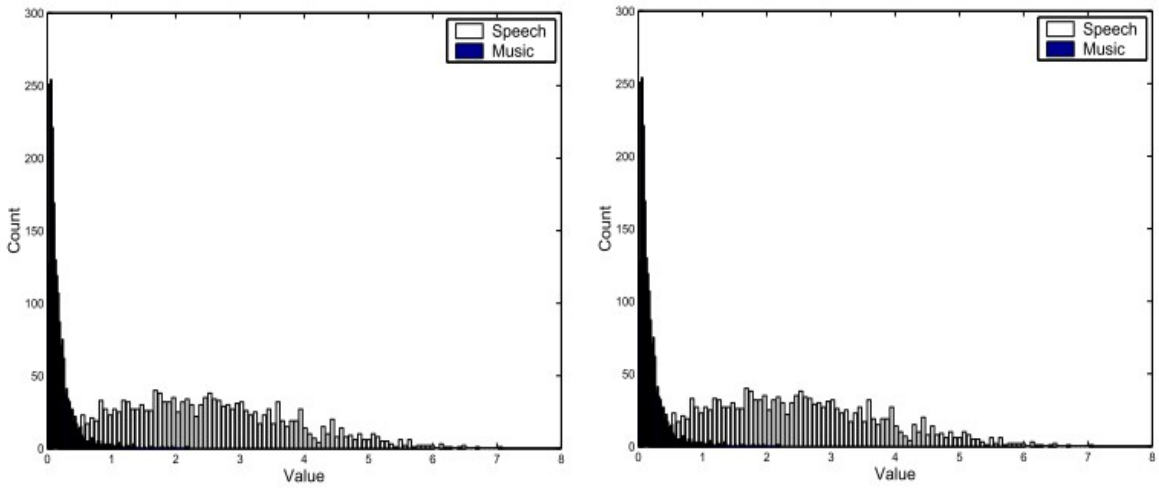
pasma częstotliwości *w* dla spektrum mocy w ramce czasowej *t*.

Muzyka wydaje się być zauważalnie bardziej „uporządkowana” niż mowa i to biorąc pod uwagę zarówno analizę sygnału jak i spektrogramów (wykres 2.1). Przewidywana wartość entropii mowy ludzkiej jest więc wyższa niż muzyki [7].



Rys. 2.1: Porównanie parametrów energii modulacji 4Hz i entropii modulacji dla mowy i muzyki.

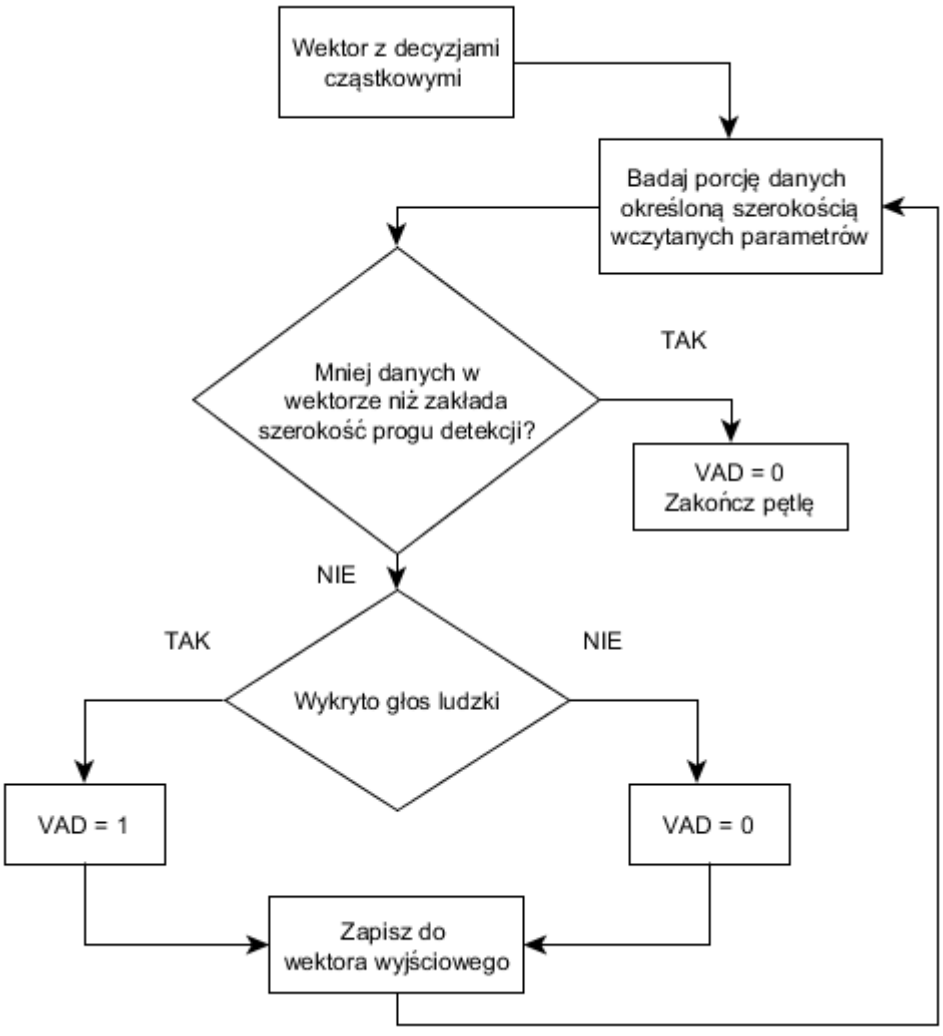
Warto zauważyć symetrię zachowania obu badanych współczynników.



Rys. 2.2: Rozkład 4Hz-owej modulacji energii oraz entropii modulacji.

2.2. Decyzyjność algorytmu

Decyzja o wykryciu mowy ludzkiej w wybranej ramce podejmowana jest na podstawie zaokrąglonej średniej ważonej mniejszych, wcześniejszych decyzji. Sprawdzanie trwa tak długo, dopóki wektor nie zostanie zbadany w całości lub pozostała część danych okaże się mniejsza niż brany pod uwagę parametr szerokości progu detekcji mowy (np. dla domyślnych 100 ms pozostała suma decyzji cząstkowych odpowiada szerokości 60ms). W tym drugim wypadku ostatnia wartość wektora wyjściowego zawsze wyniesie 0. Sposób podejmowania decyzji zobrazowany jest schematem 2.3:



Rys. 2.3: Schemat podejmowania ostatecznej decyzji o aktywności głosu

Decyzja podejmowana jest zgodnie ze wzorem:

$$VadSubDecision = \sum_{n=0}^{N} \left\lceil \frac{w_1 \cdot ZCR[n] + w_2 \cdot ENTROPY[n] + w_3 \cdot MED[n]}{w_1 + w_2 + w_3} \right\rceil , \quad (9)$$

gdzie $w1$ – średnia ważona dla *ZCR*, równa **0.2**;
$w2$ – średnia ważona dla *Entropii,* równa **0.35**
$w3$ – średnia ważona dla *MED*, równa **0.45**;

Powyższe progi zostały wybrane przeze mnie na podstawie przeczytanych i załączonych prac naukowych. Parametr gęstości przejść przez zero jest wyjątkowo narażony na błędy w środowisku z wysokim stosunkiem mocy do szumu, dlatego otrzymał najniższą wagę. Najwyższa waga została przypisana energii MED – na podstawie bardzo wysokiego współczynnika trafień uzyskanego przez twórców [3].

Zatem: $$VAD = \begin{cases} 1 & jeśli\ VadSubDecision > 0.5 \\ 0 & jeśli\ inaczej \end{cases} \quad (10)$$

Na potrzebę zobrazowania systemu załóżmy więc istnienie poniższych wektorów. Opisują one przykładowe decyzje cząstkowe obliczone dla badanego sygnału (jedna pozycja odpowiada jednej ramce sygnału o długości 10 milisekund):

ZCR = [1 0 1 0 0 0 1 1 0 1],

MED = [1 0 1 1 1 0 1 1 1 1],

ENTROPY = [1 0 0 0 0 1 1 0 1 0],

zatem początkowa analiza danych wyjściowym przyniesie rezultat w postaci:

$$VAD = [\lceil \frac{1}{1} \rceil, \lceil \frac{0}{1} \rceil, \lceil \frac{0.55}{1} \rceil, \lceil \frac{0.45}{1} \rceil, \lceil \frac{0.45}{1} \rceil, \lceil \frac{0.35}{1} \rceil, \lceil \frac{1}{1} \rceil, \lceil \frac{0.65}{1} \rceil, \lceil \frac{0.8}{1} \rceil, \lceil \frac{0.65}{1} \rceil]$$

$$VAD = [1,0,1,0,0,0,1,1,1,1]$$

Posiadając taki wynik, w końcowym kroku sprawdzamy długość ciągu danych i na tej podstawie wyciągamy ostateczne wnioski. Zgodnie z domyślnymi wartościami (szerokość progu detekcji i zakres histerezy poprzednich ramek) badane są kolejne ciągi wewnątrz wektora. Próg wykrycia aktywności głosu wynoszący 100ms wymaga więc sumarycznej wartości 10 ramek ocenionej zgodnie ze wzorem (10). Jeśli algorytm wykryje ciąg (lub kolejne ciągi) z decyzją wyjściową braku aktywności o (zsumowanej) długości równej progowi nieaktywności mowy (domyślnie 200 ms), wtedy taka wartość zostaje zwrócona.

Powyższy przykład, po ponownej analizie, zwróci więc informację, że w badanym okresie równym 100 ms mowa została wykryta.

3. Interfejs i procedura działania stworzonej biblioteki

Założeniem zawartym w temacie pracy jest wymóg działania biblioteki w czasie rzeczywistym. Dlatego wczytywanie danych z pliku oraz ich analiza następuje etapami i kolejno, ramka po ramce, zapisywaniu do wektora wyjściowego decyzji o wykryciu aktywności mowy ludzkiej w tym fragmencie. Procedura trwa do chwili zakończenia pliku lub pojawienia się jednego z krytycznych błędów.

3.1. Parametry konfiguracji

Pomimo, że biblioteka charakteryzuje się swoją hermetycznością i zamknięciem, dla użytkownika udostępniono dwa parametry, którymi wypływać może na wyjściowy wektor z danymi na temat decyzji algorytmu o obecności mowy:

- **szerokość histerezy aktywności i nieaktywności mowy –** wartości te określają wpływ poprzedniego fragmentu sygnału na kolejną decyzję algorytmu. Im wyższa wartość tego parametru, tym algorytm pobierze więcej danych z decyzją z wstecz. Wówczas decyzja zostaje podjęta na podstawie szerszego zakresu danych – pobrane wartości z decyzjami zostają porównane z gotowym wektorem o szerokości wyznaczonej przez próg decyzyjny (standardowo 100ms). Jeśli zaokrąglona suma arytmetyczna decyzji poprzednich i teraźniejszej wyniesie mniej niż 0.5, wówczas (nawet pomimo posiadania wartości równej 1) wektor ten zostaje przepisany na wartość 0.

  Domyślna wartość: 0 ms.

  Dopuszczalne wartości: 0ms, 10ms, 20ms, 50ms, 100ms.

  (Uwaga, w przypadku podania złej wartości algorytm zaokrągla ją do najbliższej dopuszczalnej)

- **próg detekcji mowy** – określa szerokość ramki czasowej, na podstawie której algorytm określa pojawienie się lub zanik mowy ludzkiej w sygnale. Parametr określa ile ramek 10 milisekundowych zostaje zsumowanych jako decyzja ostateczna.

  Domyślna wartość: 100 ms dla aktywności, 200 ms dla nieaktywności (uwaga, wartość nieaktywności musi być większa równa wartości aktywności)

  Dopuszczalne wartości: 100, 200, 300, 400, 500 ms.

3.2. Procedura i algorytm działania biblioteki

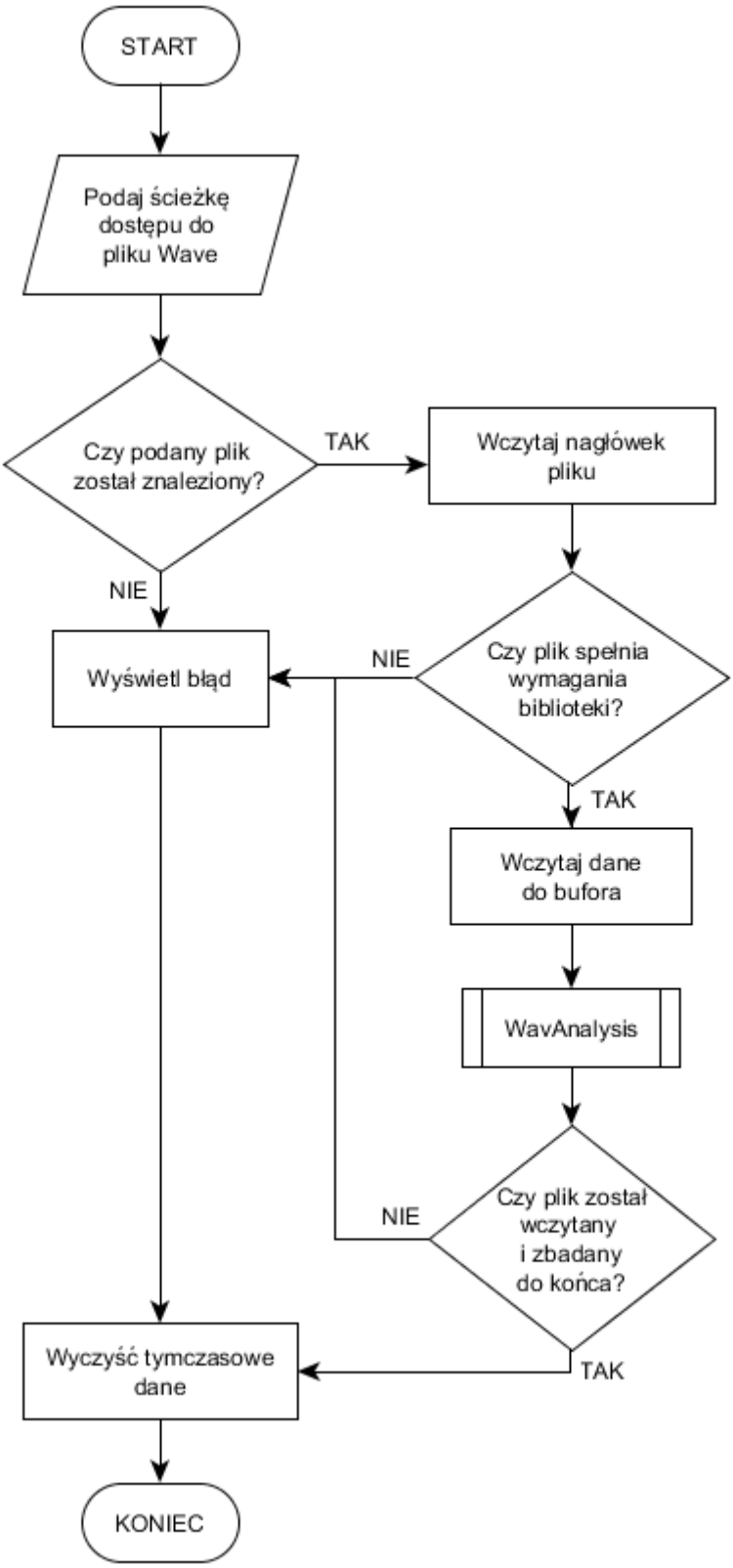
Biblioteka zaprojektowana została według założeń, które spełnić musi poddawany jej analizie plik Wave. Są to kolejno:

- **16-bitowa** postać danych,
- dźwięk jest **jednokanałowy**,
- dopuszczalne częstotliwości próbkowania to **8 kHz, 16 kHz lub 44,1 kHz.**

Sygnał dzielony jest na ramki o długości **10 milisekund**. Wybór takiego okna czasowego podyktowany został możliwością dokonania obliczeń arytmetycznych na liczbach całkowitych lub zmiennoprzecinkowych z pojedynczą precyzją (*float*), a zatem do dokonania tych obliczeń szybciej [9].

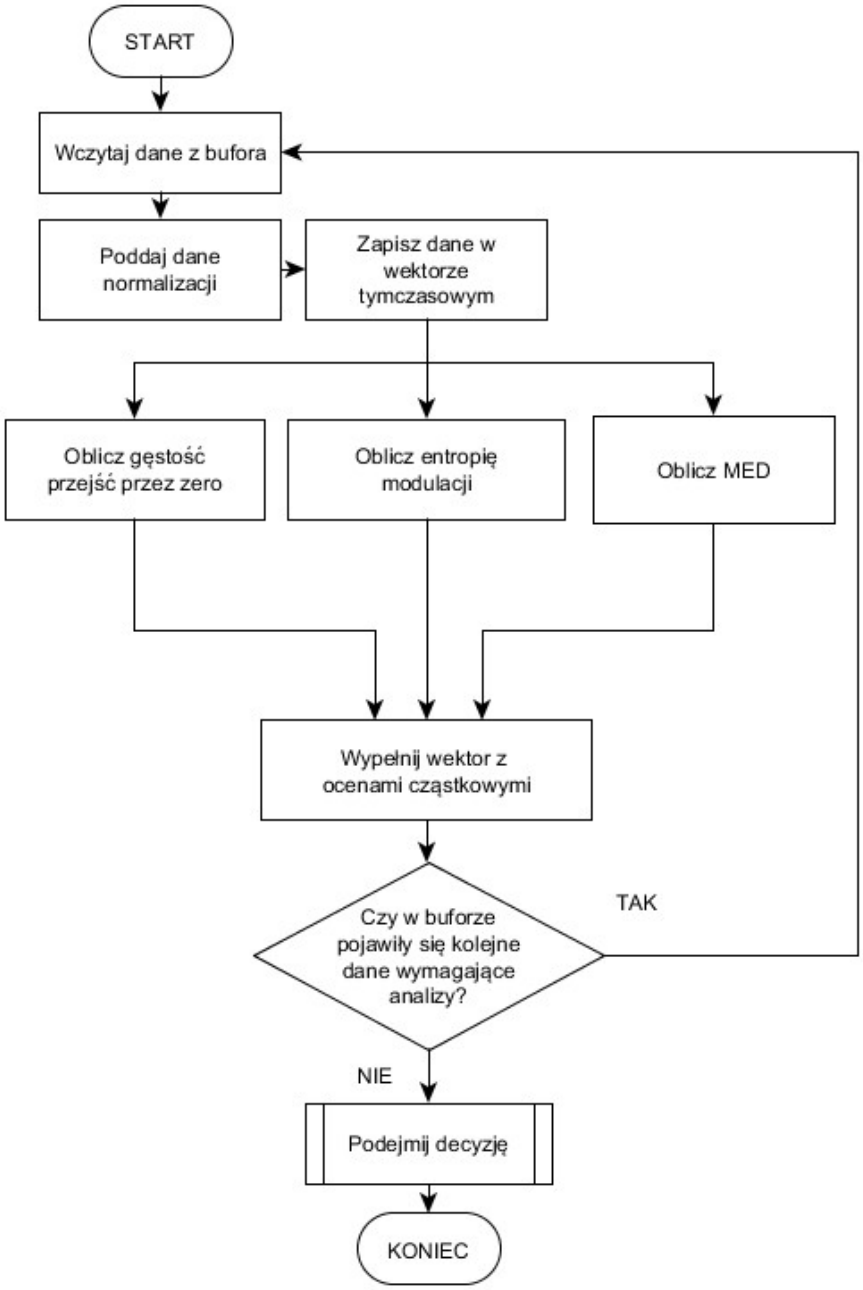
Algorytm rozpoczyna pracę żądając od użytkownika podania ścieżki do pliku Wave. W przypadku odnalezienia tego pliku i pozytywnej weryfikacji jego zawartości, dane zostają wczytywane do bufora. Jego wielkość określona jest przez pobraną z nagłówka pliku informację o częstotliwości próbkowania f_p sygnału. Są to kolejno wartości 80 próbek dla f_p = 8kHz, 160 próbek dla f_p = 16kHz oraz 441 próbek dla f_p = 44,1kHz**.** Pętla iteracyjna działa do chwili zakończenia pliku lub pojawienia się błędu uniemożliwiającego jego pełne wczytanie. W takim wypadku użytkownik zostaje uprzedzony o tym fakcie stosownym ostrzeżeniem. Koniec pracy z plikiem Wave prowadzi do wyczyszczeniu używanej pamięci podręcznej.

Schemat działania tych operacji odpowiada wbudowanej klasie o nazwie **WavRead** i zaprezentowany został w rysunku 3.1.



Rys. 3.1: Proces wczytywania i obsługi pliku Wave

Podrutyna **WavAnalysis** odnosi się do klasy biblioteki o tej samej nazwie. Odpowiada ona za całościową analizę wczytanych danych i zobrazowana została na schemacie 3.2.



Rys. 3.2: Proces analizy kolejnych próbek pobranych z pliku Wave

Załadowane dane poddawane są normalizacji, dzięki której obliczenia zostają przeprowadzone na mniejszych liczbach. Po tej operacji przekazywane są do

tymczasowego wektora. Analiza składa się z jednoczesnego obliczenia trzech parametrów. Pozyskiwane są kolejno wartości opisujące gęstość przejść przez zero fragmentu sygnału, entropię modulacji oraz MED (opisującego 4Hz-ową energię modulacji). Następnie każdy z tych wyników zapisany zostaje do przypisanego mu wektora.

W chwili zakończenia wczytywania danych, kiedy bufor jest pusty, algorytm przechodzi do ostatniego kroku, który opisany jest na wykresie jako subrutyna „Podejmij decyzję”. Przebieg jej działania obrazowany został w rys. 2.3.

Analiza odbywa się zgodnie z zasadą, że zsumowana tymczasowo ilość 10 milisekundowych ramek odpowiada i zgadza się z wartością wczytanego parametru „activity threshold”. Parametr posiada domyślną (i minimalną dopuszczalną) wartość 100 milisekund. Sprawdzane są rozpatrywane długości okien aktywności i braku aktywności mowy, zapisane w parametrze **próg detekcji mowy.** Podejmowana jest wówczas decyzja ostateczna, bazująca na sumie trzech ocen cząstkowych. Jeśli jednak danych jest mniej niż zakłada parametr, co najczęściej oznacza koniec pliku, pętla sprawdzająca zostaje zakończona, a do wektora z decyzją wpisana ostatnia wartość, z informacją o braku aktywności mowy w tym okresie.

3.3. Obsługa plików Wave

Dane dźwiękowe w plikach Wave zapisane są jako strumień audio reprezentowany w formie PCM. Taki 16 bitowy strumień ze względu na obecność znaku (dane są typu signed) wymusza użycie szerszego typu wczytywanych danych. Zatem każda próbka wczytywana jest jako dwa bity [6].

Nagłówek pliku wczytywany jest i systematyzowany na podstawie struktury **Wave_HEADER.** Tabela X.X zawiera najważniejsze pola tej struktury z opisem ich wpływu na pracę algorytmu.

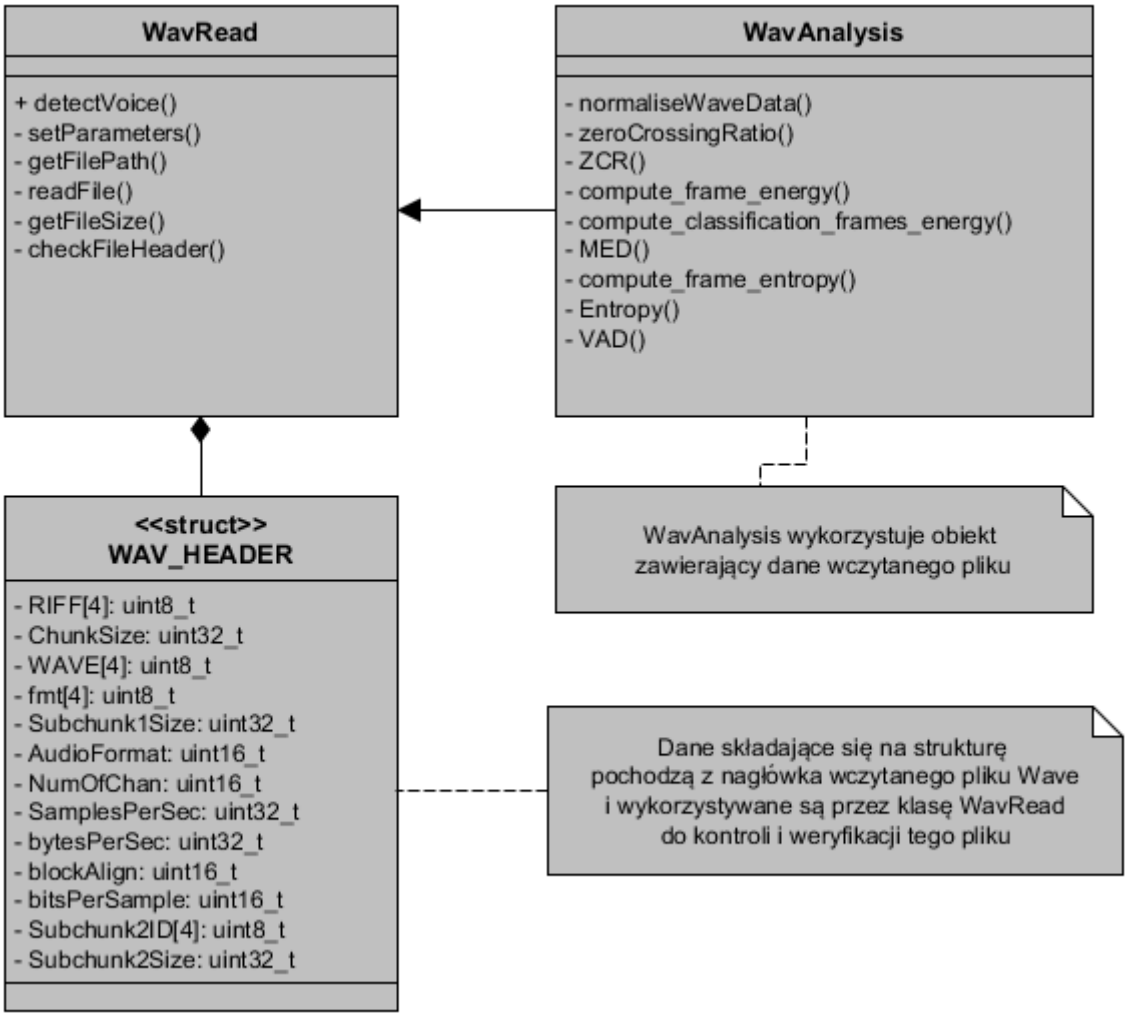
Rozmiar	Nazwa pola	Opis pola
uint16_t	AudioFormat	format audio pliku, tutaj zawsze 1, co równa się PCM (kwantyzacja linearna), wartości inne niż 1 informują o kompresji danych
uint16_t	NumOfChan	liczba kanałów, tutaj zawsze 1, czyli Mono
uint32_t	SamplesPerSec	częstotliwość próbkowania wyrażona w Hz
uint32_t	bytesPerSec	ilość bajtów na sekundę
uint16_t	blockAlign	zawsze równe 2, co oznacza plik mono 16-bitowy
uint16_t	bitsPerSample	liczba bitów na próbkę
uint32_t	Subchunk2Size	Łączna ilość próbkowanych danych

Tab. 1: Struktura nagłówka pliku Wave użyta w bibliotece.

Pozostałe pola nagłówka oraz dokładniejszy ich opis zawarte są w [6]. Zastosowane typy, czyli uint8_t, uint16_t, uint32_t, określają dane bez obecności znaku, o zawsze stałej szerokości kolejno 8, 16 i 32 bajtów.

W trakcie analizy pliku Wave sprawdzane są wartości w przedstawionej tabeli wraz z założeniami biblioteki. Ich niezgodność spowoduje zatrzymanie pracy algorytmu i wyświetlenie odpowiedniej informacji.

3.4. Diagram klas oraz opis ważniejszych metod biblioteki



Rys. 3.1: Diagram klas biblioteki

Przedstawiony na rys. 3.1 diagram klas prezentuje dokładną budowę biblioteki i zależności panujące pomiędzy jej składowymi. Klasa **WavRead** do działania korzysta ze struktury danych o nazwie **WAV_HEADER**, która wypełniana jest danymi pobieranymi z nagłówka badanego pliku Wave. Te informacje są następnie wykorzystywane do kontroli dalszego procesu analizy. Klasa **WavAnalysis** korzysta z obiektu klasy **WavRead** zawierającego zapisywane w buforze dane. Korzysta również jednej z jej metod, posługując się nią w razie konieczności wyświetlenia krytycznego błędu działania biblioteki.

3.5. Metoda publiczna

Metoda realizująca	WavRead::detectVoice()
Parametry wejściowe	std::string path, **opcjonalne:** int hysteresis, int activity_threshold, int non_activity_threshold
Zwraca	std::vector<char> VAD

Tab. 2: Opis metody detectVoice().

Metoda główna. Pobiera ścieżkę do pliku i na jej podstawie wczytuje plik, następnie poddając go analizie. W przypadku pojawienia się w trakcie pracy błędu zwrócony wektor przyjmie pojedynczą wartość równą 0.

Pozyskany na ten sposób wektor wyjściowy można przekazać dalej, przykładowo zapisując go do pliku tekstowego:

```
std::vector<char> VAD = wav.detectVoice(„filepath”);
std::ofstream output_file("X:\\path\\outcome.txt");
for (int i = 0; i < VAD.size(); i++){
     output_file << VAD[i] << ", ";
 }
```

Należy wówczas pamiętać o dołączeniu dyrektywą include <fstream>.

3.6. Wyświetlane błędy

- **E1 - Parametr „nazwa_parametru” jest źle skonfigurowany** – błąd tej treści pojawia się w sytuacji, kiedy użytkownik definiując parametry sterujące biblioteką poda ich błędne wartości. Dokładny opis parametrów i ich dopuszczalnych wartości znajduje się w rozdziale 3.1.
- **E2 - Nie znaleziono podanego pliku** – podana ścieżka do pliku jest niepoprawna. Została wpisana z błędem lub nie wskazuje na żaden plik.
- **E3 - Zły typ pliku** - pomimo, że plik pod podaną ścieżką został znaleziony, to jest on nieprawidłowego typu. Biblioteka obsługuje tylko pliki z rozszerzeniem .wav.
- **E4 - Dane są skompresowane** – pole nagłówka AudioFormat posiada wartość inną niż 1, wskazuje to na użytą kompresję danych. Wartość równa 1 jest wymagana i określa, że dane PCM poddane zostały kwantyzacja linearnej, przez co można działać na nich bezpośrednio. Biblioteka nie obsługuje plików i innej wartości.
- **E5 - Niepoprawna liczba kanałów** – wczytany plik jest dwukanałowy. Biblioteka pracuje na plikach z dźwiękiem jednokanałowym.
- **E6 - Zła częstotliwość próbkowania danych** – częstotliwość próbkowania danych w pliku nie zgadza się z założeniami. Dopuszczalne wartości to 8000, 16000, 44100 Hz.
- **E7 - Plik jest 8-bitowy** – biblioteka obsługuje tylko pliki 16-bitowe.
- **E8 - Wystąpił błąd podczas analizy pliku** – coś poszło nie tak podczas wczytywania danych i nie zostały one pobrane w całości. Wczytana ilość różni się od informacji o ilości danych zapisanych w nagłówku pliku. Dotychczasowe decyzje algorytmu zostają usunięte z pamięci.

4. Praktyczny przykład wykorzystania biblioteki

Wykorzystanie standardowe (z domyślnymi parametrami):

Po załączeniu biblioteki do kodu, w celu przeprowadzenia analizy VAD pliku, wystarczającym jest użycie poniższych instrukcji:

```
WavRead wav;
wav.detectVoice(„X:\\filepath\\file.wav”);
```

Spowoduje to utworzenie obiektu *wav* klasy *WavRead.* W ten sposób uzyskujemy dostęp do metody *detectVoice*, za której parametr podajemy ścieżkę do badanego pliku Wave (istnieje konieczność użycia podwójnego ukośnika – pojedynczy może zostać rozpatrzony przez kompilator jako biały znak).

Użycie biblioteki z wykorzystaniem parametrów wejściowych:

W celu użycia udostępnionych parametrów konieczne jest użycie wartości opisanych w rozdziale 3.1. W przypadku użycia innych zostaną one zignorowane przez bibliotekę (lub zaokrąglone do najbliższych właściwych). Podanie złych wartości wygeneruje natychmiastowy błąd z kodem **E1**, którego opis znajduje się w rozdziale 3.6.

Metoda detectVoice() przyjmuje następujące parametry (w ustalonej kolejności):

- hysteresis – szerokość histerezy aktywności i nieaktywności mowy,
- activity_treshold – próg detekcji aktywności mowy,
- non_activity_treshold – próg określający fragment niektywności mowy.

Zatem ostateczna forma użycia biblioteki przybierze postać:

```
WavRead wav;
wav.detectVoice(„X:\\filepath\\wav.wav”, 10, 200, 300);
```

4.1. Wpływ parametrów wejściowych na pracę programu

Zmiana szerokości histerezy:

Załóżmy istnienie obliczonych już danych wyjściowych z decyzją o wykryciu (bądź nie) mowy ludzkiej w jednym z fragmentów sygnału. Interesujący nas fragment zaznaczony został niebieskim kolorem. Użytkownik definiuje szerokość histerezy z krokiem jednej ramki, czyli 10 milisekund. Przypuśćmy, że zdecydował się na 50ms (ramki zaznaczone na czerwono).

Rozpatrywany przez nas sygnał zostaje więc podzielony zgodnie z poniższym:

1	0	1	1	1	0	1	1	0	0	1	0	1	1	1	0

Standardowo decyzja podejmowana jest biorąc pod uwagę jedynie niebieską część wektora z danymi. Zgodnie z powyższym, to decyzja wynosi:

$$VAD_{100\text{ms}} = \left\lceil \frac{0+1+1+0+0+1+0+1+0+0}{10} \right\rceil = \left\lceil \frac{4}{10} \right\rceil = 0 \quad .$$

Jednakże, w tym wypadku algorytm bierze pod uwagę opisywany parametr, więc równanie przyjmie postać:

$$VAD_{100\text{ms}} = \left\lceil \frac{0+1+1+0+0+1+0+1+0+0}{10} + \frac{4}{5} \right\rceil = \left\lceil \frac{12}{10} \right\rceil = 1$$

Zmiana progu detekcji aktywności oraz nieaktywności mowy:

Dzięki tym parametrom wpływać możemy na szerokości okien decyzyjnych. Załóżmy przykładowy fragment wektora wyjściowego:

0	1	1	0	0	1	0	1	1	1	1	0	1	1	1	0	1	0	0	1	1	1

Kolorem niebieskim zaznaczony jest fragment odpowiadający domyślnej długości opisywanych okien. Użytkownik postanawia zmienić domyślny 100 milisekundowy próg decyzyjny na dłuższyy, rzędu 200 milisekund. Powyższa ramka

zostanie zatem rozpatrywana zgodnie z nową długością. Więc ostatecznie zwróconą wartością będzie:

$$VAD_{200\text{ms}} = \left\lceil \frac{0+1+1+0+0+1+0+1+1+1+1+0+1+1+1+0+1+0+0+1}{20} \right\rceil$$

$$VAD_{200\text{ms}} = \left\lceil \frac{12}{20} \right\rceil = 1$$

Zmiana parametru dla **nieaktywności** działa w sposób analogiczny. Należy jednak pamiętać, że nie nie bierze on udziału w obliczeniach. Służy jedynie za warunek wpisania zera do wektora wyjściowego. Zgodnie z tym, jeśli wartość tego parametru zostanie zmieniona na 300ms, wówczas dopiero **po upływie** tego czasu wartość ostateczna wektora wyjściowego wyniesie 0.

Uwaga – parametr ten wpływa na długość wektora wyjściowego. Zmiana jego wartości (przykładowo) ze 100ms na 200ms skróci go dwukrotnie.

5. Podsumowanie, wnioski i spostrzeżenia

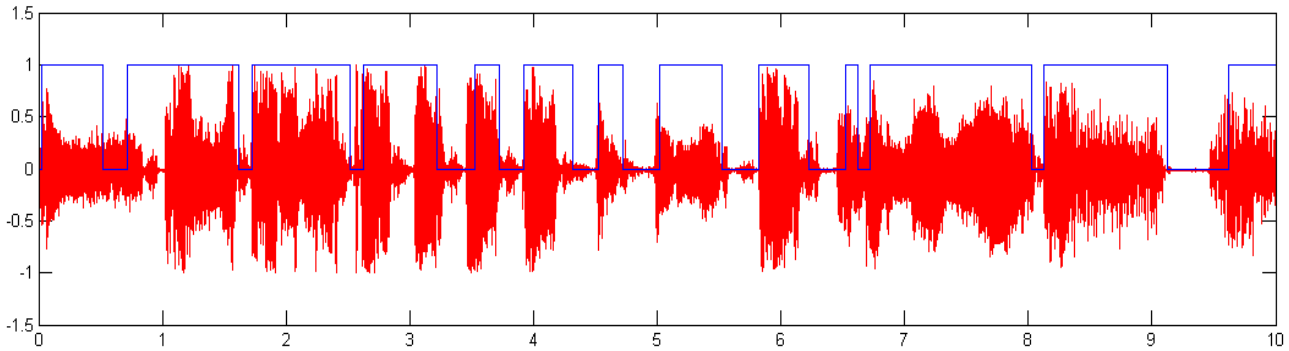
Działanie biblioteki zostało sprawdzone na 3 dość znacznie różniących się od siebie plikach. Pierwszy z plików z nich 10 sekund i charakteryzował się dużym poziomem mocy sygnału i zauważalnym szumem słyszalnym w tle. Drugi plik Wave był plikiem kilkuminutowym o bardzo niskim poziomie mocy. Trzeci plik, również zawierający sygnał o niewielkiej mocy i nieco krótszy, to w porównaniu do pierwszego, posiadał jeszcze dodatkowe tło, które wydawało się być szumem zebranym przez mikrofon mówiącego.

Poniżej prezentowane wykresy otrzymałem poprzez wyeksportowanie wektora wyjściowego do pliku tekstowego, które następnie wczytałem w środowisku Matlab (łącznie z badanym plikiem) i tam odpowiednio skalując zakres danych uzyskałem wizualizację mojej pracy.

Kolor czerwony – strumień audio.

Kolor niebieski – otrzymany wektor VAD.

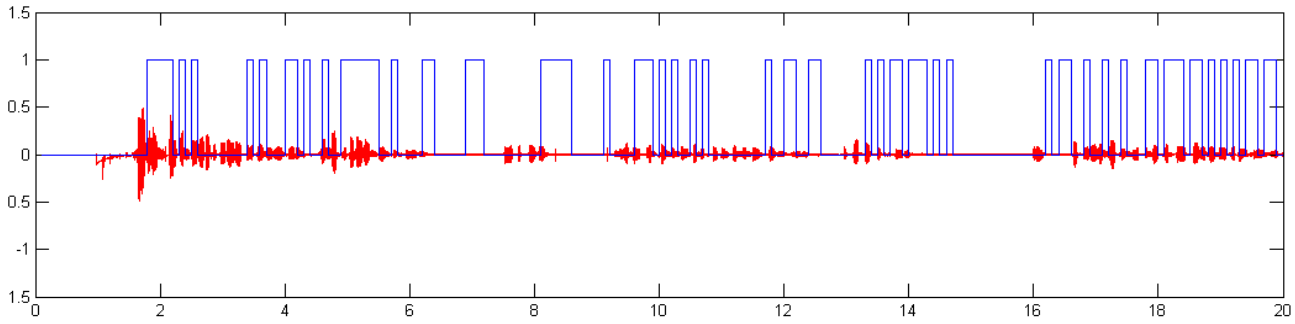
Pierwszy sygnał testowy– wysoka moc sygnału.



Rys. 5.1: Pierwszy sygnał testowy, charakteryzujący się wysokimi poziomami mocy i szumu

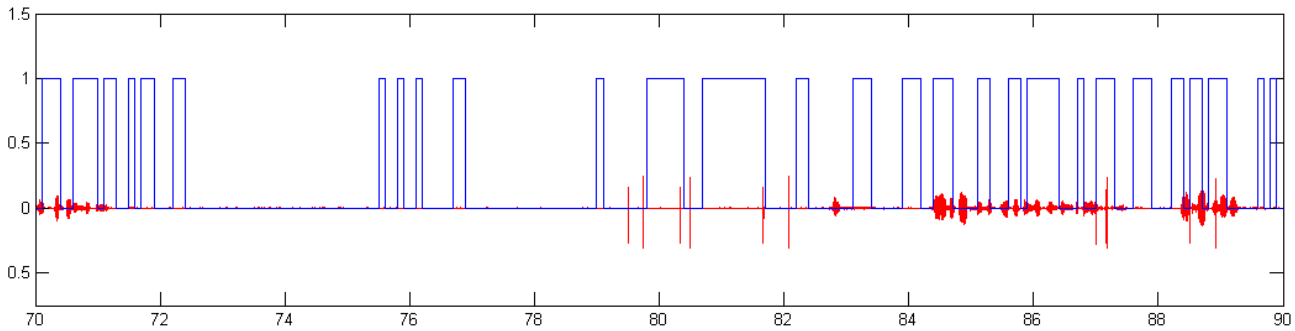
W wypadku tym zauważalna jest dokładność zastosowanego algorytmu. Zbadany sygnał jest wyraźnie podzielony na okresy mowy i ciszy. Parametry wejściowe nie muszą być zmienione.

Drugi sygnał testowy – niska moc, niski szum.



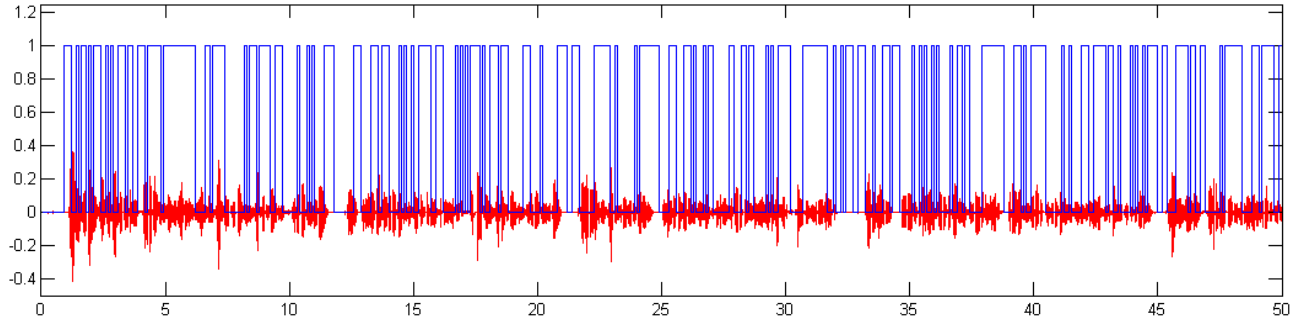
Rys. 5.2: Drugi sygnał testowy, charakteryzujący się niskimi poziomami mocy i szumu

Przypadek ten wyraźnie ukazuje niedokładność wybranych parametrów wobec takiego sygnału. Związana z niskimi amplitudami równie niska energia spowodowała, że granica pomiędzy zerem a wartością progową MED, która decyduje o uznaniu ramki za mówioną jest bardzo wąska. W wypadku tym swoją rolę lepiej spełniłoby rozwiązanie proponowane przez twórców idei parametru MED [3] w postaci arbitralnego ustawienia wartości progowej zgodnie ze wzorem (5). Postać badanego sygnału negatywnie wpływa na parametr ZCR prowokując algorytm do przekłamań (widocznych pików dla pozornie cichych fragmentów sygnału). Powodami jest oscylacja wartości wokół zero, które zmieniając znak jednocześnie podnoszą wartość ZCR i przez to karmią decyzyjną część algorytmu nieprawdziwym wnioskiem o aktywności mowy w danej ramce czasowej. Drugim powodem jest widoczna na początku sygnału ujemna, krótkotrwała wartość sygnału uznana przeze mnie za szum – na tyle jednak mocny, że wpływający na wielkość obliczanej wartości progowej parametru ZCR. Wykres 5.3 prezentuje dalszą część badanego pliku (w czasie trwania od siódmej do dziewiątej sekundy sygnału). Wybrany został on przeze mnie jako wyjątkowo dobrze ukazujący wpływ omawianych cech tego sygnału na pojawiające się błędy.



Rys. 5.3: Część dalsza sygnału, widoczne są tu błędne decyzje algorytmu spowodowane cechami sygnału.

Trzeci sygnał testowy – niska moc, zauważalny szum.



Rys. 5.3: Trzeci sygnał testowy, domyślne parametry wejściowe.

Trzeci sygnał, różniący się od poprzedniego gęstością oraz poziomem energii jest też zauważalnie bardziej dynamiczny. Jako, że w tym wypadku średni poziom energii jest wartością niemiarodajną, to odpowiedzialność za decyzję algorytmu bierze na siebie tutaj rozrzut amplitud sygnału – jego entropia. Zgodnie z rozdziałem 2.1 wysoka jej wartość (w badanej ramce), do pary z widoczną wysoką częstotliwością przejść sygnału przez zero, rozpatrywana jest jako pojawienie się mowy ludzkiej. W takim sygnale udostępnione użytkownikowi parametry wejściowe mogą stanowić **klucz** do poprawnej analizy.

W swojej pracy nie tylko stworzyłem bibliotekę pozwalającą w szybki i prosty sposób na wykrywanie obszarów mowy ludzkiej w podanym sygnale dźwiękowym. Wybrane do części decyzyjnej parametry są uznawane za dla tej dziedziny podstawowe i często problematyczne dla różnej natury sygnałów – co udowodniłem. Założeniem, którym kierowałem się projektując i tworząc bibliotekę była jak najmniejsza ingerencja użytkownika w nią. Dlatego wszelkie wartości progowe wyliczane są na podstawie początkowych fragmentów pobranych z plików. Wpływa to bezpośrednio na osiągi algorytmu decyzyjnego – uzależniając go od danych wejściowych. Możliwym rozwiązaniem byłoby wprowadzenie sztywnych wartości progowych uzyskanych w sposób empiryczny – na podstawie bardzo dokładnych i osobistych testów, z parametrem tym wprowadzanym jako parametry funkcji wewnętrznych. Wymagałoby to jednak przebudowy biblioteki i rozbudowania jej API – co można uznać za potencjalną furtkę do dalszego rozwoju aplikacji. Nie wolno zapominać też o niewykorzystanych tutaj, bardziej rozbudowanych, ale pracujących z o wiele większą dokładnością mechanizmach – jak badanie na podstawie współczynników cepstralnych MFCC z wykorzystaniem modelu GMM lub algorytmu DTW (ang. *Dynamic Time Warping*).

W słowie końcowym, za najważniejszy wniosek, który wyciągnąłem z tej pracy, uznaję wiedzę o złożoności podjętego przeze mnie tematu. Pozorna prostota zaimplementowanych przeze mnie obliczeń zderzyła się z ogromem wpływających na nie czynników zewnętrznych – podane parametry wejściowe biblioteki, jakość pliku dźwiękowego, sposób jego nagrania, czy siły sygnału użytecznego względem szumu. Dalsze prace nad biblioteką przełożyłyby się bezpośrednio na jej dokładność, najprawdopodobniej możliwe byłoby również skondensowanie jej kodu i oparcie o bardziej skomplikowane, ale efektywniejsze, rozwiązania języka C++.

W obecnej postaci biblioteka spełnia jednak wszystkie swoje założenia.

6. Bibliografia

[1] Ramírez, J., Segura, J.C., Benítez, C., de la Torre, Á., Rubio, A.J., „*A New Kullback-Leibler VAD for Speech Recognition in Noise*”, IEEE Signal Processing letters 11(2), 266–269 (2004)

[2] 3GPP TS 26.094 V6.1.0, Voice Activity Detector (VAD) for Adaptive Multi-Rate speech codec (2006)

[3] S. Kacprzak, M. Ziółko „*Speech/music discrimination via energy density analysis*” Dostępny: http://www.dsp.agh.edu.pl/_media/pl:kacprzakziolkotarragonaslsp2013.pdf (odwiedzona 21.12.2015).

[4] Kedem B., „*Spectral Analysis and Discrimination by Zero-Crossings*”, Proceedings of the IEEE, Vol. 74, No. 11, November 1986.

[5] Voice activity decision base on zero crossing rate and spectral sub-band energy . Patent US 8296133 B2 . Dostępny: https://www.google.com/patents/US8296133 (odwiedzony 21.12.2015)

[6] Microsoft WAVE soundfile format. Dostępny: http://soundfile.sapp.org/doc/WaveFormat/ (odwiedzona 21.12.2015)

[7] Julien Pinquier, Jean-Luc Rouas, Régine André-Obrecht, „*Robust speech / music classification in audio documents*”, 7th International Conference on Spoken Language Processing, ICSLP2002 - INTERSPEECH 2002, Denver, Colorado, USA, Wrzesień 16-20, 2002.

[8] Philippe Renevey, Andrzej Drygajlo, „*Entropy Based Voice Activity Detection in Very Noisy Conditions*” Dostępny: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.6098&rep=rep1&type=pdf (odwiedzona 21.12.2015)

[9] Ying, G. S., C. D. Mitchell, and L. H. Jamieson. "*Endpoint detection of isolated utterances based on a modified Teager energy measurement*." Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on. Vol.

2. IEEE, 1993

[10] Ryszard Tadeusiewicz, *Sygnał mowy*, Wydawnictwo Komunikacji i Łączności, Warszawa, 1988.