POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

PROEJKT Z PROJEKTOWANIA SYSTEMÓW Z DOSTĘPEM W JĘZYKU NATURALNYM

Sterowanie telewizorem za pomocą komend głosowych

AUTORZY:
Michał Czapowski 181225
Grzegorz Grzegorczyk 181121

PROWADZĄCY ZAJĘCIA:
Dr inż. Dariusz Banasiak

OCENA PRACY:

Wrocław 2013

Spis treści

1. Wstęp

Celem projektu jest stworzenie programu, który będzie rozpoznawał i klasyfikował wypowiadane komendy. Zbiór słów ma służyć obsługą telewizora. Komendy będą pojedynczymi słowami, np. „włącz”, „przełącz”.

Program może posłużyć jako część softwarowa większego projektu, którego celem byłoby stworzenie urządzenia służącego do obsługi telewizora przy pomocy języka naturalnego. Z tego powodu program został napisany przy pomocy języka Python, który można w prost sposób uruchomić na płycie Raspberry Pi. Do takiej płytki można dodać diodę (IR) przez którą można sterować telewizor, która byłaby sterowana właśnie przy pomocy komend w języku naturalnym.

2. Opis projektu

Do projektu wykorzystano skryptowy język programowania Python. Do analizy dźwięku posłużyły biblioteki:

- `scipy.signal` – biblioteka zawiera wiele popularnych typów filtrów
- `pyaudio` – biblioteka do obsługi dźwięku z mikrofonu
- `scipy.fftpack` – umożliwiła wykonanie transformaty Fouriera na próbkach sygnału
- `mealfeat.py` – biblioteka umozliwiająca wyznaczeniewspółczynników MFCC

Słownik z komendami zawiera 10 słów:

- jedynka
- dwójka
- następny
- poprzedni
- podległość
- ścisz
- wyłącz
- włącz
- wycisz
- uruchom

Przed uruchomieniem programu posiadamy już przygotowane wektory klasyfikatorów dla poszczególnych komend wyliczone uprzednio na podstawie 10 próbek. Działanie programu opiera się na trzech etapach:

1) Nagranie słowa (niekoniecznie komendy)

2) Analiza słowa (wygenerowanie wektora cech)

3) Przypisanie słowa do klasy (porównanie ze słownikiem komend)

3. Zarys prac nad rozpoznawaniem mowy

3.1 Klasyfikacja systemów rozpoznawania mowy

Systemy rozpoznawania mowy można podzielić ze względu na różne kategorie:

- systemy dedykowane dla mówcy
- systemy ze słownikiem (*command and control*), lub dowolnymi słowami
- zakres rozpoznawanych słów
- systemy czasu rzeczywistego, lub offline'owe

Nasz system jest przystosowany tylko dla jednego mówcy. Wykorzystamy przygotowany przez nas słownik komend. Zbiór słów będzie niewielki, dlatego zakres systemu określamy

jako mały. System powinien umożliwić użytkownikowi sterowanie urządzeniem (telewizorem), dlatego musi działać szybko i podejmować decyzje online.

3.2 Podejścia do rozpoznawania mowy

3.2.1 Podejście akustyczno-fonetyczne

Algorytm opiera się na podziale wypowiedzianego słowa na fonemy, czyli najmniejsze jednostki mowy. Algorytm dzieli nagrane słowo na takie właśnie fonemy. Podział na te jednostki opiera się na rozpoznawaniu cech sygnału charakterystycznych dla danego fonemu. Po takiej analizie otrzymujemy zapis fonetyczny słowa, wystarczy tylko sprawdzić znaczenie danego zapisu fonetycznego. Wadą jest nietrywialna zamiana sygnału (niekoniecznie słowa) na zapis fonetyczny. Utrudnieniem jest również posiadanie zapisu fonetycznego dla każdego słowa ze słownika.

3.2.2 Podejście wykorzystujące rozpoznawanie wzorców

Metoda zakłada,że każde słowo posiada charakterystyczne cechy zmian amplitudy sygnału. Działanie algorytmu polega na podziale słowa na krótkie odcinki sygnałów a następnie zmierzenie amplitudy w każdym z przedziałów. Wyliczone wartości stworzą wykres cech sygnałów. Słabością tego algorytmu jest jego założenie. Nie każde słowa mają wyraźną różnicę w widmie amplitudowym. Dlatego pojawiają się problemy podczas rozpoznawania podobnie brzmiących słów. Z tego powodu metoda ta sprawdza się dla niewielkich i odpowiednio dobranych słowników. Przykładem takiego zbioru są cyfry. Metoda ta daje dobre wyniki przy analizie tego popularnego zbioru, a jednocześnie jest ona bardzo prosta w działaniu i implementacji. Utrudnieniem jest „uczenie" systemu, które jest potrzebne aby stworzyć zbiór wzorców.

3.2.3 Podejście wykorzystujące sztuczną inteligencję

Mechanizmy z tej rodziny opierają się na różnych algorytmach sztucznej inteligencji. Do tego celu wykorzystują metody hybrydowe, które bazują na analizie fonetycznej, ale jednocześnie porównuje słowo (lub jego fragmenty) z wyuczonymi wzorcami.

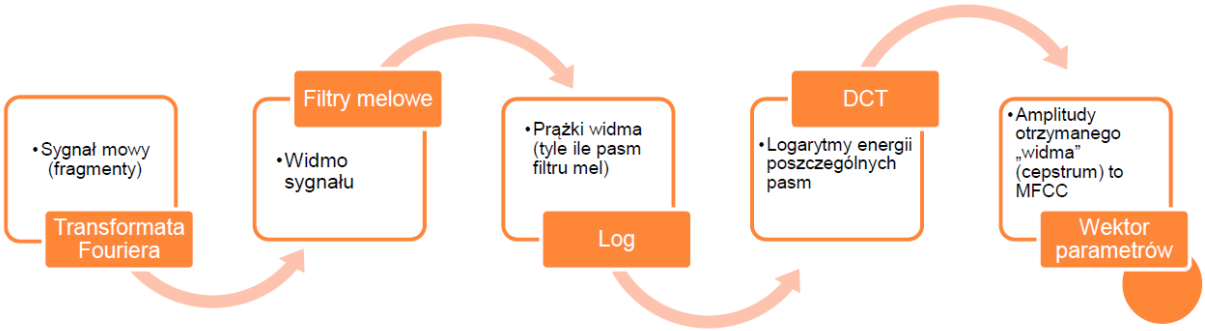
3.3 Rodzaje współczynników do rozpoznawania mowy

3.2.4 Linear Predictive Coding (LPC)

To podejście polega na na wyznaczaniu współczynników LPC wyliczane z krótkich fragmentów sygnału. Najpierw liczone są współczynniki autokorelacji pomiędzy próbkami ramki. Następnie, z nich uzyskuje się współczynniki LPC stosując algorytm Levinsona-Durbina. Metoda ta umożliwia zaoszczędzenie pamięci (przebieg sygnału w czasie nie jest zapisywany - jedynie pojedyncza ramka) oraz zaoszczędzenie czasu (nie ma ekstrakcji cech sygnału po nagraniu, a w trakcie nagrywania).

3.3.1 Mel-Frequency Cepstral Coefficients (MFCC)

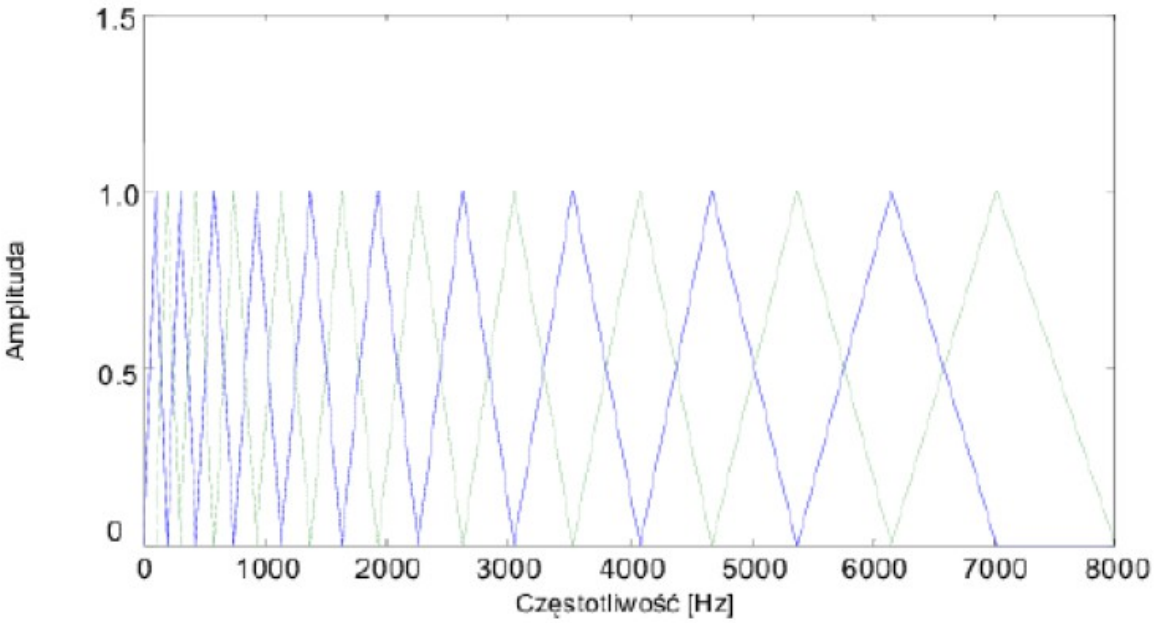
Współczynniki otrzymuje się ze spektrum sygnału przedstawionego w skali melowej (mel-cepstrum). W naszym projekcie wykorzystaliśmy właśnie te współczynniki, dlatego zostanie ona opisana szerzej.



Ilustracja 1: Etapy pozyskiwania współczynników MFCC

Pierwszym krokiem jest podział sygnału na fragmenty, każde takie okno sygnału posłuży do obliczenia jednego współczynnika. Następnie obliczamy widmo częstotliwości, na przykład przy pomocy szybkiej transformaty Fouriera. Kolejnym krokiem jest przepuszczenie sygnału przez filtr melowy. Filtr ten jest trójkątny, o wzorze (wg. Beranka):

$$F_{mel}(f_{kHz}) = 1127 * \ln\left(1 + \frac{f_{kHz}}{0.7}\right)$$



Ilustracja 2: Wykres filtru melowego

Po wyliczeniu sygnału otrzymamy prążki widma, otrzymany wykres należy jeszcze zlogarytmować, ponieważ skala molowa jest logarytmiczna. Aby uzyskać lepsze dopasowanie współczynników warto jeszcze wykonać na sygnale dyskretną transformatę cosinusową (DCT).

Wektor cech będzie tak długi jak długi (ilość pasm filtra) jest filtr melowy. Będą to cechy opisujące wycinek sygnału. Dlatego ten proces należy powtórzyć dla każdego okna sygnału. W rezultacie otrzymuje się macierz cech, opisujących pojedynczą próbkę (słowo).

4. Implementacja

3.3 Struktura plików projektu

Folder z projektem zawiera:

- katalog z próbkami dźwięku kilku komend
- katalog z plikami tekstowymi z już obrobionymi wartościami cech MFCC
- plik RealTimeVoiceAnalyzer.py - moduł główny łączący wszystko, pozwala na przeprowadzenie testu jaka jest skutecznosc rozpoznawania komend oraz pozwala na rozpoznawanie komend w czasie rzeczywistym czyli poprzez podawanie ich do mikrofonu
- plik mealfeat.py - klasa do wyznaczania współczynników MFCC
- plik RecordModule.py - to modul do nagrywania naszych próbek komend, w czasie działania właściwego programu funkcje z tego modułu odpowiadają za nagrywanie dźwięku, oraz wykrycie słowa
- plik PlotModule.py - moduł do rysowania wykresów
- plik VoiceCommand.py - do przechowywania probek komend oraz wspołczynników MFCC

3.4 Przechowywanie próbek komend

W obiektach klasy `VoiceCommand` przechowujemy informacje o próbkach, zarówno tych wyuczonych, jak i komendy analizowanej. Klasa ta przechowuje informacje o położeniu katalogu w którym znajduje się plik w którym zapiszemy informacje o wektorze cech MFCC. Zapisujemy również długość próbki, tym samym informacje o długości komendy. Atrybuty `learned_ceps` i `learned_ceps_abs`, są tablicami przechowującymi wartości próbek, a w późniejszym etapie analizy wartości współczynników MFCC.

```
class VoiceCommand(object):
    name = ''
    folderPath = ''
    numOfSpeech = 0
    uniqueId = 0
    learned_ceps = []
    learned_ceps_abs = []
```

Tekst 1: Atrybuty klasy `VoiceCommand`

4.1 Pozyskiwanie sygnału ze źródła (mikrofonu)

Nasłuchiwanie i nagrywanie dźwięku wykonuje się dzięki metodą z pliku `RecordModule`. Pomocna okazała się biblioteka `pyaudio`. Umożliwia sterowanie mikrofonem, obiekt tej klasy jest połączeniem z portem mikrofonu. W naszej metodzie `record` służy on nam do czytania z tego portu.

```
def record():

    p = pyaudio.PyAudio()
    stream = p.open(format=pyaudio.paInt16, channels=CHANNELS,
             rate=44100, input=True, output=True,
             frames_per_buffer=1024)
```

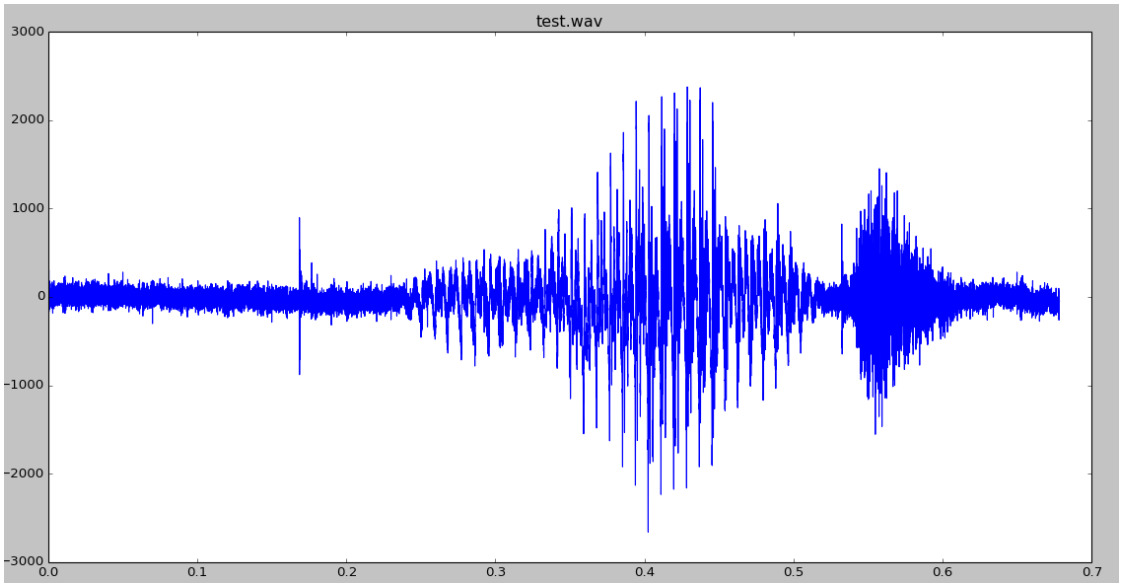
Tekst 2: Fragment metody `record`, przedstawiający łączenie z mikrofonem

Konfiguracja połączenia jest przedstawiona powyżej: częstotliwość próbkowania to 44100 ramek na sekundę, a każda ramka ma rozmiar 1024 bitów. Po ustanowieniu połączenia można nagrać dźwięk o określonej liczbie ramek. Zdecydowaliśmy się wstępnie analizować dźwięk, dlatego pobieramy ramki pojedynczo i sprawdzamy, czy jest to dźwięk cichszy od wybranego przez nas progu, jeśli nie wtedy próbka trafia do danych wyjściowych `data_all`.

```
while True:
        data_chunk = array('h', stream.read(1024))
        if byteorder == 'big':
            data_chunk.byteswap()
        data_all.extend(data_chunk)

        silent = is_silent(data_chunk)

        if audio_started:
            if not silent:
                silent_chunks = 0
                speech_chunks += 1
                if speech_chunks > MAX_LENGTH_SEC :
                    break

            if silent :
                silent_chunks += 1
                if silent_chunks > SILENT_CHUNKS :
                    if speech_chunks > SPEECH_CHUNKS:
                        break
                    else:
                        data_all = data_all[len(data_all) -
                                    silent_chunks : len(data_all)]
                        speech_chunks = 0
                        silent_chunks = 0
                        audio_started = False

        elif not silent:
            audio_started = True
            speech_chunks = 0
            silent_chunks = 0
```

Tekst 3: Wycinanie szumu z sygnału

Po nagraniu już próbki o określonej przez nas długości trzeba zamknąć połączenie z mikrofonem, oraz bufor z którego czytamy dane. Sygnał nagrany przy pomocy tych metod wygląda jak na rysunku poniżej.



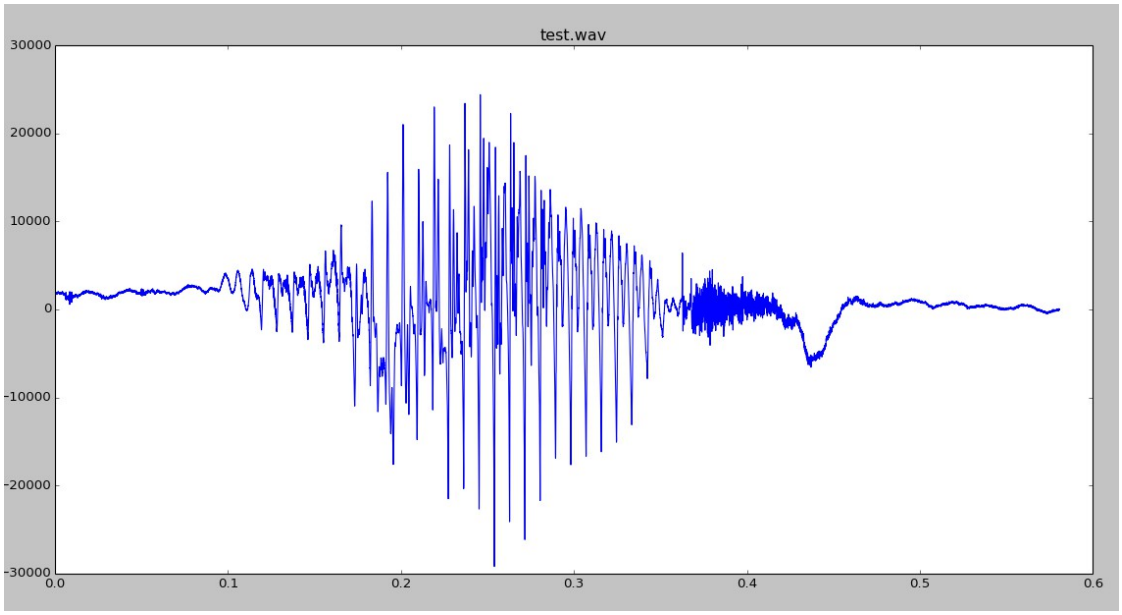
Ilustracja 3: Nagrany sygnał, bez usunięcia zakłóceń

Nim jednak przekażemy dane na wyjście funkcji należy jeszcze wzmocnić składowe wyższych częstotliwości (pre-emfaza), oraz znormalizować wartości próbek. Stąd metoda *trim*.

```
stream.stop_stream()
stream.close()
p.terminate()

data_all = trim(data_all)
data_all = normalize(data_all)
```

Tekst 4: Zamknięcie połączenia z mikrofonem, oraz obróbka danych

Po obróbce sygnału otrzymujemy oczyszczony z zakłóceń dźwięk.



Ilustracja 4: Sygnałbez zakłuceń

Dane otrzymane przez metodę `record` są zbyt obszerne, dlatego warto zmniejszyć ilość naszych próbek, czyli zmniejszyć częstotliwości próbkowania. Będzie to wykonywane w metodzie `getSpeechFromMic`, i to tą metodę będziemy wywoływać w naszym programie do rozpoznawania dźwięku.

```
def getSpeechFromMic():

    data = record()
    y = numpy.atleast_2d(data)[0]
    timp=len(y)/RATE
    t=numpy.linspace(0,timp,len(y))
```

Tekst 5: Zmniejszenie częstotliwości danych

4.2 Algorytm detekcji pojedynczego słowa (algorytm Rabinera-Sambura)

Przed rozpoczęciem analizy należy podzielić dźwięk na próbki o mniejszych długościach. Tą operacje zrobiliśmy już na początku, kiedy pobieraliśmy dźwięk z mikrofonu (metoda `record`). Kolejnym krokiem jest przyjęcie średniego poziomu energii słowa. Wartość tą można obliczyć na podstawie próbek początku słowa. Konsekwencja tego kroku jest, to że nagranie (tym samym słowo) musi być wystarczająco długie, aby móc wyliczyć średnią wartość energii i aby mieć choćby kilka próbek aby ją do nich przyrównać. Początkowe próbki pozwalają na obliczenie ilości przejść przez zero sygnału.

Energia m-tej ramki sygnału s liczona jest według:

$$E_s[m] = \sum_{n=(m-1)N}^{m_{(N-1)}} |s[n]|$$

gdzie:

N – liczba próbek w ramce,
m – numer ramki,
n – numer próbki sygnału.

Następnie obliczana jest maksymalna energia sygnału IMX oraz energia szumu IMN pierwszych 100ms. Na ich podstawie liczone są następnie parametry I1 i I2, zgodnie z:

$$I1 = 0.03(IMX - IMN) + IMN$$

oraz

$$I2 = 4 * IMN$$

Ze wzoru widać, że I1 jest w przybliżeniu wartością odpowiadającą 3% wartości szczytowej energii IMX. Z kolei zgodnie z drugim wzorem odpowiada czterokrotnej wartości energii szumu tła IMN.

Następnie z parametrów I1 i I2 wybierany jest ten o mniejszej wartości, który następnie definiuje niższy próg ITL według wzoru:

$$ITL = min(I1, I2)$$

Górny próg ITU obliczany jest na podstawie znajomości niższego progu ITL zgodnie z:

$$ITU = 5 * ITL$$

Obliczany jest próg liczby przejść przez zero dla szumu tła zgodnie z:

$$IZCT = min(IF, IZC + 2 * \delta_{IZC})$$

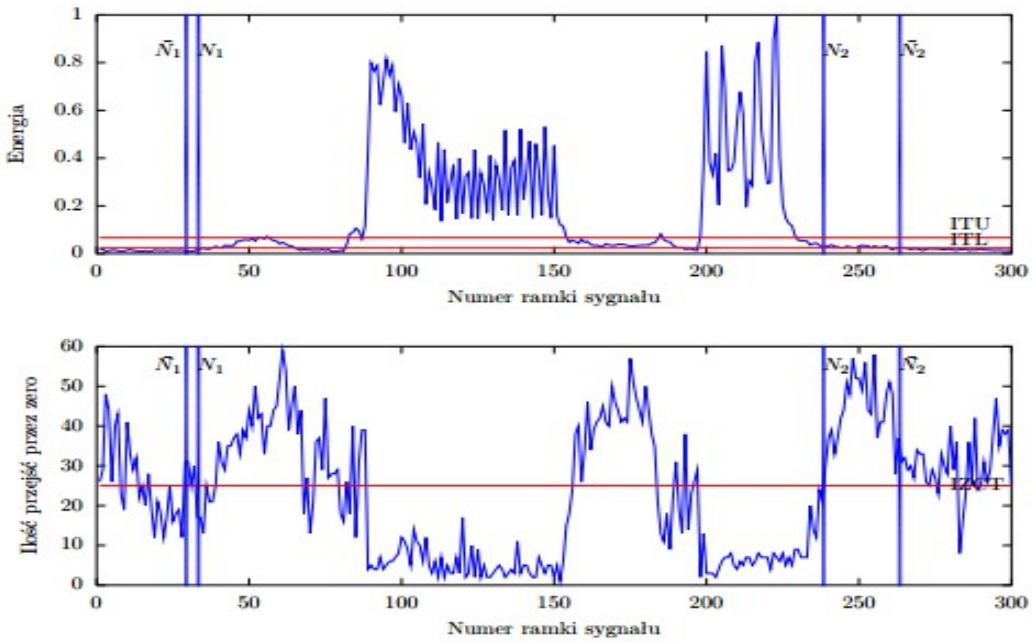
gdzie:

IF – 25 przejść przez zero w ciągu 10ms,

IZC – średnia zmierzona liczba przejść przez zero dla szumu tła mierzona w ciągu pierwszych 100ms sygnału,

δizc – odchylenie standardowe liczby przejść przez zero.

Przykładowy wynik wartości progów, oraz analizowany sygnał jest umieszczony na poniższej ilustracji. Jak widać wartości progów opisanych powyżej zgadzają się z naszą intuicją.



Ilustracja 5 Sygnał mowy z oznaczonymi progami algorytmu Rabinera-Sambura

Uzupełnienie algorytmu znajdywania początku i końca słowa, bazującego na pomiarze energii sygnału, o algorytm liczby przejść przez zero powinno zwiększyć dokładność wyboru początku. Metoda implementująca ten algorytm znajduje się w pliku `RecordModule`, w metodzie `detectSingleWord`.

```
for m in range(0,frames,1):
    if(wordstart > 0):
       break

    if(wordspower[m] >= ITL ):
       for i in range(m, frames, 1):
           if(wordspower[i] < ITL ):
                break
           else:
                if(wordspower[m] >= ITU ):
                    wordstart = i
                    if(i == m):
                        wordstart = wordstart-1
                    break
```

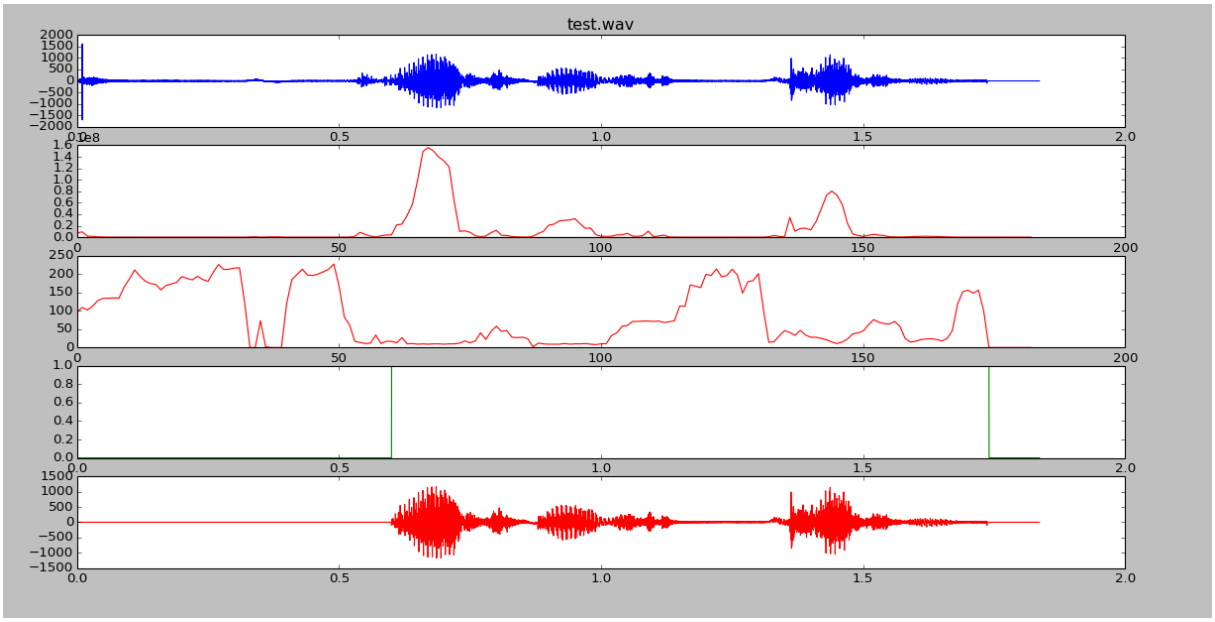
Tekst 6: Detekcja początku słowa

Wykrywanie początku słowa i końca jest analogiczne. Aby próbka nie była zbyt krótka trzeba wziąć pod uwagę opadający ton głosu. Aby algorytm nie ucinał końcówek słów trzeba sprawdzić czy sygnał przechodzi przez zero po wykrytym końcu słowa. Ilustruje to poniższy kod.

```
for i in range(wordstop, frames-1, 1):
        if(hi < wordstop and wordszeros[i] >= IZCT ):
            hi  = i
        if(hi>= wordstop and wordszeros[hi+1]<IZCT and (150>(i -
wordstop)*framelen)) :
            wordstop = hi+1
            break
```

Tekst 7: Przemieszczenie znaku końca słowa z uwzględnieniem liczby przejść przez zero.

Działanie tego algorytmu ilustruje poniższy wykres. Na pierwszym wykresie:



Ilustracja 6: Kolejne kroki działania algorytmu detekcji słowa (słowo "dziesięć")

Pierwszy wykres przedstawia słowo po dokonanej preemfazie. Kolejny wykres ilustruje zmianę mocy sygnału w czasie. Trzeci wykres przedstawia liczbę przejść sygnału przez wartość zero. Jak widać na wykresie fragmenty gdzie występuje sam szum tła (teoretyczna 'cisza') są łatwe do zidentyfikowania. Czwarty wykres ilustruje wyliczone jednostki czasu w których rozpocznie i zakończy się słowo. Kiedy sygnał wejściowy przepuścimy przez filtr o takim właśnie kształcie otrzymamy sygnał widoczny na ostatnim wykresie.

4.3 Parametryzacja sygnału – MFCC

Do obliczania parametrów metoda banków filtrów, do tego wykorzystujemy open-sourc'ową bibliotekę `mealfeat.py` . Biblioteka daje do dyspozycji metody umożliwiające obliczanie z próbki dźwiękowej widma w skali melowej, oraz wygenerowanie wektora cech MFCC. Metoda `calcMelFeatures` automatycznie oblicza wektor cech dla każdej próbki dźwięku.

```
def calcMelFeatures(self, data):
      outTuple = self.stft(data)

      wts = self.filtbank(self.numFilts, self.minfrq, self.maxfrq,
            self.width, nfft)
```

Tekst 8: Przygotowanie sygnału do obliczenia wektora cech MFCC

Najpierw obliczane jest widmo częstotliwościowe próbki dźwięku, służy do tego metoda w tej klasie `stft`. Następnym krokiem jest utworzenie banku filtrów o zadanych

parametrach. Obiekt `self` (czyli obiekt tej klasy) jest tworzony w klasie głównej która zarządza całym procesem rozpoznawania mowy i tam szczegółowiej opiszemy wartości jakie posłużyły nam do stworzenia naszych filtrów.

```
P   = np.dot(wts,Xp)

Q = np.log(P);

C = self.dct(Q,self.numcepBasic)

C_cmn = self.cmn(C);

d1 = self.deltas(C_cmn,self.del_w)
d2 = self.deltas(d1,self.dbl_del_w)
```

Tekst 9: Wyliczanie współczynników MFCC

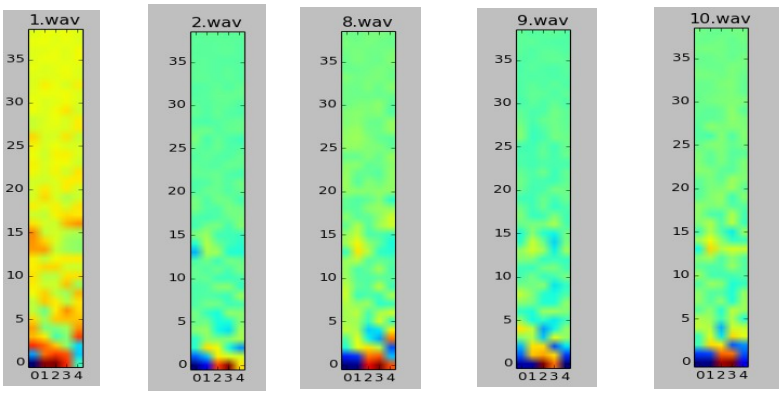
W pierwszym kroku mnożymy sygnał wejściowy przez wektor banku filtrów. Wynik działania należy zlogarytmować, wynika t z charakteru filtrów wykorzystanych w programie. Następnie dokonujemy na próbkach dyskretnej transformaty cosinusowej (DCT). Oto całe działanie algorytmu. Teraz wystarczy wyliczyć deltę (czyli pierwszą pochodną) i drugą deltę (drugą pochodną). W zmiennej `C_out` znajduje się 13 pierwszych współczynników MFCC, `d1` i `d2` przechowują odpowiednio pierwszą i drugą pochodną współczynników.

Nasze rozwiązanie nie wymaga tak dokładnych obliczeń, do rozpoznawania komend wystarczy znajomość wektora cech będącego największą wartością spośród kilku mu sąsiadujących. Dlatego zmodyfikowaliśmy metodę:

```
sizeBand = int(len(C_out.T) / self.numcepsBands)
if(sizeBand==0):
    sizeBand=1
lpBand = 0
C_out2 = [[0 for x in range(self.numcepsBands)]
           for y in range(self.numallceps)]

if(True):
   for kk in range(self.numallceps):
       for ll in range(self.numcepsBands) :
           amplMax = max(C_out[kk][ll*sizeBand:ll*sizeBand+sizeBand])
           amplMin = min(C_out[kk][ll*sizeBand:ll*sizeBand+sizeBand])
           if amplMax > abs(amplMin) :
                C_out2[kk][ll] = amplMax
           else :
                C_out2[kk][ll] = amplMin
```

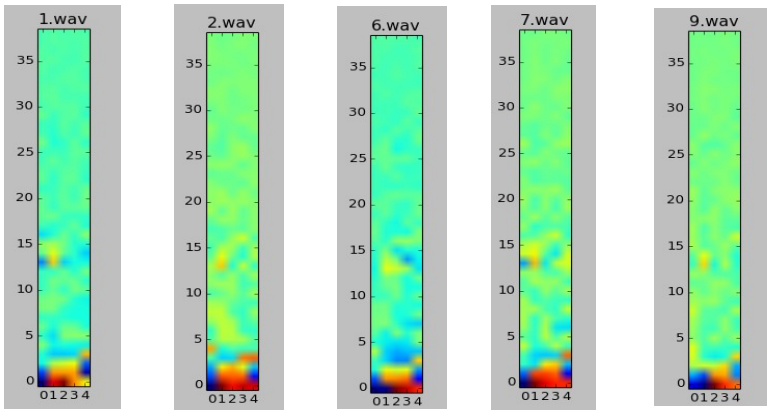
Tekst 10: Obliczanie największych wartości wektora cech dla kilku sąsiednich próbek

Teraz macierz będąca wynikiem działania tej funkcji (`C_out2`) zawiera w pojedynczym wierszu największe, odpowiadające wartości spośród 6-ciu sąsiednich wektorów cech próbek. Widma capstralne można przedstawić na wykresach dwu-wymiarowych:



Rysunek 1: Widma capstralne dla komendy "wyłącz"

Dla porównania zamieszczamy widma capstralne innej komendy „podgłoś”.



Rysunek 2: Widma capstralne dla komendy "podgłoś"

Barwa bliższa odcieniowi czerwieni oznacza wyższą wartość cechy, natomiast im kolor bliższy do niebieskiego tym niższy. Oba pokazane zestawy cech różnią się siłą, między innymi, w pierwszej składowej. Dzięki takim właśnie różnicą rozróżniamy poszczególne słowa i klasyfikujemy je.

4.4 Moduł główny – analiza i klasyfikacja mowy

Głównym plikiem który zarządza wszystkimi elementami jest klasa `RealTimeVoiceAnalyzer`. Działanie tej klasy zaczyna się w metodzie `go`.

```
def go():
    t, y = RecordModule.getSpeechFromMic()
    predict = getCepsMatrixFromData(t, y)
    getClasificationDecision(predict)
```

Tekst 11: Ciało metody `go`

Ta krótka metoda inicjuje to co najważniejsze. Przy pomocy metody `getSpeechFromMic` z klasy `RecordModule` pobiera dźwięk z mikrofonu. Drugim krokiem jest wyliczenie widm capstralnych, dzięki metodzie `getCepsMatrixFromData`. Na końcu wystarczy tylko porównać otrzymany wynik analizy z bazą widm wszystkich komend ze słownika (metoda `getClasificationDecision`).

```
def getCepsMatrixFromData(t,y):
    y = RecordModule.preemp(y)
    fr, wordspower, wordszeros, wordsdetect, ITL ,ITU,  word_fr,
                      word_y = RecordModule.detectSingleWord(t,y)
    MelFeat = mealfeat.MelFeatures()
    ceps_matrix = MelFeat.calcMelFeatures(word_y)
```

Tekst 12: Ciało metody `getCepsMatrixFromData`

Metoda ta rozpoczyna od preemfazy sygnału (metoda `preemp` z klasy `RecordModule`) . Tak przygotowany sygnał poddajemy analizie algorytmowi Rabinera-Sambura, aby wyodrębnić słowo z zadanej próbki dźwięku. Kiedy jesteśmy w posiadaniu słowa wystarczy wyliczyć jego widmo capstralne przy pomocy obiektu klasy `mealfeat` i zmienionej przez nas metody `calcMelFeatures`. I owocem działania tej metody jest macierz cech MFCC.

4.5 Klasyfikacja sygnału mowy

Proces dopasowania próbki odbywa się w metodzie `getClasificationDecision`. Mechanizm bazuje na trzech krokach. Najpierw oblicza wartości algorytmów NN, alfa NN i NM, następnie oblicza sumę ważoną tych współczynników dla każdej komendy ze słownika, aby w ostatnim etapie wskazać komendę której suma ważona współczynników jest najwyższa.

```
def getClasificationDecision(predict):
    NNRes = nearestNeighbour(predict)
    NMRes = nearestMean(predict)
    NANRes = nearestAlfaNeighbour(predict, 20)

    for i in range(len(learned_speech_tab)):
        ALLRes[i] = (0.2*NNRes[i]) + (0.5*NMRes[i]) + (0.3*NANRes[i])
        ALLRes[i] = ALLRes[i]*100.0

    ai=0
    max_val = -1
    for i in range(len(learned_speech_tab)):
        if ALLRes[i] > max_val :
            ai = i
            max_val = ALLRes[i]

    if(max_val < prog_Komenda_nieznana):
        ai = len(learned_speech_tab)+1

    return ai
```

Tekst 13: Ciało metody `getClasificationDecision`

Należy również wspomnieć, że istnieje jeszcze jeden krok, który służy do klasyfikacji słów nie będącymi znanymi komendami.

4.5.1 NN - najbliższy sąsiad

Jednym z podstawowych algorytmów służących do klasyfikacji mowy jest algorytm najbliższego sąsiada. Proces dopasowania osobników opiera się na liczeniu podobieństwa między dwoma wektorami cech. Przykładowo odległość dwóch punktów od środka w układzie kartezjańskim można wyrazić jako pierwiastek sumy kwadratów składowych X i Y.

Różne cechy opisujące osobniki mogą wymagać różnych sposobów miar odległości. W tym projekcie zastosowaliśmy miarę odległości Euklidesowej.

$$\rho(x,y)=(x-y)'(x-y)=(\sum_{\rho}^{i=1}(x_i-y_i)^2)^{\frac{1}{2}}$$

Metryka ta jest niczym innym jak sumą kwadratów różnic odpowiadających sobie cech. W przypadku macierzy parametrów MFCC polega to na wyliczeniu takiej sumy dla każdej pozycji macierzy (każdej cechy). Aby większa wartość dopasowania wektorów oznaczała lepsze dopasowanie musimy obliczyć odwrotność sumy.

```
def fSimilatiry( v1, v2 ):
    ret = 0.0

    for j in range(len(v1)):
        ret += (v1[j] - v2[j])**2
    ret = math.sqrt(ret)

    ret = 1.0/ret

    return ret
```

Tekst 14: Metodaliczqca podobieństwo dwóch wsektorów cech (alg. Euklidesa)

Wynikiem działania tego algorytmu jest wartość dopasowania, oraz identyfikator najlepiej dopasowanej komendy. W kodzie programu metoda licząca NN, jest przypadkiem działania algorytmu *alfa* NN, gdzie *alfa* przyjmuje wartość 1.

4.5.2 *Alfa* NN - alfa najbliższych sąsiadów

Algorytm *alfa* najbliższych sąsiadów, ma na celu wyznaczenie kilku (parametr *alfa*) najbliższych sąsiadów. Algorytm działa identycznie z tą jedną różnicą, że rozpatrzmy więcej niż jeden wektor cech. Działanie algorytmu przypomina wstępną selekcję.

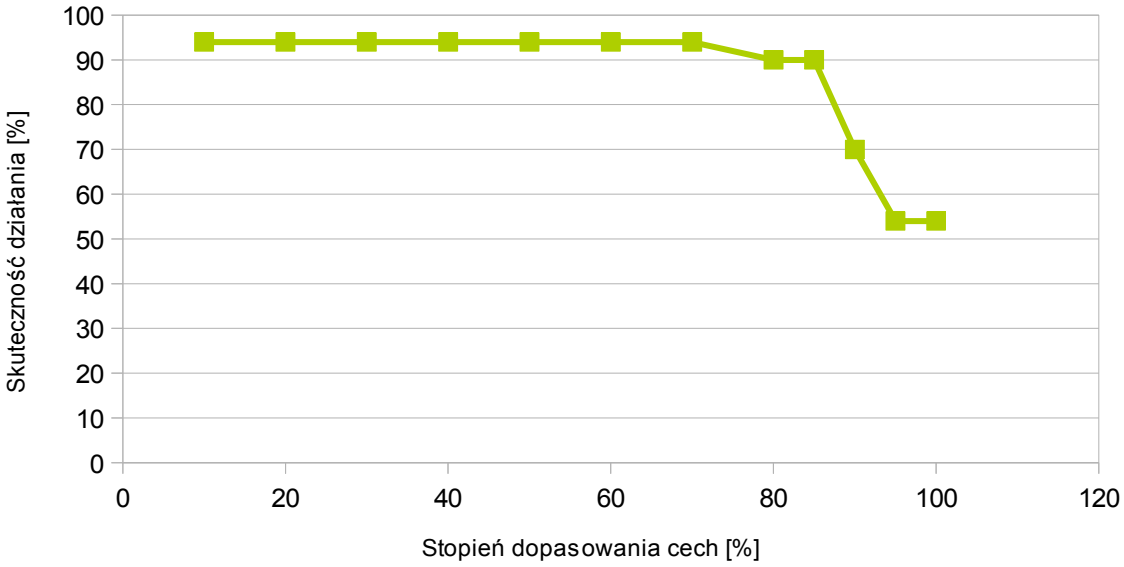
Samo działanie algorytmu jest proste. Najpierw wyliczana jest wartość odległości (dopasowania) klasyfikowanego słowa względem każdego słowa ze słownika. Następnie wektor tych odległości jest sortowany malejąco, w ten sposób dostaniemy wartości największe. Zwracamy wartość mody, czyli wartość klasy występującej najczęściej w zbiorze *alfa*.

4.5.3 NM - najbliższa średnia

Ten algorytm, w przeciwieństwie do algorytmów najbliższego sąsiada, nie oblicza dopasowania do próbki ze słownika, ale odległość do sygnału wzorcowego. To względem tego wektora cech mierzone i porównywane są wszystkie wektory cech zarówno te ze słownika, jak i próbka klasyfikowana.

5. Testy i wyniki

Wynikiem tego projektu jest program który wykrywa słowa z nagranych próbek dźwiękowych, oraz umożliwiający poprawną klasyfikację wykrytych słów jako poszczególne komendy mogące sterować telewizorem. Program jest napisany w całości w języku Python, dzięki czemu jest przenoszalny i posiada nieskomplikowaną składnie i budowę. Przeprowadziliśmy szereg testów mających zweryfikować poprawność działania programu, wyniki reprezentuje poniższy wykres.



Ilustracja 7: Wykres skuteczności działania

Powyższy rysunek ilustruje jakość klasyfikacji przez nasz program. Uzyskaliśmy średnią skuteczność klasyfikacji na poziomie 94%. Okazuje się, że istotny parametr którym jest stopień dopasowania cech ma wpływ dopiero, jeśli będą odrzucane próbki o zgodności mniejszej niż 80%. Oznacza to, że słowa testowane, były w 4/5 zgodne.

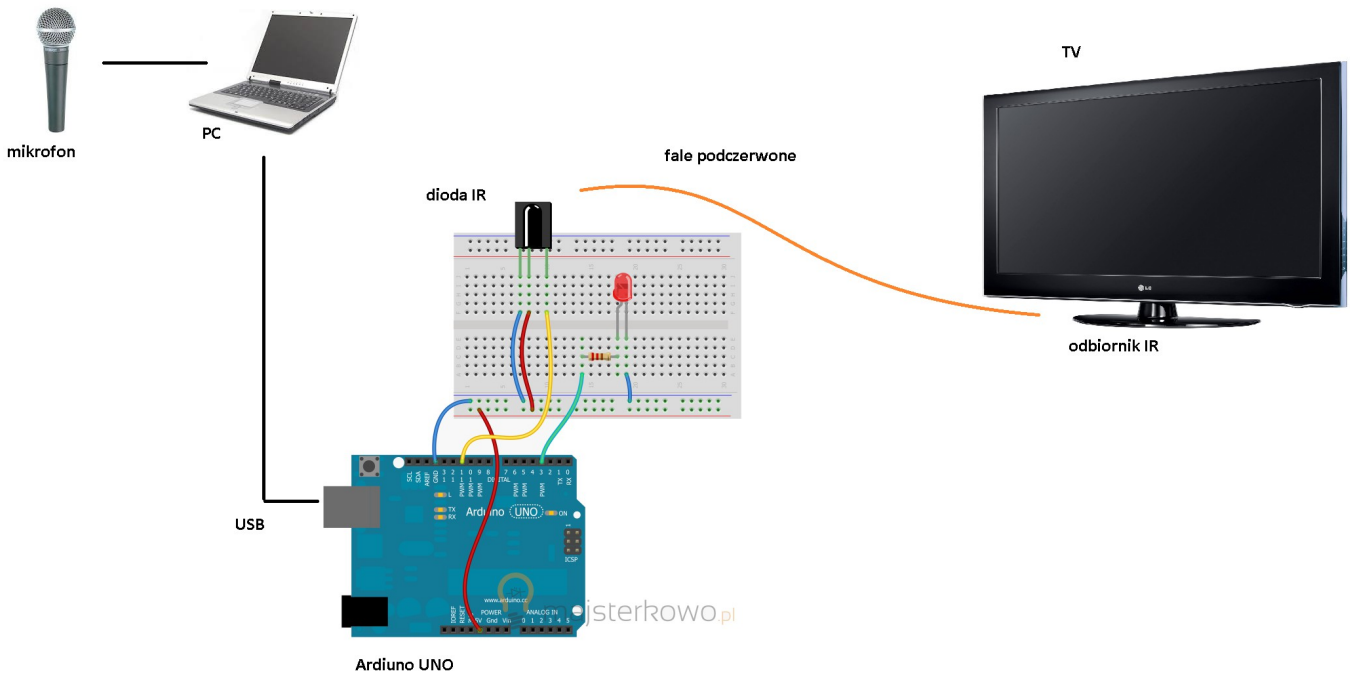
Taka skuteczność została uzyskana dzięki kilku zabiegom. Przede wszystkim wszystkie próbki dźwięku były nagrywane przez jedną osobę. W ten sposób nie pojawiły się kłopoty wynikające z różnic w wymowie, lub akcencie. Drugim ważną decyzja było przygotowanie sygnału przed analizą. Wzmocnienie częstotliwości wysokich, mniejsze próbkowanie sygnału, oraz normalizacja sygnału. Dzięki temu sygnał był wyrazisty i mógł być powtarzalny.

Wysoka skuteczność jest niewątpliwie również zasługą mechanizmu klasyfikacji, czyli połączenia trzech metod, oraz analizy capstralnej. Algorytmy te dają wyśmienite wyniki, oraz są proste do implementacji.

Podczas tworzenia projektu podejmowaliśmy również próby implementacji innych algorytmów niż przez nas użyte, jednak okazały się ona rozwiązaniami gorszymi od zaprezentowanych. Gorsze wyniki dawało liczenie odległości między sąsiadami przy pomocy miary Hamminga.

Ważnym błędem który popełniliśmy podczas tworzenia projektu było liczenie wektora cech z całej próbki. W efekcie otrzymywaliśmy uśredniony wektor z całej próbki. Wyniki były wtedy bardzo niskie (rzędu 20%). Lecz po obliczaniu całych macierzy z poszczególnych fragmentów dźwięku wyniki osiągnęły wartości jak na powyższym wykresie.

Napisany program działa i zwraca bardzo dobre wyniki. Dlatego liczymy, że w przyszłości uda się zaimplementować na samodzielne urządzenie. To oprogramowanie umożliwiłoby sterowanie telewizorem za pośrednictwem komend głosowych.



Ilustracja 8: Schemat działania systemu sterującego telewizorem przy pomocy komend głosowych

6. Literatura

[1] L. Muda, M.Begam, I. Elamvazuthi „*Voice Recognition Algorithms using Mel Frequencyw Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques*", Journal of computing, volume 2, issue 3, march 2010, ISSN 2151-9617

[2] P. Walendowski „*Zastosowanie sieci neuronowych typu SVM do rozpoznawania mowy*„ , Politechnika Wrocławska, Wrocław 2008

[3] http://www.staff.amu.edu.pl/~drizzt/images/DSSU/W8.pdf

[4] S. Kacprzak „*Inteligentne metody rozpoznawania dźwięku*", Politechnika Łódzka, Łódź 2010