



**Państwowa Wyższa Szkoła Zawodowa
w Tarnowie**

Instytut Politechniczny

Kierunek: Informatyka
Specjalność: Informatyka stosowana
2012/2013

Piotr Kozłowski

PRACA INŻYNIERSKA

Sterowanie głosowe robotem Kawasaki
w wybranych grach planszowych

Opiekun pracy:
dr inż. Robert Wielgat

Tarnów 2013

Oświadczam, że niniejszą pracę wykonałem samodzielnie,
oraz że praca nie narusza praw autorskich innych osób.

.....
Podpis

Oświadczam, że wersja elektroniczna pracy znajdująca się na dołączonej płycie jest zgodna z niniejszym wydrukiem.

.....
Podpis

Składam serdeczne podziękowania Panu dr inż. Robertowi Wielgatowi, za poświęcony czas oraz cenne wskazówki, które pomogły mi ukończyć niniejszą pracę.

Spis treści

1. Wstęp	6
1.1. Cel.....	6
1.2. Zawartość	7
2. Systemy rozpoznawania mowy	8
2.1. Podział zagadnień	8
2.2. Zastosowanie.....	9
2.3. Skuteczność	9
2.4. Przegląd gotowych rozwiązań	10
2.5. Schemat działania systemu	11
2.6. Podział metod klasyfikacji.....	12
3. Sygnał mowy	13
3.1. Mowa w języku polskim.....	13
3.2. Reprezentacja.....	14
3.3. Detekcja	16
3.4. Przetwarzanie wstępne.....	17
3.5. Ekstrakcja cech	17
3.5.1. MFCC.....	18
4. Ukryte Modele Markowa jako jedna z metod klasyfikacji sygnału mowy	22
4.1. Wprowadzenie teoretyczne.....	22
4.2. Parametry modeli Markowa.....	23
4.3. Trenowanie	25
4.3.1. Przebieg.....	25
4.3.2. Algorytm Bauma-Welcha.....	28
4.4. Rozpoznawanie	32
4.4.1. Przebieg.....	32
4.4.2. Algorytm Viteriego	34
5. Wykorzystane narzędzia	36
5.1. HTK (Hidden Markov Model Toolkit)	36

5.1.1. Podstawy HTK	36
5.1.2. Architektura	38
5.1.3. Parametry narzędzi HTK.....	39
5.2. Raspberry Pi.....	40
5.2.1. Parametry	41
5.3. Robot przemysłowy Kawasaki	42
5.4. Stockfish	43
6. System sterujący robotem za pomocą komend głosowych	45
6.1. Schemat działania aplikacji	45
6.2. System mikroprocesorowy	47
6.3. Szczegóły użycia HTK	47
6.3.1. Przygotowanie danych	48
6.3.2. Trenowanie	53
6.3.3. Testowanie.....	54
6.3.4. Analiza	55
6.3.5. Etapy rozgrywki.....	56
6.4. Robot	61
6.5. Gry planszowe	63
6.5.1. Szachy	64
6.5.2. Wilk i owce	64
7. Podsumowanie	66
7.1. Jakość systemu.....	66
7.2. Możliwości rozwoju	67
Bibliografia	69
 Dodatki	 76
A. Instrukcja wykorzystania skryptów do zautomatyzowania pracy z HTK	76
A.1. Wstęp	76
A.2. Trenowanie oraz rozpoznawanie.....	78
B. Zawartość płyty	86

1. Wstęp

Tematyka niniejszej pracy jest związana z komunikacją werbalną na lini człowiek - maszyna. Jeszcze do niedawna sterowanie sprzętem elektronicznym za pomocą komend głosowych, było spotykane częściej w filmach s-f niż w rzeczywistych zastosowaniach. Obecnie jest to zadanie możliwe do wykonania, niemniej jednak ze względu na skomplikowany charakter sygnału mowy bywa często trudne do realizacji.

Systemy rozpoznawania mowy są rozwijane od wielu lat zarówno jako projekty komercyjne, jak i naukowe. Obecnie większość telefonów komórkowych, tzw. „smartfonów” posiada aplikacje ułatwiające korzystanie z nich, pozwalające na głosowe wybieranie połączeń lub zamieniające mowę na tekst. Liczba zastosowań systemów rozpoznawania mowy ciągle rośnie, ze względu na oczywisty fakt, że mowa jest naturalnym oraz wygodnym sposobem przekazywania myśli i odczuć człowieka. Uzyskiwana jest również coraz większa skuteczność takich systemów, jednakże nie tak duża, by pozwalała na bezpieczne i efektywne wykorzystywanie ich w każdym przypadku. W celu poprawienia skuteczności wciąż trzeba szukać nowych rozwiązań.

Przeglądając artykuły i prace dotyczące tematyki tutaj poruszanej często spotkać się można ze stwierdzeniem, że problem rozpoznawania mowy dla języka polskiego nie został dotychczas tak dobrze opracowany, jak to zostało zrobione w przypadku innych języków, tzw. języków dominujących, takich jak angielski, niemiecki, francuski, hiszpański czy chiński. Ta granica powoli zaczyna się zacierać, coraz więcej rozwiązań tworzonych jest również z myślą o języku polskim.

1.1. Cel

Niniejsza praca koncentruje się na wykorzystaniu osiągnięć projektu HTK oraz stworzeniu systemu rozpoznawania mowy dla języka polskiego na przykładzie sterowania robotem przemysłowym w grach planszowych. Rozgrywka polega na wydawaniu komend głosowych, które sterują ruchami robota. Jest on zarówno przeciwnikiem użytkownika, jak również wykonuje za niego wszystkie ruchy podczas rozgrywki, użytkownik skupia się zatem tylko na wydawaniu poleceń głosowych. Gra z robotem jest formą demonstracji systemu stworzonego na potrzeby niniejszej pracy, a więc jest jej poświęcone mniej uwagi niż tematyce rozpoznawania mowy.

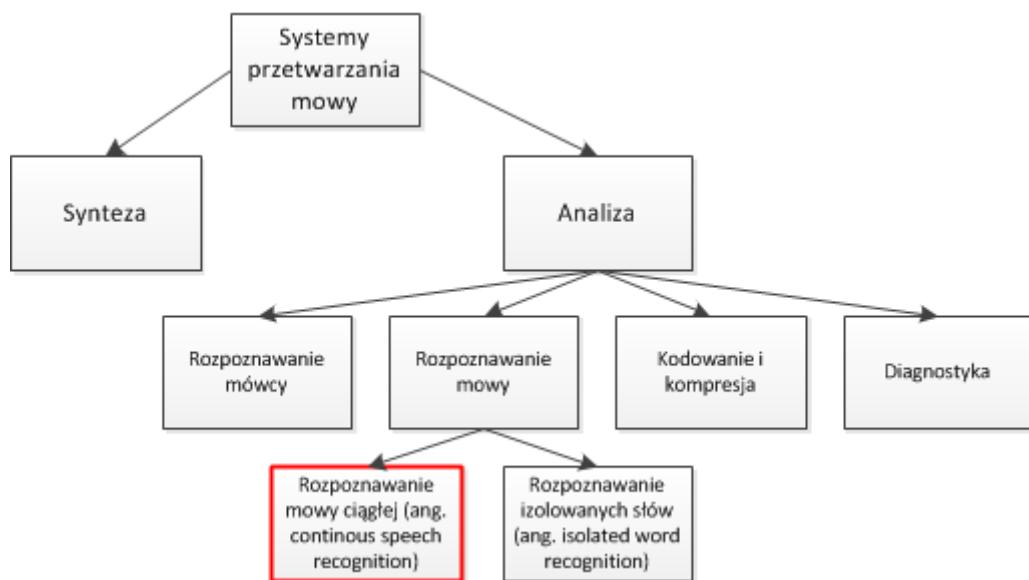
1.2. Zawartość

We „Wstępie” (rozdział 1) przybliżono tematykę oraz cel niniejszej pracy. W rozdziale drugim przedstawione zostały podstawy systemów rozpoznawania mowy (m.in. ich zastosowanie, parametry wpływające na skuteczność działania). Rozdział trzeci opisuje sygnał mowy, m.in. ekstrakcję cech, która jest bardzo ważnym punktem tworzenia systemów rozpoznawania mowy. W rozdziale czwartym przedstawiony został ogólny zarys metody Ukrytych Modeli Markowa, będącej podstawą działania systemu stworzonego na potrzeby niniejszej pracy. W kolejnym rozdziale, zostały opisane narzędzia wykorzystane przy tworzeniu niniejszej pracy. Następnie w rozdziale szóstym zaprezentowany został efekt końcowy czyli aplikacja sterująca robotem. Ostatni rozdział podsumowuje niniejszą pracę oraz zawiera wnioski autora na temat zaprezentowanych wcześniej zagadnień i wyników zaimplementowanego systemu.

2. Systemy rozpoznawania mowy

2.1. Podział zagadnień

Na początek warto określić miejsce systemów rozpoznawania mowy na tle innych zagadnień z tej dziedziny (rys. 2.1). Na czerwono zaznaczono miejsce w którym znajduje się system stworzony na potrzeby niniejszej pracy.



Rysunek 2.1: Ogólny podział systemów przetwarzania mowy

Systemy automatycznego rozpoznawania mowy (ASR, ang. Automatic Speech Recognition) cechują się tym, że działają w czasie rzeczywistym (ang. real time). System ten ma za zadanie przeprowadzić detekcję sygnału mowy (wykryć kiedy została wypowiedziana jakaś sentencja) następnie, jako wynik rozpoznawania utworzyć jej transkrypcję fonetyczną (zamienić mowę na tekst) lub wykonać określone zadanie (np. przedstawienie wybranej figury szachowej). Wszystko to powinno zostać zrobione automatycznie, tzn. bez ingerencji użytkownika. Taka sama idea przywieca systemowi stworzonemu na potrzeby niniejszej pracy, który działa dla mowy ciągłej.

2.2. Zastosowanie

Mowa jest najbardziej naturalnym oraz skutecznym sposobem porozumiewania się ludzi. Jest znacznie szybsza niż pisanie czy gestykulacja, a co najważniejsze nie wymaga użycia rąk. Nie dziwi zatem fakt, że wraz z rozwojem sprzętu komputerowego, wzrostem jego mocy obliczeniowej oraz miniaturyzacji, systemy rozpoznawania mowy mają coraz częstsze zastosowanie. Wśród przykładowych zastosowań takich systemów można wymienić:

- sterowanie głosem urządzeń elektronicznych i elektromechanicznych,
- teleinformatyczne systemy informacyjne,
- systemy STT (ang. Speech To Text),
- urządzenia dla osób niepełnosprawnych. [30]

Warto zaznaczyć, że systemy rozpoznawania mowy, nie znajdą zastosowania, tam gdzie jest wymagana szybka reakcja użytkownika (np. sytuacja prowadzenia pojazdu) lub istnieją alternatywne, prostsze sposoby komunikacji na lini człowieek - maszyna (np. za pomocą wygodnego i ergonomicznego panelu operatora).

2.3. Skuteczność

Na skuteczność oraz szybkość rozpoznawania wpływa wiele parametrów. Do najważniejszych z nich należą:

- stopień zależności od mówcy (systemy zależne od mówcy (ang. speaker-dependent) oraz niezależne od mówcy (ang. speaker-independent)),
- rozmiar słownika, w którym zdefiniowane są słowa rozpoznawane przez system,
- podobieństwo akustyczne i fonetyczne słów,
- stosunek sygnału do szumu (SNR, ang. signal-to-noise ratio),
- parametry transmisji sygnału mowy. [30]

System stworzony na potrzeby niniejszej pracy jest zależny od mówcy, oznacza to, że największą skuteczność uzyskuje dla osoby, której próbki głosu zostały wykorzystane do jego wytrenowania. Systemy niezależne od mówcy z założenia mają działać dla różnych osób, wymaga to jednak większego oraz bardziej zróżnicowanego zbioru treningowego. Rozmiar słownika ma wpływ na skuteczność rozpoznawania jak również na szybkość działania systemu. Dla większych słowników rozpoznawanie trwa trochę dłużej, a skuteczność zmniejsza się ze względu na większe prawdopodobieństwo wystąpienia słów podobnych fonetycznie lub akustycznie. Przy dobórze słownika chcąc osiągnąć zadowalające wyniki warto wybierać słowa dłuższe oraz jak najbardziej różniące się od siebie w wymowie. Krótkie jednostki (słowa) trudniej

rozpoznawać niż długie wypowiedzi. Jako parametry transmisji sygnału mowy mogące mieć wpływ na skuteczność rozpoznawania warto wymienić m.in. zniekształcenia analogowe sygnału mowy wnoszone przez kanał transmisyjny, liczbę poziomów kwantyzacji oraz częstotliwość próbkowania. Skuteczność rozpoznawania zależy również od środowiska w jakim działa dany system. Może być ona inna dla zamkniętego pomieszczenia oraz otwartej przestrzeni. Negatywny wpływ mają tutaj wszelkiego rodzaju szумy towarzyszące pracy systemu rozpoznawania mowy (np. szum komputera, ruch uliczny, odgłosy rozmów innych ludzi). Nie bez wpływu pozostają również parametry sprzętu użytego do nagrania próbek głosu (np. redukcja szumu, czułość mikrofonu) jak również cechy samego mówcy takie jak sposób wymowy (gwara, akcent), szybkość oraz głośność. Jeżeli czułość mikrofonu podczas nagrywania zostanie ustawiona na niską, wymaga to głośniejszych wypowiedzi w celu poprawnego zarejestrowania sygnału mowy, lecz w zamian za to, przenosi mniej szumów otoczenia, które mogą być obecne w czasie pracy systemu.

2.4. Przegląd gotowych rozwiązań

Wśród rozwiązań komercyjnych obecnie przodujących na rynku warto wymienić te przeznaczone dla urządzeń mobilnych, czyli m.in. telefonów komórkowych, które w obecnych czasach często posiadają możliwości zbliżone do komputerów osobistych. Firmy, których produkty zdobiły rynek systemów przeznaczonych na urządzenia mobilne mają opracowane własne technologie rozpoznawania mowy połączone ze sztuczną inteligencją oraz syntezą mowy. I tak kolejno, Apple posiada Siri [19], Google - Google Now [21] a Microsoft - TellMe [26]. Wymienione produkty mają za zadanie ułatwić korzystanie z telefonu. Pełnią rolę wirtualnego asystenta, z którym można porozmawiać, zapytać o pogodę, najbliższą restaurację, poprosić o wpis do kalendarza lub przeczytanie wiadomości SMS. Produkt firmy Apple był najwcześniej wprowadzony na rynek, więc wydaje się być technologią najbardziej zaawansowaną. Natomiast Google prócz nowej technologii Google Now, posiada również aplikację Google Mobile App [20], która służy m.in. do wyszukiwania głosowego (np. kontaktów, wiadomości lub innych informacji korzystając z wyszukiwarki internetowej Google). Poza produktami wymienionymi wcześniej firm, powstaje coraz więcej aplikacji typu third-party (czyli tworzonych przez osoby lub firmy niezwiązane z producentami systemów operacyjnych) posiadających podobne funkcjonalności (np. S-Voice firmy Samsung lub Iris, obie przeznaczone na platformę Android firmy Google).

Do najstarszych projektów komercyjnych korzystających z rozpoznawania mowy można zaliczyć IBM ViaVoice, Microsoft Speech API, Oracle Java Speech API oraz Dragon Naturally Speaking [27] firmy Nuance, która ostatecznie przejęła od IBM technologię ViaVoice. Wymienione wyżej technologie (poza aplikacją Google Mobile App) nie są dostępne dla języka polskiego. Rozwiązania przeznaczone na urządzenia mobilne są uzupełniane o nowe języki, zatem w najbliższej przyszłości będą one dostępne również dla języka polskiego.

W Polsce powstaje coraz więcej firm projektujących i wdrażających rozwiązania biznesowe oparte na rozpoznawaniu mowy, zwane systemami IVR (ang. Interactive Voice Response), które umożliwiają interaktywną obsługę osoby dzwoniącej. Są to m.in. firma Primespeech (produkująca systemy przeznaczone na rynek telekomunikacyjny, np. dla Zarządu Transportu Miejskiego w Warszawie, działające na zasa-

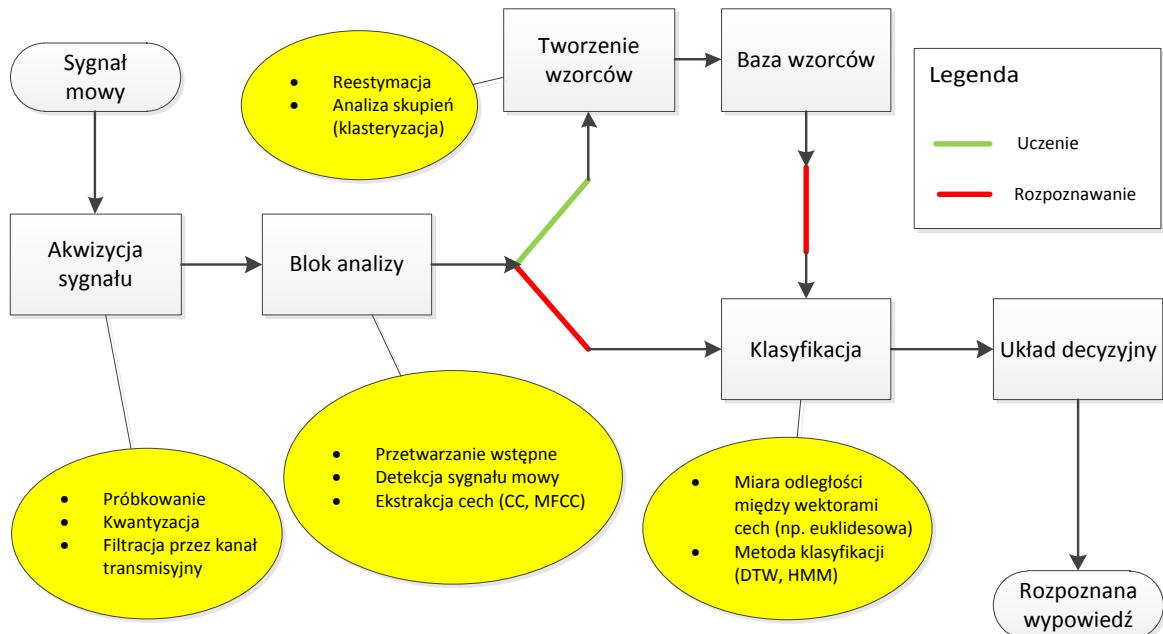
dzie wirtualnych konsultantów, informujących o rozkładach jazdy, cenach biletów, aktualnościach lub przyjmujących różnego rodzaju zgłoszenia klientów) [10], MagicScribe (systemy zamieniające mowę na tekst, MagicScribeMedical - system stworzony dla medycyny, wspomagający obsługę pacjenta, tworzenie dokumentacji medycznej itp. oraz MagicScribeLegal, rozwijanie dla adwokatów, notariuszy, radców prawnych) [9]. Spośród innych polskich firm zajmujących się tematyką rozpoznawania mowy należałoby wspomnieć o Stanusch Technologies S.A., VOICE LAB oraz SkryBot.pl.

Jeśli chodzi zaś o projekty naukowe związane z rozpoznawaniem mowy, to warto wymienić tutaj m.in. HTK [6], Julius [7], CMUSphinx [3] lub rodzimy projekt SARMATA (używany m.in. przez instytucje wymiaru sprawiedliwości do zarządzania dokumentacją procesową) [12].

W obecnych czasach tworzenie systemów rozpoznawania mowy jest dużo prostsze, ze względu na obecność gotowych narzędzi wspomagających, m.in. HTK czy CMUSphinx, które posiadają zaimplementowane algorytmy, pozwalające stworzyć system rozpoznawania mowy nawet osobie nie znającej szczegółów aparatu matematycznego. Użytkownik korzystający z takich bibliotek często używa ich jak „czarnej skrzynki”, niemniej jednak potrafi stworzyć w pełni działający system. Na tyle skuteczny, aby nie musieć zagłębiać szczegółów zaimplementowanych algorytmów. Co prawda taka niewiedza stwarza problemy przy dobiorze optymalnych parametrów mających istotny wpływ na działanie systemu, lecz nie przeszkadza na dobrą ich w sposób eksperymentalny i osiągnięcie zadowalającej skuteczności.

2.5. Schemat działania systemu

Na rysunku 2.2 jest widoczny ogólny schemat działania systemu rozpoznawania mowy.



Rysunek 2.2: Schemat blokowy systemu rozpoznawania mowy

Proces tworzenia systemu rozpoznawania mowy można podzielić na dwie fazy, uczenie (trenowanie) oraz rozpoznawanie. Trenowanie ma na celu przygotowanie systemu do rozpoznawania i musi zostać wykonane w pierwszej kolejności. Podczas tej fazy za pomocą wypowiedzi uczących są tworzone i ulepszane wzorce, z których system korzysta w trakcie rozpoznawania. W fazie rozpoznawania system dopasowuje najbardziej prawdopodobny wzorzec, czego wynikiem jest etykieta nieznanej wypowiedzi. Szczegółły wymienionych wyżej faz opisane są w kolejnych rozdziałach.

2.6. Podział metod klasyfikacji

Współczesne systemy rozpoznawania mowy zakładają, że sygnał mowy jest sekwencją pewnych elementarnych jednostek fonetycznych (np. głosek, fonemów lub słów). Charakteryzuje się on również pewną przypadkowością. Wypowiedź tego samego mówcy za każdym razem posiada niepowtarzalny przebieg czasowy, co może stwarzać trudności w automatycznym rozpoznawaniu wypowiedzi. Niepowtarzalność przebiegu czasowego wiąże się m.in. z tym, że nagrany sygnał mowy często bywa zaszułmiony i zniekształcony.

Metody klasyfikacji wykorzystywane w systemach rozpoznawania mowy można podzielić na dwie kategorie:

- metody deterministyczne, w których są obliczane błędy porównania dźwięku ze wzorcem,
- metody niedeterministyczne, w których obliczane są wartości prawdopodobieństw, reprezentujące dopasowanie dźwięku do stosowanych modeli probabilistycznych (wzorców). [22]

Do metod deterministycznych należy m.in. nieliniowa transformata czasowa (DTW, ang. Dynamic Time Wrapping), metoda skuteczna dla izolowanych słów.

Ograniczenia metod deterministycznych, to również m.in. duża zajętość pamięci potrzebnej na przechowywanie wzorców (dla dużych słowników) oraz to, że czas rozpoznawania jest proporcjonalny do rozmiaru słownika. [30]

Biorąc pod uwagę fakt, że wypowiadanie słów ciągiem (bez znaczących przerw między nimi) jest dla człowieka bardziej naturalne oraz, że przy analizie sygnału mowy warto uwzględnić jego przypadkowość, metody niedeterministyczne osiągają lepszą skuteczność rozpoznawania. Do takich właśnie metod należą Ukryte Modele Markowa (HMM, ang. Hidden Markov Models).

3. Sygnał mowy

3.1. Mowa w języku polskim

Alfabet języka polskiego składa się z 32 liter. Łacińskie litery q, v i x występują jedynie w pisowni wyrazów obcych. Alfabet fonetyczny języka polskiego składa się z około 78 dźwięków. Dokładna liczba głosek zależy od sposobu traktowania wariantów. Relacja litera – głoska jest typu wiele – wiele. Tak jest również w wielu innych językach. Strukturę języka można przedstawić w postaci modelu warstwowego (rys. 3.1). Elementy warstwy wyższej zawierają pewną liczbę elementów warstwy niższej np. sylaby składają się z fonemów.

myśli
wypowiedzi
zdania
słowa
sylaby
głoski/fonemy

Rysunek 3.1: Warstwowy model języka [23]

Badaniem struktury dźwiękowej języka zajmują się dwa pokrewne działy, fonetyka oraz fonologia. Fonetyka jest nauką o głoskach, czyli dźwiękach mowy. „Bada i opisuje dźwięki mowy ze względu na ich właściwości fizyczne, tzn. ustala artykulacyjne i akustyczne ich cechy.” [18] Fonologia natomiast bada dźwięki mowy, pod kątem pełnionych przez nich funkcji w procesie komunikacji. Podstawową jednostką fonologii jest fonem. Fonem uznaje się za najmniejszy rozróżnialny segment dźwiękowy mowy, który może odróżniać znaczenie dźwięków mowy danego języka o różnicach wynikających wyłącznie z charakteru indywidualnej wymowy lub kontekstu. Dla systemów rozpoznawania mowy istotne jest to, że każdy fonem jest zespołem cech dystynktywnych pozwalających na odróżnienie go od pozostałych fonemów, co przekłada się bezpośrednio na parametry akustyczne sygnału. [18]

Miarę podobieństwa między sygnałem mowy można podzielić na dwie kategorie, podobieństwo akustyczne oraz fonetyczne.

- Podobieństwo fonetyczne dwóch słów jest ustalane na podstawie tzw. odległości Levenshteina określanej jako minimalny koszt przekształcenia jednego słowa w drugie.

- Podobieństwo akustyczne dwóch słów jest ustalane na podstawie odległości (prawdopodobieństwa identyczności) między dwoma słowami wynikającej z przyjętych cech sygnału mowy w ramce oraz przyjętej metody klasyfikacji.

Zazwyczaj im słowa są bardziej podobne fonetycznie, tym bardziej są podobne akustycznie. Zdarzają się jednak sytuacje, w których zależność taka nie zachodzi. Jest to jedna z przyczyn pojawiania się błędów w rozpoznawaniu mowy. [30]

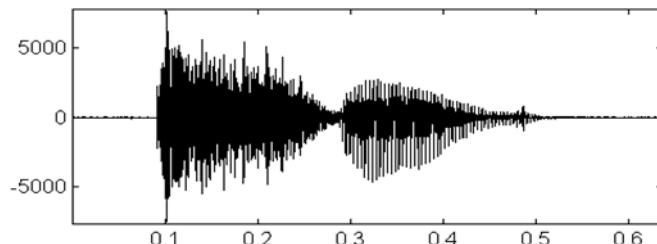
3.2. Reprezentacja

Źródłem sygnału mowy mogą być drgania wiązań głosowych (głoski dźwięczne) jak i szum powstający w skutek przepływu powietrza przez narządy mowy (głoski bezdźwięczne). [25]

Analogowy sygnał mowy nie nadaje się bezpośrednio do analizy komputerowej. Najpierw musi zostać wykonana jego dyskretyzacja, czyli próbkowanie i kwantyzacja.

Typowym formatem zapisu sygnału analogowego w postaci cyfrowej jest WAV (ang. wave form audio format), został on opracowany przez Microsoft oraz IBM w roku 1991. Jest to format bezstratny. Opis struktury pliku WAV można znaleźć w wielu źródłach, m.in. w pozycji [16].

Sygnal mowy w postaci cyfrowej może być opisywany na różne sposoby. Podstawowym i najprostszym jest opis w dziedzinie czasu (rys. 3.2). Ma on jednak skomplikowany przebieg, będący odzwierciedleniem złożonego charakteru artykulacji i często zawierający różne przypadkowe szумy pochodzące z otoczenia.



Rysunek 3.2: Reprezentacja słowa *trzy* w dziedzinie czasu

Matematycznie można go przedstawić jako splot przebiegu czasowego sygnału źródła $u_z(t)$ oraz odpowiedzi impulsowej kanału głosowego $h(t)$, czyli:

$$u(t) = \int_0^{\tau} h(t-\tau)u_z(\tau)d\tau \quad (3.1)$$

Z punktu widzenia rozpoznawania mowy, reprezentacja czasowa sygnału mowy nie przenosi wystarczającej ilości informacji. [24] Potrzeba zatem dokonać dyskretnej transformacji Fouriera (DFT, ang. Discrete Fourier Transform) sygnału w celu umożliwienia jego analizy w dziedzinie częstotliwości. Realizację tego przekształcenia oraz jego odwrotną formę przedstawiają następujące równania [29]:

$$X(k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}kn}, k = 0, 1, 2, \dots, N-1 \quad (3.2)$$

$$x(n) = \sum_{k=0}^{N-1} X(k)e^{j\frac{2\pi}{N}kn}, n = 0, 1, 2, \dots, N-1 \quad (3.3)$$

W praktyce stosuje się tzn. szybką transformację Fouriera (FFT, ang. Fast Fourier Transform) opisaną m.in. w rozdziale 9 pozycji [29].

Wadą DFT dokonywanej na całym sygnale jest brak jawnego informacji o czasie w widmie sygnału, która jest bardzo istotna w analizie sygnału mowy, ponieważ jest on ciągiem pewnych zdarzeń (zmian częstotliwości, amplitudy oraz następstw fonemów i słów), których kolejność jest bardzo istotna. Dlatego stosuje się tzn. krótkookresową transformację Fouriera (STFT, ang. Short-Time Fourier Transform) zwaną też okienkową transformacją Fouriera (ang. Windowed Fourier Transform). Polega ona na dokonywaniu transformacji krótkich fragmentów sygnału wyznaczonych za pomocą okna $w(n)$.

Szerokość okna determinuje rozdzielcość częstotliwościową i czasową otrzymanego widma czasowo-częstotliwościowego. Najprostszą funkcją okna jest okno prostokątne [29]:

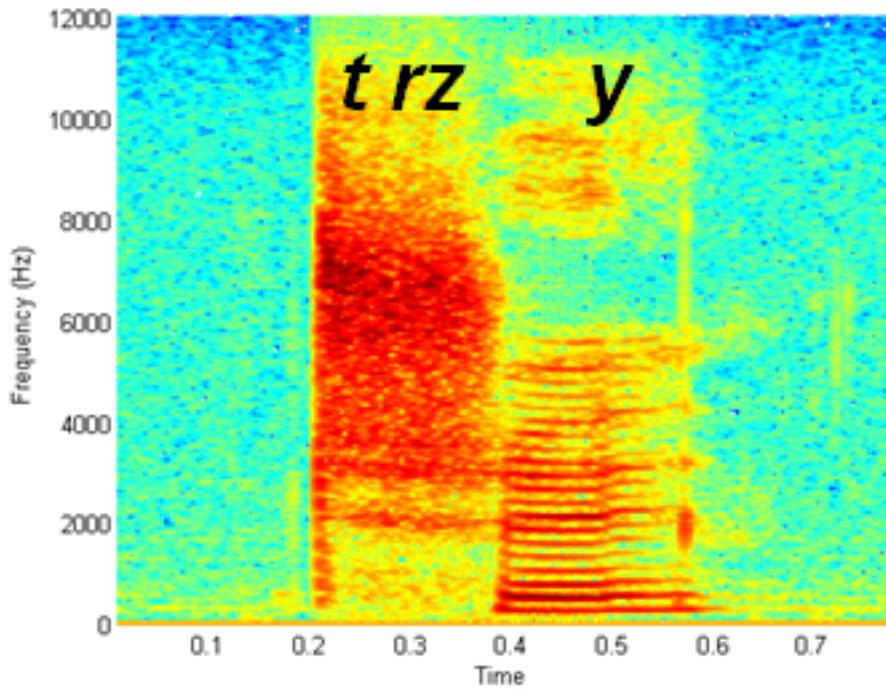
$$w(n) = 1, 0 \leq n \leq N-1 \quad (3.4)$$

Wprowadza ono jednak znaczne zafalowania i „rozmycie” widma analizowanego sygnału, które są związane z efektami brzegowymi wyciętego fragmentu. To zjawisko można minimalizować stosując inny typ okna, np. okno Hamminga, które ma węższe widmo [29]:

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right), 0 \leq n \leq N-1 \quad (3.5)$$

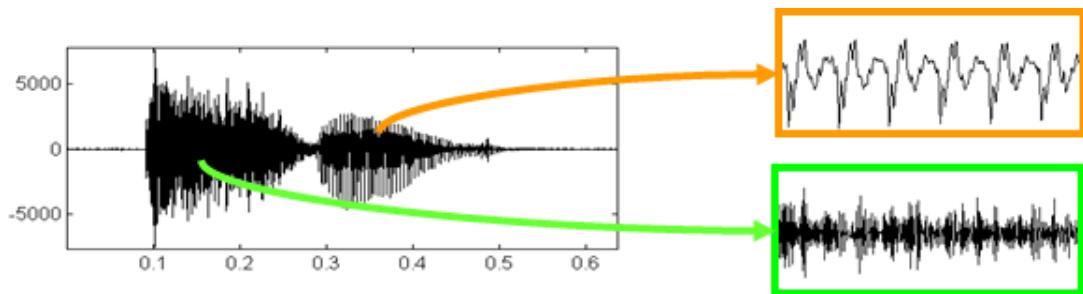
Bardziej szczegółowy opis okna Hamminga oraz innych typów okien można znaleźć m.in. w rozdziale 8 pozycji [29].

Po przeprowadzeniu transformacji Fouriera sygnał mowy może zostać przedstawiony na spektrogramie (rys. 3.3).



Rysunek 3.3: Reprezentacja słowa *trzy* w dziedzinie częstotliwości [30]

Patrząc na spektrogram (rys. 3.3) można zauważać różnice pomiędzy głoskami bezdźwięcznymi a głoską dźwięczną „y”. W przypadku głosek dźwięcznych można wyróżnić pewną okresowość, natomiast głoski bezdźwięczne często mieszają się z szumem (rys. 3.4). Co za tym idzie, słowa różniące się tylko częścią bezdźwięczną trudniej rozpoznawać.



Rysunek 3.4: Segment bezdźwięczny oraz dźwięczny słowa „trzy” [30]

3.3. Detekcja

Detekcję sygnału mowy wykonuje się w celu podniesienia skuteczności rozpoznawania oraz skrócenia czasu obliczeń. Dokonuje się jej po uprzednim podziale sygnału mowy na ramki. Można ją przeprowadzić w oparciu o kilka parametrów m.in.:

- moc sygnału w ramce,
- liczba przejść przez zero w ramce,

- entropia widma.

W każdej ramce obliczany jest wybrany parametr a następnie na podstawie jego wartości podejmowana jest decyzja o zaakceptowaniu ramki jako sygnału mowy lub jej odrzuceniu. Można też przyjąć zasadę, że po wstępnej detekcji sygnału mowy odrzuca się fragmenty sygnału krótsze niż 100 ms, chyba że leżą w odległości czasowej mniejszej niż 100 ms od innych fragmentów sygnału. [30]

W niniejszej pracy detekcja mowy przeprowadzana jest według następujących zasad. Za sygnał mowy zostaje uznany dźwięk trwający dłużej niż 100 ms o natężeniu większym niż 1% całości.

3.4. Przetwarzanie wstępne

W każdym systemie analizy mowy można wyróżnić początkowe etapy analizy tzn. przetwarzanie wstępne do którego mogą należeć następujące kroki:

- preemfaza, stosuje się ją w celu wzmacniania wyższych częstotliwości w sygnale mowy osłabionych na skutek filtracji przez kanał transmisyjny,
- odszumianie sygnału mowy (np. metoda odejmowania widmowego (ang. spectral subtraction), filtr Wienera, metody perceptualne),
- inne rodzaje filtracji. [30]

W widmie sygnału mowy więcej energii znajduje się w niskich częstotliwościach, to niekorzystne z punktu widzenia przetwarzania sygnałów zjawisko nazywane jest przekrzywieniem widma (ang. spectral tilt). Aby usunąć skutki przekrzywienia stosuje się w/w preemfazę według następującego wzoru:

$$y(n) = x(n) - a * x(n-1) \quad (3.6)$$

gdzie a to współczynnik preemfazy, $0.9 \leq a \leq 1$. [33]

3.5. Ekstrakcja cech

Najczęściej stosowane cechy sygnału mowy to:

- współczynniki LPC (ang. Linear Prediction Coefficients),
- parametry mel-cepstralne (MFCC, ang. Mel-frequency cepstral coefficients),
- parametry dyskretnej transformacji falkowej (DWT, ang. Discrete Wavelet Transform).

Każdy zestaw cech może zostać uzupełniony o tzn. cechy dynamiczne, czyli współczynniki różnicowe. Polega to na obliczeniu pierwszej i drugiej pochodnej powyższych współczynników (tzw. współczynników delta oraz delta-delta) względem kilku ramek. [30]

Pakiet HTK może korzystać zarówno z cech mel-cepstralnych jak i LPC. Opisane zostały jedynie cechy mel-cepstralne, ponieważ zostały one wykorzystane w systemie stworzonym na potrzeby niniejszej pracy.

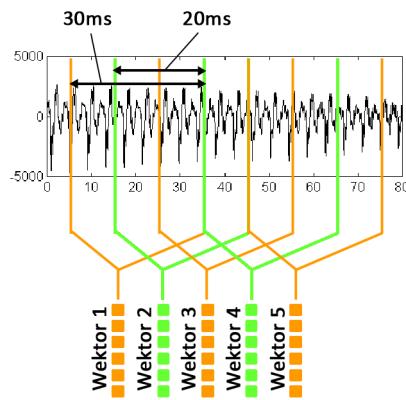
Przed rozpoczęciem wyznaczania współczynników MFCC sygnał mowy poddaje się zazwyczaj pre-emfazie.

3.5.1. MFCC

Ekstrakcję cech sygnału mowy za pomocą współczynników MFCC można ogólnie przedstawić w następujących krokach:

- blokowanie sygnału w ramki, okienkowanie oknem Hamminga,
- przeprowadzenie FFT na zokienkowanych ramkach sygnału,
- filtracja za pomocą melowego banku filtrów,
- obliczenie mocy FFT w określonych pasmach częstotliwościowych,
- obliczenie logarytmu zakumulowanych współczynników widmowych,
- przeprowadzenie dyskretnej transformata kosinusowej (DCT, ang. Discrete Cosine Transform) na zlogarytmowanych współczynnikach widmowych,
- opcjonalnie, wyznaczenie cech dynamicznych. [30]

Pierwszym etapem tej metody jest podział sygnału na ramki. Przykład pokazano na rysunku 3.5, wektory 6 parametrów ekstrahowanych w ramkach o czasie trwania 30 ms, zachodzących na siebie na odcinkach 20 ms.



Rysunek 3.5: Ekstrakcja cech na podstawie [30]

Następnie ramki wymnaża się z oknem Hamminga i oblicza FFT.

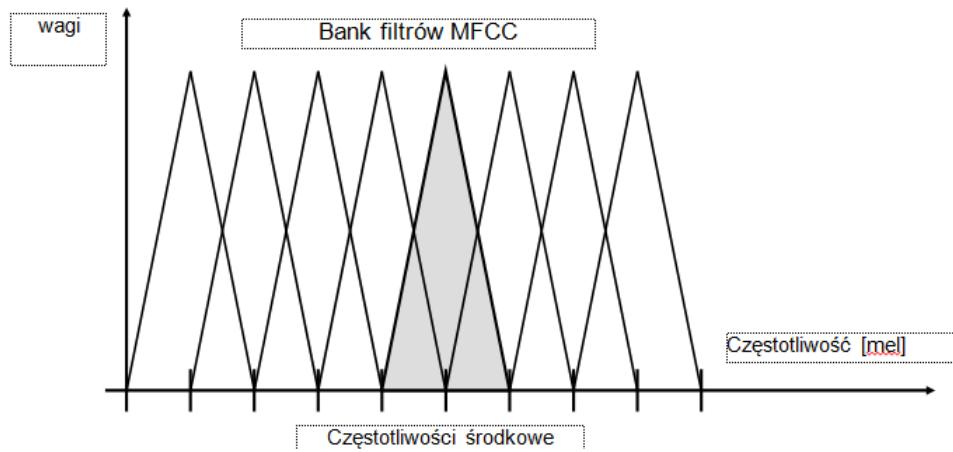
Widma ramek sygnału po uprzednim przeprowadzeniu FFT poddawane są filtracji za pomocą melowego banku filtrów pasmowo przepustowych. W systemach rozpoznawania mowy zazwyczaj stosuje się

banki 26 filtrów melowych. Filtry tworzone są dla kolejnych pasm częstotliwości, rozmieszczonych w nieliniowy sposób wyznaczony przez skalę Mel. Zdefiniowane są w dziedzinie częstotliwości, co umożliwia łatwe wymnożenie ich przez przekształcony sygnał. Filtracja za pomocą banku filtrów polega na podniesieniu wartości prążków widma do kwadratu, czyli wyznaczeniu estymaty funkcji gęstości widmowej mocy sygnału a następnie uśrednieniu grupy prążków widma za pomocą nakładających się funkcji wagowych o kształcie trójkątnym. Funkcje wagowe są dla wyższych częstotliwości coraz szersze. [29] Jest to spowodowane tym, że ucho człowieka reaguje nieliniowo na zmieniającą się częstotliwość dźwięku. Częstotliwości powyżej 1 kHz są słabiej odczuwane niż różnice w zakresie niskich częstotliwości. Dlatego im wyższa częstotliwość tym są potrzebne coraz większe odstępy między kolejnymi pasmami dla zrekompensowania nieliniowości. W tym celu zastosowano skalę melową (Mel) zamiast hercowej (Hz). [33, 29]

$$m = 2595 \cdot \log\left(1 + \frac{\omega}{700\text{Hz}}\right) \quad (3.7)$$

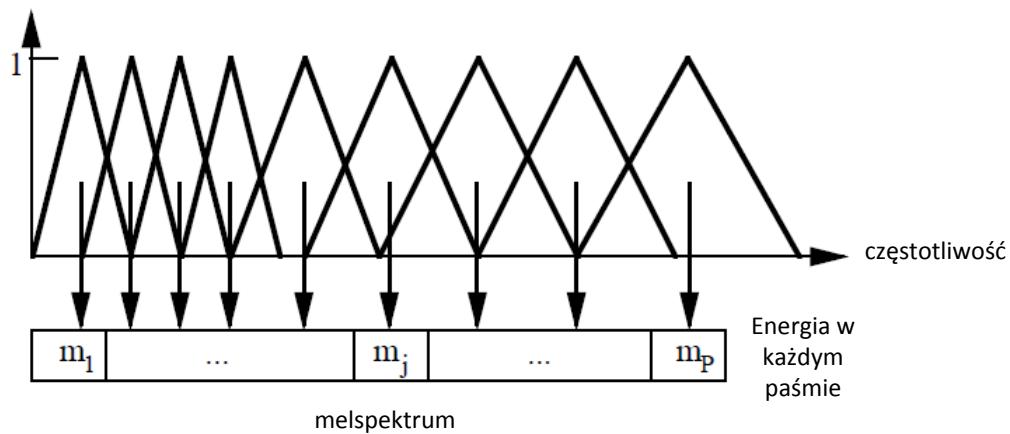
$$f = 700 \cdot (10^{m/2595} - 1) \quad (3.8)$$

W skali melowej (rysunek 3.6) prototypy wag są identycznymi trójkątami, które nakładają się na siebie w połowie szerokości podstawy.



Rysunek 3.6: Bank filtrów w skali melowej na podstawie [30]

Po transformowaniu ich spowrotem do skali hercowej (wzór 3.8) otrzymuje się trójkątne funkcje wagowe (rysunek 3.7), które stosuje się do uśredniania wyznaczonej wcześniej estymaty widma mocy.



Rysunek 3.7: Bank filtrów w skali hercowej na podstawie [32]

Natężenie dźwięku jest odczuwane przez ludzi w skali logarytmicznej, dlatego przy obliczaniu cepstrum sygnał otrzymyany po przejściu przez bank filtrów melowych jest logarytmowany.

Następnie oblicza się transformację kosinusową DCT, może ona zostać przeprowadzona według następującego wzoru:

$$c_i = \sqrt{\frac{2}{N}} \sum_{j=1}^N m_j \cos\left(\frac{\pi i}{N}(j - 0.5)\right). \quad (3.9)$$

Gdzie:

m_j - to logarytm amplitudy funkcji wagowej,

N - oznacza liczbę zastosowanych funkcji wagowych (filtrów),

i - liczbę wyznaczanych współczynników mel-cestralnych. [32]

Aby usunąć szkodliwy wpływ podstawowych drgań krtaniowych na zestaw cech można zastosować przekształcenie zwane liftowaniem (ang. liftering).

$$c_i^{lift} = (1 + \frac{\lambda}{2} \sin(\frac{\pi n}{\lambda})) c_i \text{ dla } i = 1, \dots, \lambda, \quad (3.10)$$

gdzie c_n to i -ty współczynnik MFCC a stała λ odpowiada indeksowi cechy związanej z częstotliwością podstawową.

Podstawowy wektor cech można uzupełnić o tzn. cechę „energetyczną”, która pomaga odróżnić sygnał ciszy od sygnału mowy a także słabe bezdźwięczne spółgłoski od silnych dźwięcznych samogłosek. Sumaryczną energię w ramce sygnału τ oblicza się sumując kwadraty próbek w dziedzinie czasu według następującego wzoru:

$$E^{(\tau)} = \sum_{i=1}^M f_i^2 \quad (3.11)$$

[24, 33]

Otrzymany wektor cech MFCC zawiera liczbę elementów równą liczbie pasm melowych. Do dalszego przetwarzania rozpatruje się zazwyczaj pierwsze 12 współczynników, do których dokłada się energię sygnału w ramce oraz można uwzględnić dynamikę zmian współczynników w czasie (wielkość zmian oraz ich tempo), poszerzając wektor o ich przyrosty kolejno pierwszego i drugiego rzędu (czyli współczynniki delta oraz delta-delta). W wyniku tych operacji otrzymuje się 39-elementowy wektor cech mel-cepstralnych.

Podsumowując w wyniku ekstrakcji cech uzyskuje się wartości parametrów zawierających информацию o treści wypowiedzi i będących niezależnymi od indywidualnych cech głosu mówcy. Parametry te tworzą wektory cech, na podstawie których dokonuje się klasyfikacji sygnału.

4. Ukryte Modele Markowa jako jedna z metod klasyfikacji sygnału mowy

4.1. Wprowadzenie teoretyczne

„Ukryte Modele Markowa stanowią serce większości współczesnych systemów rozpoznawania mowy.” [23] Sprawdzają się również w wielu innych zastosowaniach, głównie w przypadku rozpoznawania sygnałów niedeterministycznych. Można tutaj wymienić takie obszary zastosowań jak:

- rozpoznawanie obrazów,
- modelowanie DNA,
- modelowanie danych ekonomicznych.

Pierwsze prace na temat Ukrytych (niejawnych) Modeli Markowa prowadził Baum wraz ze współpracownikami w latach 60-tych oraz 70-tych. Dotyczyły one metody estymacji parametrów modeli HMM nazwanej algorytmem Bauma-Welcha. [22] Do celów rozpoznawania mowy HMM zostały użyte poraz pierwszy w połowie lat siedemdziesiątych przez pracowników firmy IBM. W obecnych czasach jest to najczęściej wykorzystywane narzędzie klasyfikacji w systemach rozpoznawania mowy. [23] Opisane zostało w wielu pozycjach literaturowych, m.in. w [31], tutaj zostanie przedstawiony jedynie ogólny zarys tej metody.

W HMM wykorzystywane są modele statystyczne posiadające własności Markowa pierwszego rzędu co oznacza, że aktualny stan modelu zależy tylko od stanu poprzedniego. W ukrytym modelu Markowa zakłada się, że stany są niewidoczne dla obserwatora, natomiast wyjście (wartości obserwowane) jest jasne, stąd w nazwie występuje słowo *ukryte*. [33] W przypadku sygnału mowy, wartości obserwowane bądź też obserwacje są to wektory cech wyekstrahowane z pojedynczej ramki sygnału mowy. Ukryty model Markowa opisuje pewien układ (proces stochastyczny), który w danym momencie może znajdować się tylko w jednym ze stanów.

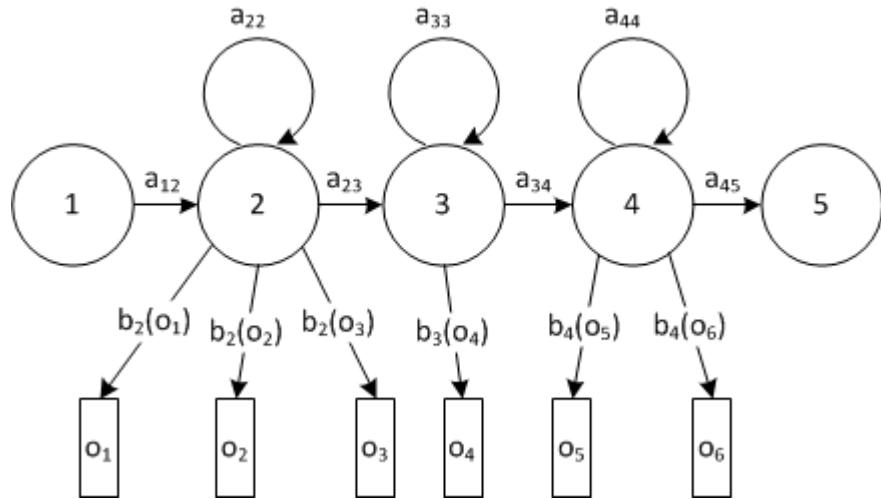
Ogólne założenia metody HMM z punktu widzenia rozpoznawania mowy można przedstawić następująco:

- dla każdej klasy (np. głoski, fonemu lub słowa) tworzy się pewien model stochastyczny $\Lambda_m, m = 1, \dots, M$, gdzie M to liczba klas,
- dla każdej nieznanej sekwencji stanów X oblicza się prawdopodobieństwa generowania sekwencji wektorów $p(O, X | \Lambda_m)$ przez modele dla poszczególnych klas,

- klasyfikacja odbywa się na zasadzie największego prawdopodobieństwa, X jest przypisywane do takiej klasy Λ_m , która posiada największe prawdopodobieństwo $p(O, X | \Lambda_m)$. [30]

4.2. Parametry modeli Markowa

Na rysunku 4.1 widoczny jest 5-cio stanowy model markowa typu left-to-right z przejściami zapisanymi w macierzy 4.1.



Rysunek 4.1: Model Markowa 5-cio stanowy typu left-to-right

W rozpoznawaniu mowy zazwyczaj wykorzystywane są modele typu left-to-right (wyjątek to np. model ciszy) oznacza to, że z aktualnego stanu modelu możliwe jest przejście do stanu kolejnego, pozostanie w aktualnym stanie, lub przejście o kilka stanów do przodu, nie można się cofać.

Stan początkowy oraz końcowy czyli w przypadku 5-cio stanowego modelu stan pierwszy oraz piąty zwane są stanami nieemitującymi, ponieważ nie generują żadnych obserwacji, mają za zadanie jedynie rozpoczęć oraz zakończyć proces.

Macierz przejść definiuje możliwe przejścia pomiędzy stanami modelu oraz wartości ich prawdopodobieństw:

$$\begin{bmatrix} 0 & a_{12} & 0 & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & 0 \\ 0 & 0 & 0 & a_{44} & a_{45} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.1)$$

Dla każdego przejścia z chwili czasowej t do $t + 1$ przejście ze stanu x_i do stanu x_j następuje z prawdopodobieństwem a_{ij} .

Suma prawdopodobieństw w wierszach a więc suma wszystkich wyjść ze stanu zawsze wynosi jeden.

Dla każdej ramki czasowej t , może być przez każdy stan s_θ modelu Λ_m generowany wektor obserwacji o_t . Suma prawdopodobieństw generowania obserwacji dla jednego stanu również zawsze wynosi jeden.

Dla modelu na rysunku 4.1 do wygenerowania wszystkich wektorów obserwacji w kolejności od o_1 do o_6 potrzebna jest sekwencja stanów $X = 1, 2, 2, 2, 3, 4, 4, 5$.

Obserwacje (wektory cech) o_t emitowane są z prawdopodobieństwem b_j określonym zazwyczaj jako wielowymiarowy rozkład Gaussa:

$$b_j(o_t) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_j|}} e^{-\frac{1}{2}(o_t - \mu_j)' \Sigma_j^{-1} (o_t - \mu_j)} \quad (4.2)$$

Gdzie:

Σ_j - macierz kowariancji,

μ_j - wektor średni obserwacji,

n - wymiar wektora obserwacji.

Prawdopodobieństwo obserwacji można również aproksymować za pomocą mieszaniny wielowymiarowych rozkładów Gaussa:

$$p_j(o_t) = \sum_{k=0}^{K-1} c_{jk} b_{jk}(o_t) \quad (4.3)$$

$$\sum_{k=0}^{K-1} c_{jk} = 1 \quad (4.4)$$

Gdzie:

K - liczba rozkładów Gaussa,

c_{jk} - k -ty współczynnik wagowy mieszaniny dla stanu j ,

b_{jk} - k -te prawdopodobieństwo obserwacji dla stanu j .

Duża liczba parametrów w przypadku mieszanin o większej liczbie rozkładów wymusza użycie dużej ilości danych treningowych. Można jednak uprościć model redukując liczbę jego parametrów poprzez np.:

- wiązanie parametrów, podczas którego stosuje się wspólną macierz kowariancji dla tych rozkładów, dla których wartości kowariancji są zbliżone,

- zastosowanie diagonalnej macierzy kowariancji, gdzie zakłada się, że elementy wektora cech są wzajemnie nieskorelowane, dlatego dopuszczalne jest zastosowanie macierzy posiadającej elementy tylko na diagonali, powstają przez to błędy aproksymacji, które można minimalizować przez zastosowanie większej liczby rozkładów.

Prawdopodobieństwo $P(O, X|\Lambda_m)$ wygenerowania przykładowej sekwencji wektorów obserwacji o_1, o_2, o_4, o_6 nawiązując do modelu przedstawionego na rysunku 4.1 oblicza się następująco:

$$P(O, X|\Lambda_m) = a_{12}b_2(o_1)a_{22}b_2(o_2)a_{23}b_3(o_4)a_{34}b_4(o_6)a_{45} \quad (4.5)$$

[30]

W praktyce jedynie sekwencja obserwacji O jest znana, natomiast sekwencja stanów X jest ukryta. Wymagane prawdopodobieństwo jest wyliczane przez sumę wszystkich możliwych sekwencji stanów $X = x(1), x(2), x(3), \dots, x(T)$ daną wzorem:

$$P(O|\Lambda_m) = \sum_X a_{x(0)x(1)} \prod_{t=1}^T b_{x(t)}(o_t) a_{x(t)x(t+1)} \quad (4.6)$$

Jako alternatywę dla powyższego wzoru można rozważyć w przybliżeniu, najbardziej prawdopodobną sekwencję stanów daną wzorem:

$$P(O|\Lambda_m) = \max_X \left\{ a_{x(0)x(1)} \prod_{t=1}^T b_{x(t)}(o_t) a_{x(t)x(t+1)} \right\} \quad (4.7)$$

Gdzie:

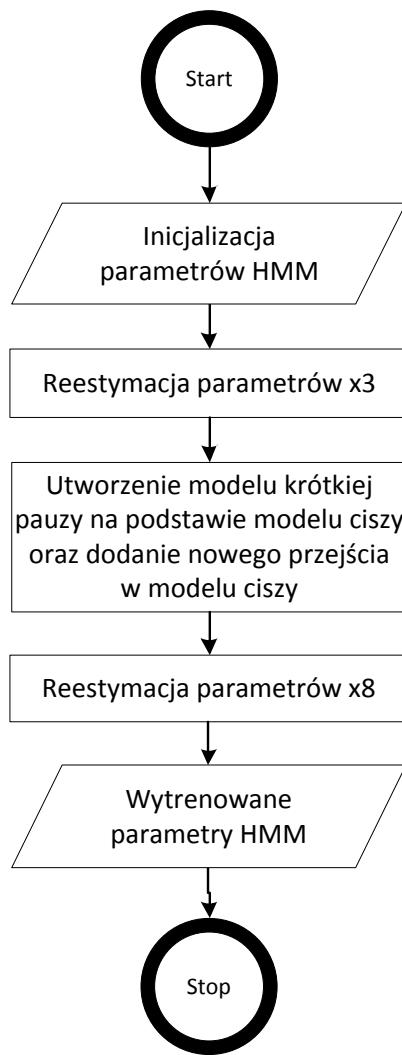
$x(0)$ i $x(T + 1)$ to stan pierwszy i ostatni modelu czyli stany nieemitujące.

Zakłada się, że parametry a_{ij} oraz $b_j(o_t)$ są znane dla każdego modelu. „Tutaj właśnie leży elegancja i skuteczność metody HMM” [32]

4.3. Trenowanie

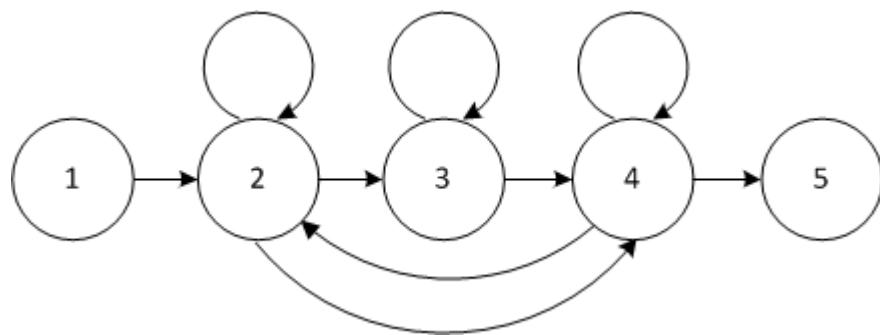
4.3.1. Przebieg

Schemat blokowy fazy trenowania modelu Markowa przyjęty w pracy widoczny jest na rysunku 4.2.



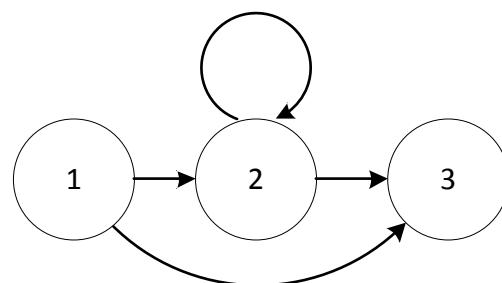
Rysunek 4.2: Trenowanie modeli Markowa na podstawie [30]

Jednofonowe modele Markowa dla ciszy i krótkich pauz są przedstawione kolejno na rysunkach 4.3 oraz 4.4.



Rysunek 4.3: Model HMM dla ciszy

W modelu ciszy znajduje się dodatkowe przejście ze stanu drugiego do stanu czwartego oraz ze stanu czwartego do stanu drugiego.

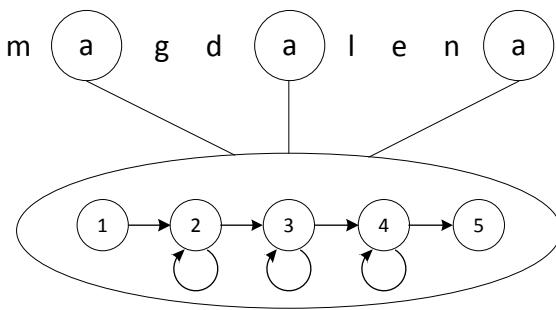


Rysunek 4.4: Model HMM dla krótkiej pauzy

W modelu krótkiej pauzy znajduje się dodatkowe przejście pomiędzy stanami pierwszym oraz trzecim czyli stanami nieemitującymi. Stan trzeci modelu ciszy oraz stan drugi modelu krótkiej pauzy są takie same. Model krótkiej pauzy tworzy się na podstawie modelu ciszy, usuwając stany dwa oraz cztery, powstaje wówczas model trzy-stanowy, gdzie poprzedni stan trzeci staje się stanem drugim modelu pauzy.

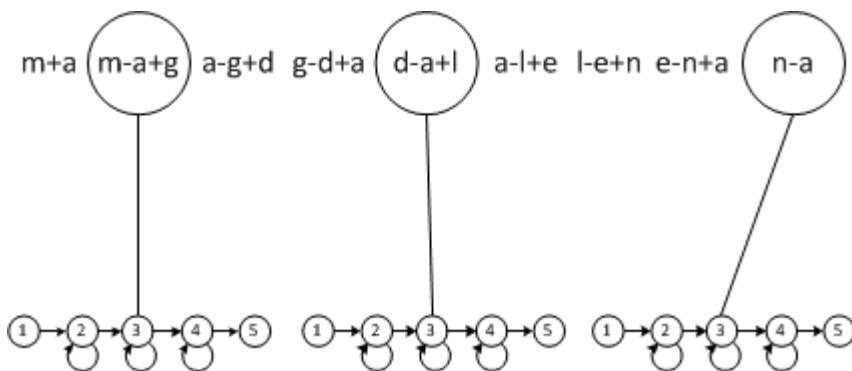
W przypadku gdy najmniejszą niepodzielną jednostką fonetyczną systemu są fonemy powinno się rozpatrywać modele nie tylko dla jednofonów, lecz również dla trójfonów, czyli możliwych kombinacji fonemów w otoczeniu innych fonemów.

Zapis jednofonowy pokazany jest na rysunku 4.5.



Rysunek 4.5: Zapis jednofonowy na podstawie [30]

Natomiast zapis trójfonowy na rysunku 4.6.



Rysunek 4.6: Zapis trójfonowy [30]

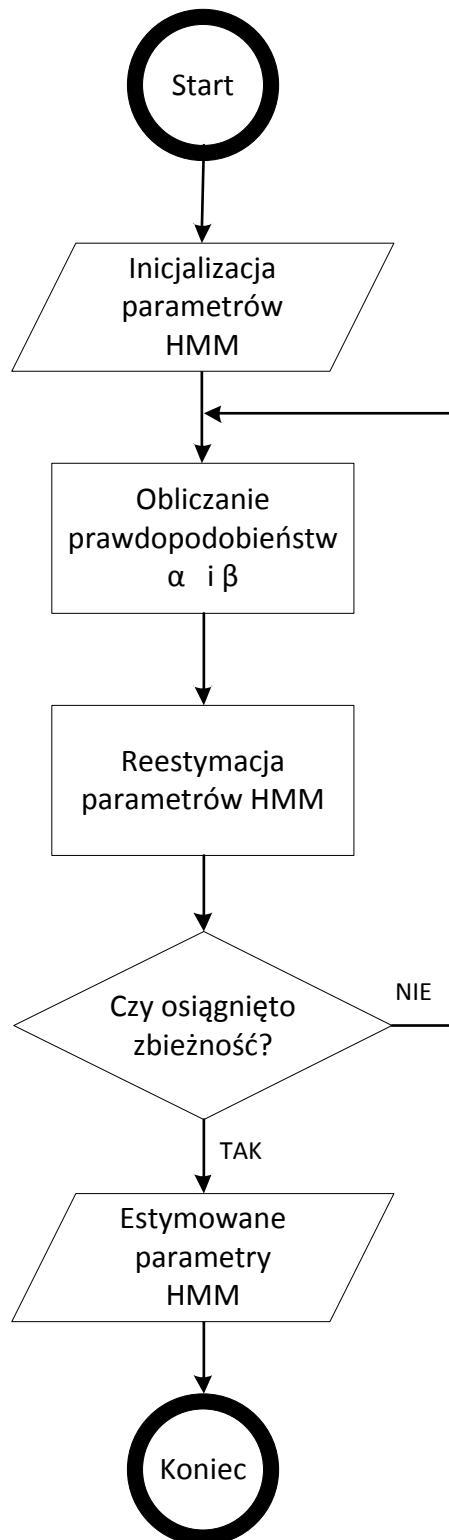
Znak „-” oznacza fonem poprzedzający, natomiast znak „+” fonem następujący.

W przypadku niniejszej pracy opartej na połączonych słowach, powyższe kroki zostały pominięte.

Modele HMM są trenowane dla każdego słowa znajdującego się w słowniku systemu poprzez wielokrotne próbki dźwięku nagrane dla tego słowa.

4.3.2. Algorytm Bauma-Welcha

Algorytm Bauma-Welcha jest wykorzystywany do reestymacji parametrów modeli HMM. Jest oparty na kryterium ML (ang. Maximum Likelihood) - maksymalizacji prawdopodobieństwa wygenerowania sekwencji obserwacji przez model HMM. Jest to procedura iteracyjna w wyniku której można otrzymać lepsze estymaty parametrów HMM. Schemat blokowy został przedstawiony na rysunku 4.7.



Rysunek 4.7: Reestymacja Bauma-Welcha [30]

Początkowe parametry HMM określane są na zasadzie zgadywania, dokładniejsze (w sensie największego prawdopodobieństwa) parametry mogą być znalezione dopiero podczas reestymacji. Estymaty parametrów μ_j oraz \sum_j dla pojedynczego wielowymiarowego rozkładu Gaussa wyznaczane są ze wzorów:

$$\hat{\mu}_j = \frac{\sum_{t=1}^T L_j(t)o_t}{\sum_{t=1}^T L_j(t)} \quad (4.8)$$

$$\hat{\sum}_j = \frac{\sum_{t=1}^T L_j(t)(o_t - \hat{\mu}_j)'}{\sum_{t=1}^T L_j(t)} \quad (4.9)$$

Gdzie:

$L_j(t)$ - oznacza prawdopodobieństwo przebywania w stanie j w czasie t .

Podobne, lecz trochę bardziej skomplikowane są wzory na estymaty prawdopodobieństw przejść pomiędzy stanami, szczegóły znajdują się w rozdziale 8 pozycji [32].

Do obliczenia prawdopodobieństwa $L_j(t)$ wykorzystuje się algorytm *Forward-Backward*.

Prawdodobieństwo *forward* dla modelu M zawierającego N stanów jest zdefiniowane jako:

$$\alpha_j(t) = P(o_1, \dots, o_t, x(t) = j | M) \quad (4.10)$$

Jest to połączone prawdopodobieństwo obserwacji (wygenerowania) pierwszego wektora cech oraz przebywania w stanie j w czasie t . Obliczane jest za pomocą rekursji danej wzorem:

$$\alpha_j(t) = \left[\sum_{i=2}^{N-1} \alpha_j(t-1)a_{ij} \right] b_j(o_t) \quad (4.11)$$

Ta rekursja polega na tym, że prawdopodobieństwo przebywania w stanie j w czasie t oraz emitowania obserwacji wylicza się poprzez zsumowanie prawdopodobieństw *forward* dla wszystkich możliwych poprzedników stanów determinowanych przez prawdopodobieństwa przejść a_{ij} . Stany 1 oraz N są stanami nieemitującymi. Początkowe warunki powyższej rekursji są następujące:

$$\alpha_1(1) = 1 \quad (4.12)$$

$$\alpha_j(1) = a_{1j}b_j(o_1) \quad (4.13)$$

dla $1 < j < N$.

A finalne waruki:

$$\alpha_N(T) = \sum_{i=2}^{N-1} \alpha_i(T) a_{iN} \quad (4.14)$$

Trzeba zauważyć, że z definicji $\alpha_j(t)$,

$$P(O \mid M) = \alpha_N(T) \quad (4.15)$$

Stąd obliczenie prawdopodobieństwa *forward* również daje prawdopodobieństwo całkowite $P(O \mid M)$.

Z kolei prawdopodobieństwo *backward* jest zdefiniowane jako:

$$\beta_j(t) = P(o_{t+1}, \dots, o_T, x(t) = j \mid M) \quad (4.16)$$

Jak w przypadku prawdopodobieństwa *forward*, prawdopodobieństwo *backward* również liczy się za pomocą rekursji:

$$\beta_j(t) = \sum_{j=2}^{N-1} a_{1j} b_j(o_{t+1}) \beta_j(t+1) \quad (4.17)$$

z warunkami początkowymi:

$$\beta_j(T) = a_{iN} \quad (4.18)$$

dla $1 < i < N$ i finalne warunki:

$$\beta_1(1) = \sum_{j=2}^{N-1} a_{1j} b_j(o_1) \beta_j(1) \quad (4.19)$$

Warto zauważyć, że w powyższych definicjach prawdopodobieństwo *forward* jest prawdopodobieństwem połączonym, podczas gdy prawdopodobieństwo *backward* jest prawdopodobieństwem warunkowym.

Prawdopodobieństwo przebywania w danych stanie jest wyznaczane przez iloczyn dwóch prawdopodobieństw zgodnie ze wzorem:

$$\alpha_j(t) \beta_j(t) = P(O, x(t) = j \mid M) \quad (4.20)$$

Stąd:

$$L_j(t) = P(x(t) = j \mid O, M) = \frac{P(O, x(t) = j \mid M)}{P(O|M)} = \frac{1}{P} \alpha_j(t) \beta_j(t) \quad (4.21)$$

Gdzie $P = P(O|M)$.

Zakłada się, że parametry dla HMM są reestymowane dla pojedynczej sekwencji obserwacji, co odpowiada pojedyńczej próbce mowy nagranej do wytrenowania modelu. Do uzyskania dobrych estymat parametrów potrzeba wielu próbek tego samego słowa. [32]

Wzory na reestymację parametrów dla mieszaniny rozkładów Gaussa można znaleźć w rozdziale 8 pozycji. [32]

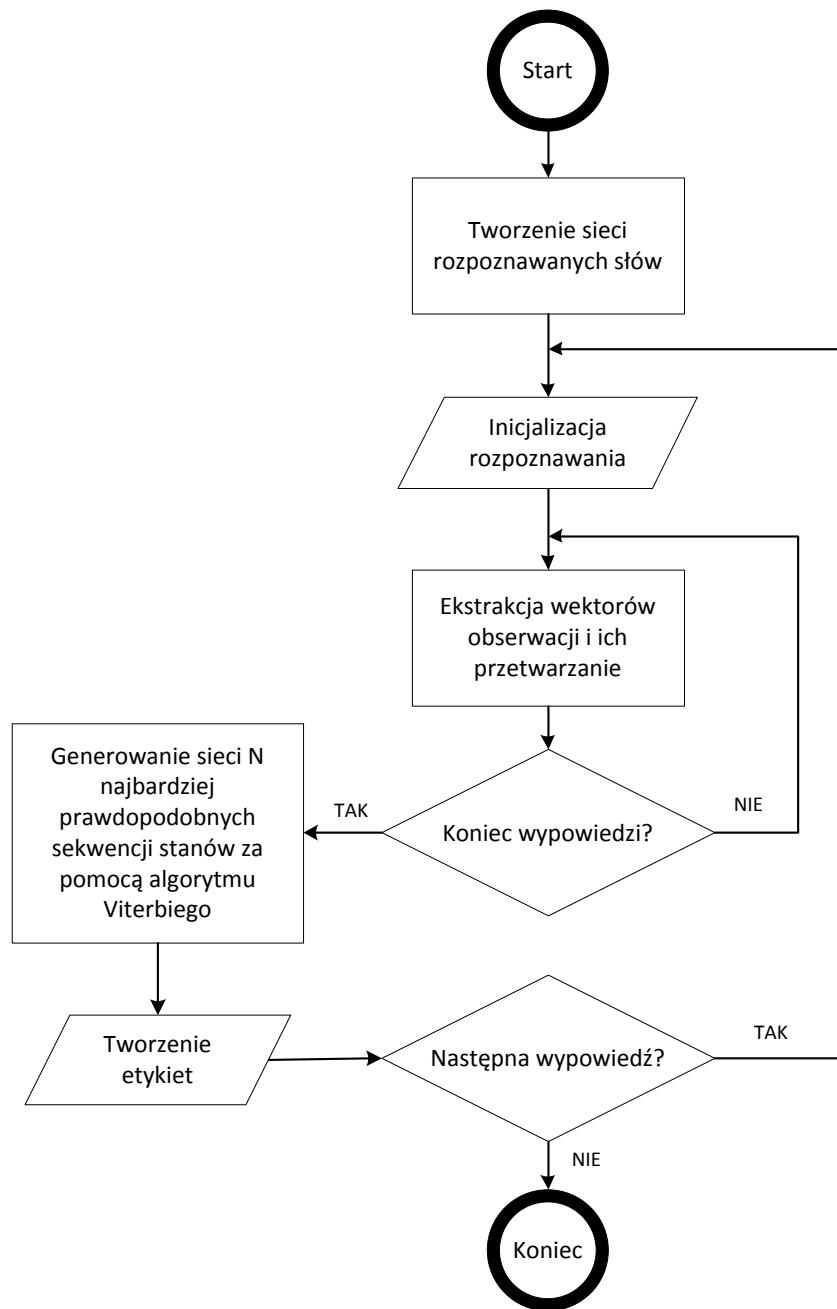
Warto wspomnieć, że podczas wyznaczania prawdopodobieństw *forward* oraz *backward* liczony jest iloczyn dużej liczby prawdopodobieństw. Oznacza to, że przetwarzane liczby mogą stać się bardzo małe. Stąd aby uniknąć problemów numerycznych, powyższe obliczenia powinny być przeprowadzane w skali logarytmicznej.

Podsumowując, problem reestymacji parametrów polega na doborze jak największych prawdopodobieństw przejść pomiędzy stanami oraz prawdopodobieństw wystąpienia obserwacji. Najpierw ustala się początkowe, przybliżone wartości tych parametrów a następnie przebiega reestymacja parametrów, czyli próba znalezienia ich dokładniejszych wartości. [22]

4.4. Rozpoznawanie

4.4.1. Przebieg

Fazę rozpoznawania przedstawia schemat blokowy widoczny na rysunku 4.8.

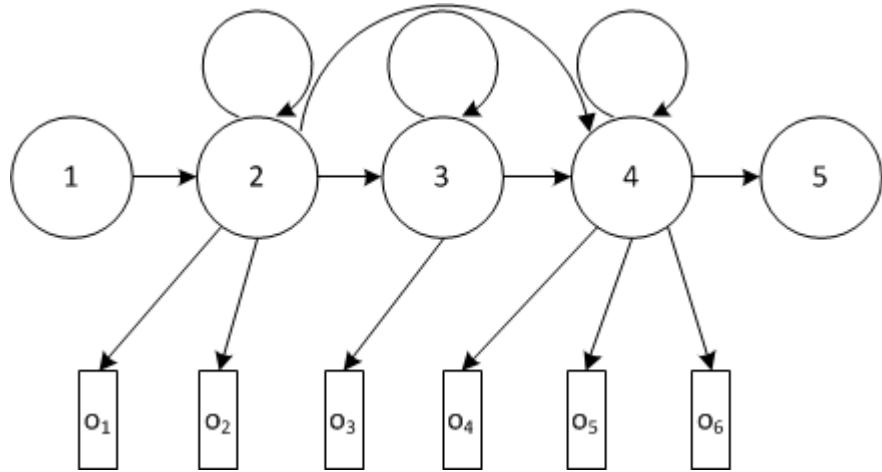


Rysunek 4.8: Schemat blokowy rozpoznawania [30]

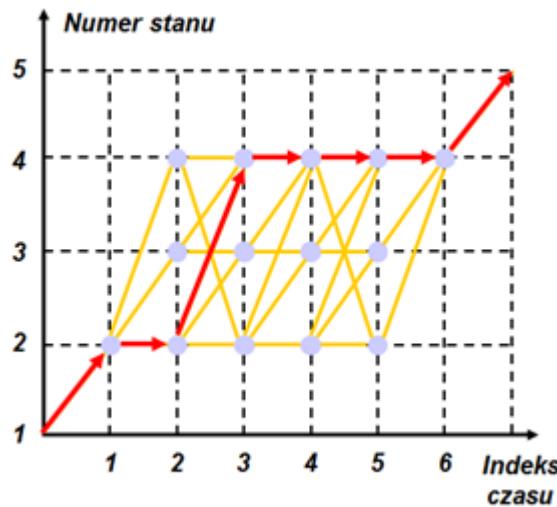
Najważniejszym etapem tej fazy jest wyznaczenie sekwencji najbardziej prawdopodobnych stanów a więc optymalnej ścieżki w modelu. Do rozwiązania tego problemu stosowany jest zazwyczaj algorytm Viterbiego.

4.4.2. Algorytm Viterbiego

Na rysunku 4.10 przedstawiony jest przykład użycia algorytmu Viterbiego dla modelu z rysunku 4.9.



Rysunek 4.9: Przykładowy model HMM



Rysunek 4.10: Algorytm Viterbiego [30]

W każdym kolejnym kroku algorytmu zapamiętywany jest argument maksymalny, wynikiem działania algorytmu jest ścieżka dająca największe prawdopodobieństwo, czyli najbardziej prawdopodobna sekwencja jednostek fonetycznych które zostały przyjęte. [22]

Od strony matematycznej prawdopodobieństwo to, jest wyliczane podobnie jak prawdopodobieństwo *forward* za pomocą rekursji:

$$\phi_j(t) = \max_t \{\phi_i(t-1)a_{ij}\} b_j(o_t) \quad (4.22)$$

$$\phi_1(1) = 1 \quad (4.23)$$

$$\phi_j(1) = a_{1j} b_j(o_1) \quad (4.24)$$

dla $1 < j < N$.

Największe prawdopodobieństwo $\hat{P} = (O|M)$ dane jest wzorem:

$$\phi_N(T) = \max_i \{\phi_i(T)a_{iN}\} \quad (4.25)$$

Jak w przypadku reestymacji tutaj także jest stosowana skala logarytmiczna, stąd równanie ma postać [32]:

$$\psi_j(t) = \max_i \{\psi_i(t-1) + \log(a_{ij})\} + \log(b_j(o_t)) \quad (4.26)$$

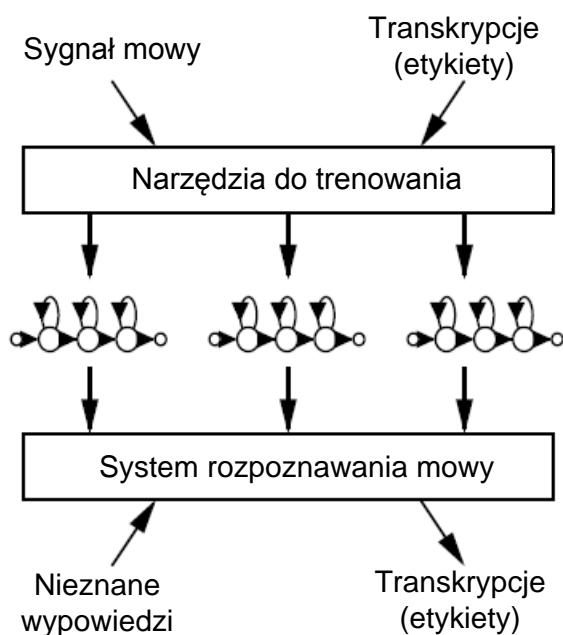
5. Wykorzystane narzędzia

5.1. HTK (Hidden Markov Model Toolkit)

5.1.1. Podstawy HTK

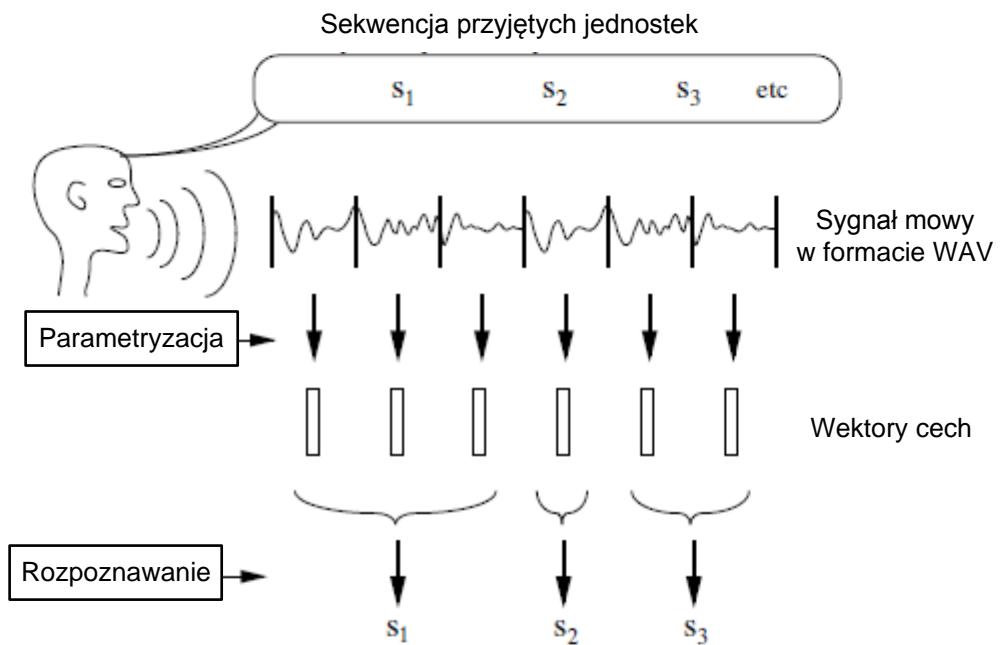
HTK jest zbiorem narzędzi napisanych w języku C i korzystających z HMM w kontekście rozpoznawania mowy. Pierwsza wersja projektu powstała w 1995 na uniwersytecie w Cambridge. Zespołem kierował profesor Steve Young. Projekt początkowo był finansowany przez Microsoft Corporation, ostatecznie jego prawa autorskie przejął Cambridge University Engineering Department. Ostatnia jego wersja o numerze 3.4 powstała w roku 2006. Na tej właśnie wersji opiera się niniejsza praca.

HTK posiada kompleksowy zestaw bibliotek pozwalających przetwarzać sygnał mowy. Jak widać na rysunku 5.1, działanie HTK można ogólnie podzielić na dwie główne fazy. Pierwsza w której odbywa się trenowanie HMM i druga podczas której przy użyciu narzędzi do rozpoznawania mowy otrzymuje się transkrypcję fonetyczną nieznanej wypowiedzi.



Rysunek 5.1: Organizacja bibliotek HTK na podstawie [32]

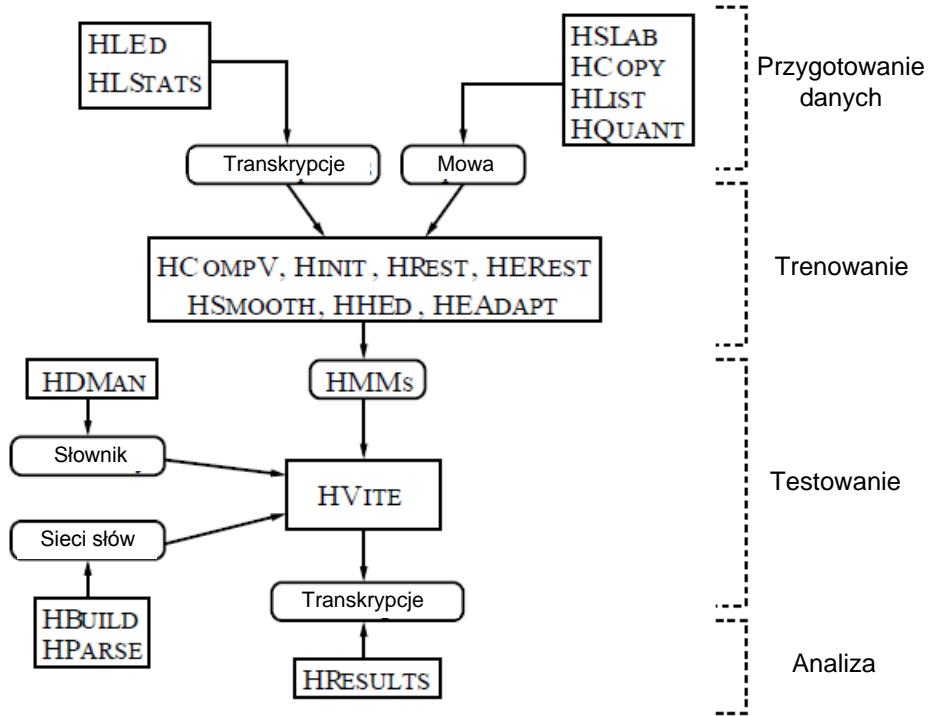
Podstawowe założenia HTK można zobrazować prostym schematem widocznym na rysunku 5.2.



Rysunek 5.2: Enkodowanie/dekodowanie sygnału mowy na podstawie [32]

Założenie, że mowa jest sekwencją pewnych dyskretnych jednostek fonetycznych jest typowym założeniem współczesnych systemów rozpoznawania mowy o czym zostało wspomniane w rozdziale 2 niniejszej pracy.

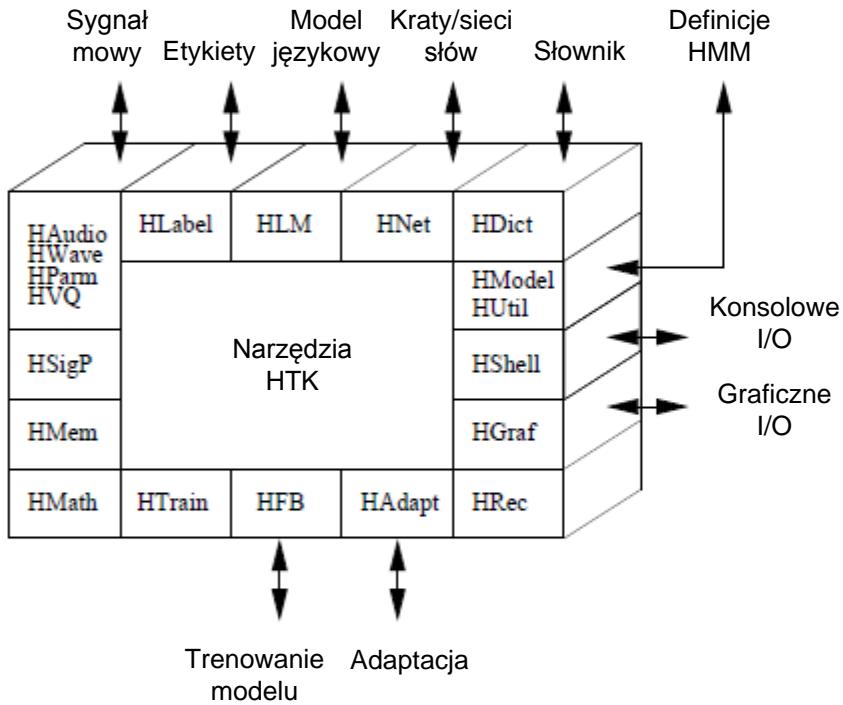
Budowa systemu rozpoznawania mowy ciągłej z użyciem narzędzi HTK przebiega w czterech fazach (rys. 5.3). Zostały one opisane w rozdziale 6 niniejszej pracy.



Rysunek 5.3: Cztery główne fazy budowy systemu rozpoznawania mowy z HTK na podstawie [32]

5.1.2. Architektura

Większość funkcjonalności HTK jest wbudowane w moduły, które są bibliotekami systemowymi. Mogą zostać skompilowane zarówno na systemach z rodziny Windows jak również na systemach opartych o jądro Linuxa. Moduły te zapewniają każdemu narzędziu interfejs do komunikowania się ze światem zewnętrznym w podobny sposób. Dostarczają również centralny zasób najczęściej używanych funkcji. Na rysunku 5.4 przedstawione zostały typowe narzędzia HTK oraz pokazane interfejsy wejść i wyjść.



Rysunek 5.4: Architektura HTK na podstawie [32]

Nad interakcją z użytkownikiem czuwa biblioteka *HShell* oraz biblioteka zarządzająca pamięcią *HMem*. Wsparcie matematyczne zapewnia *HMath*, a operacje związane z przetwarzaniem sygnałów dźwiękowych są kontrolowane przez bibliotekę *HSigP*. Dla każdego interfejsu są przeznaczone odpowiednie typy plików. *HLabel* dostarcza pliki etykiet, *HLM* pliki modelu językowego, *HNet* sieci i krat słów, *HDict* słownika, *HModel* definicji HMM. Wszystkie wejściowe oraz wyjściowe pliki dźwiękowe w formacie WAVE kontroluje bilbioteka *HWave*, parametry tych plików *HParm*. Bezpośrednie wejście dźwięku zapewnia biblioteka *HAudio*, z kolei *HGraf* rysuje wykresy. *HUtil* zawiera narzędzia manipulujące modelami HMM podczas gdy biblioteki *HTrain* oraz *HFB* zapewniają wsparcie w trakcie trenowania modeli. *HRec* z kolei zawiera główne funkcje procesu rozpoznawania. Szczegóły wspomnianych jak i pozostałych narzędzi z pakietu HTK znajdują się w pozycji [32].

5.1.3. Parametry narzędzi HTK

Narzędzia HTK posiadają tradycyjny konsolowy interfejs użytkownika. Każde narzędzie posiada zestaw wymaganych argumentów oraz argumenty opcjonalne. Przykład ich użycia zostanie zaprezentowany na nieistniejącym narzędziu nazwanym *HFoo* (listing 5.1).

Listing 5.1: Przykład wywołania narzędzia *HFoo*

```
HFoo -T 1 -f 34.3 -a -s myfile file1 file2
```

Narzędzie to posiada dwa główne argumenty nazwane *file1* i *file2* oraz argumenty opcjonalne. Argumenty zawsze zaczynają się od pojedynczej litery poprzedzonej znakiem „-” a następnie podawana jest ich wartość. Kolejne opcje (argumenty) rozdzielone są znakiem spacji. Czasami wartość opcji jest wartością typu zmiennoprzecinkowego (opcja *-f*) a innym razem jest to wartość całkowita (opcja *-T*, która zawsze kontroluje wyjście narzędzia czyli, to co zostanie wyświetcone w konsoli) lub też ciąg znaków (opcja *-s*). Bywa też, że opcje są zwykłą flagą (tzw. przełącznikiem) i nie potrzebują żadnej wartości, np. opcja *-a*, która odblokowuje lub zablokuje pewną cechę narzędzia.

Narzędzia mogą być również parametryzowane za pomocą pliku konfiguracyjnego (listing 5.2).

Listing 5.2: Przykład wywołania narzędzia HTK wraz z plikiem konfiguracyjnym

```
HFoo -C config -f 34.3 -a -s myfile file1 file2
```

Wywołanie powyższego polecenia ustawia parametry zapisane w pliku *config*. Pliki konfiguracyjne mogą być czasami alternatywą dla argumentów podawanych podczas wywołania narzędzia w konsoli.

Konsolowe działanie narzędzi HTK może wydawać się trochę przestarzałe w porównaniu z graficznymi interfejsami użytkownika różnych współczesnych aplikacji, lecz daje przejrzysty podgląd działania poszczególnych narzędzi oraz pozwala w prosty sposób zautomatyzować wykonanie kilku funkcji poprzez pisanie skryptów. Aby mieć podgląd wszystkich możliwych opcji wybranego narzędzia wystarczy wywołać je w konsoli bez żadnych dodatkowych argumentów.

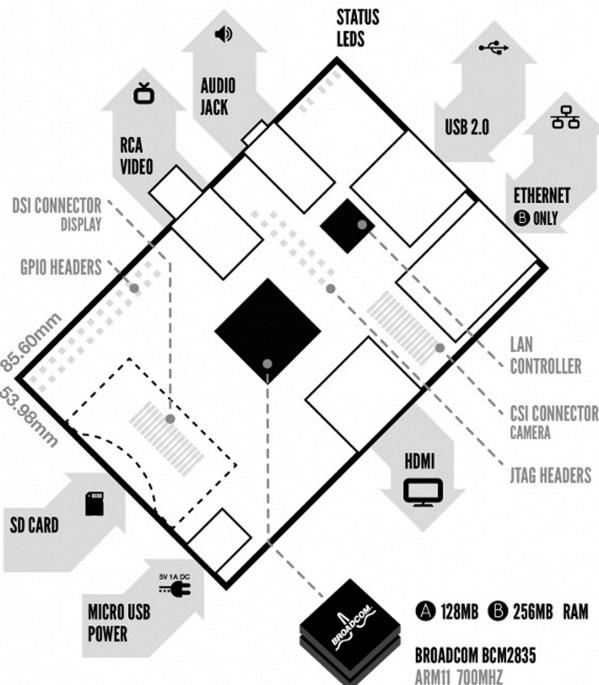
5.2. Raspberry Pi

Raspberry Pi jest produktem fundacji non-profit, której celem było stworzenie małej, taniej oraz w pełni funkcjonalnej platformy komputerowej.

Urządzenie oparte jest na architekturze ARM, nie posiada wbudowanego dysku twardego, lecz w zamian za to, posiada złącze kart SD w celu załadowania systemu operacyjnego i przechowywania danych. Raspberry Pi posiada również złącze USB do podłączenia dowolnych zewnętrznych urządzeń, wyjście dźwięku (np. do podłączenia głośników) oraz złącze Ethernet, które umożliwia podłączenie go do sieci Internet lub LAN za pomocą kabla Ethernet. Urządzenie jest zasilane kablem micro USB. Ciekawostką jest fakt, że Raspberry PI nie posiada wbudowanego zegara czasu rzeczywistego co powoduje, że system musi korzystać z zewnętrznego źródła czasu za pomocą sieci Internet lub pytać użytkownika o czas podczas uruchamiania. Urządzenie pracuje na systemie Linuxowym zainstalowanym na karcie SD. [11]

5.2.1. Parametry

Na rysunku 5.5 przedstawiono schemat Raspberry Pi a w tabeli 5.1 jego szczegółową specyfikację. Powstały dwa modele tego urządzenia A oraz B, różniące się nieco parametrami. Model B jest lepszy w porównaniu do A i to on został wykorzystany w niniejszej pracy.



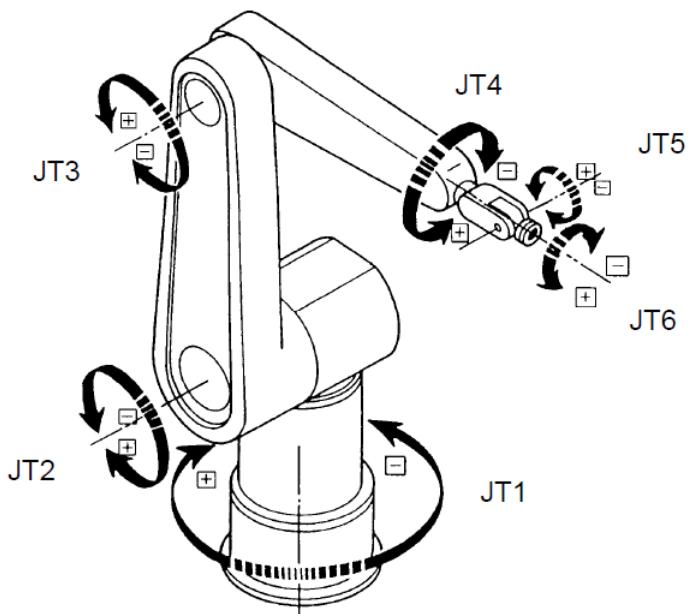
Rysunek 5.5: Schemat Raspberry Pi [13]

Tablica 5.1: Specyfikacja urządzenia [11]

Parametr	Model B
SoC:	Broadcom BCM2835
CPU:	700 MHz ARM1176JZF-S core
GPU:	Broadcom VideoCore IV, OpenGL ES 2.0, 1080p30 h.264/MPEG-4 AVC high-profile decode
Pamięć:	256 MB (współdzielona przez GPU)
Porty USB 2.0:	2 (za pomocą zintegrowanego koncentratora USB)
Wyjścia wideo:	Composite RCA, HDMI
Wyjścia dźwięku:	3.5 mm jack, HDMI
Nośnik danych:	złącze kart SD / MMC / SDIO
Połączenia sieciowe:	10/100 Ethernet (RJ45)
Pozostałe łączka:	8 x GPIO, UART, szyna I ² C, szyna SPI z dwoma liniami CS, +3.3 V, +5 V, masa
Zasilanie:	700 mA (3.5 W)
Źródło zasilania:	5 V przy pomocy złącza MicroUSB, ewentualnie za pomocą złącza GPIO
Wymiary	85.60 x 53.98 mm
Obsługiwane systemy operacyjne:	Debian GNU/Linux, Fedora, Arch Linux

5.3. Robot przemysłowy Kawasaki

Robot wykorzystywany do zaprezentowania sterowania głosowego jest produktem firmy Kawasaki. Jego dystrybucją w Polsce zajmuje się firma ASTOR. Robot jest produktem z serii F oznaczonym symbolem FS03N. Waży zaledwie 20 kg. Posiada sześć stopni swobody co oznacza, że jego ramię porusza się w sześciu płaszczyznach. Każda osi została oznaczona symbolem od JT1 do JT6 (rysunek 5.6).



Rysunek 5.6: Osie robota

Maksymalna prędkość przemieszczania się wzdłuż każdej osi to około 6 m/s.

Kilka dodatkowych parametrów robota przedstawia tabela 5.2.

Tablica 5.2: Parametry modelu FS03N [8]

Parametr	Wartość
Udźwig	3 kg
Zakres osi	JT1 $\pm 160^\circ$
	JT2 $+150^\circ \sim -60^\circ$
	JT3 $+120^\circ \sim -150^\circ$
	JT4 $\pm 360^\circ$
	JT5 $\pm 160^\circ$
	JT6 $\pm 360^\circ$
Zasięg poziomy	700 mm
Zasięg pionowy	830 mm
Powtarzalność (dokładność)	± 0.05 mm

Za sterowanie urządzeniem odpowiada kontroler serii D. Jest on wyposażony w port Ethernet, który

umożliwia komunikację za pomocą kabla sieciowego (korzystając z protokołu Telnet). Drugim możliwym interfejsem komunikacji jest port szeregowy COM, zwany też RS-232.

Sterowanie robotem może być wykonywane za pomocą panelu operatorskiego zwanego *teach pendant*, który jest podłączony do kontrolera robota lub zaprogramowanymi instrukcjami. Dostępne są dwa sposoby programowania robota, język AS oraz *Block teaching* - metoda uczenia odtwarzania. Metoda uczenia polega na zmienianiu pozycji robota z użyciem panela operatorskiego, zapisywaniu wybranych pozycji a następnie odtwarzania zapisanych pozycji. Natomiast programy napisane w języku AS są zapisane w pamięci trwałej kontrolera robota. Użycie języka AS zostało opisane w pozycjach [2] oraz [28].

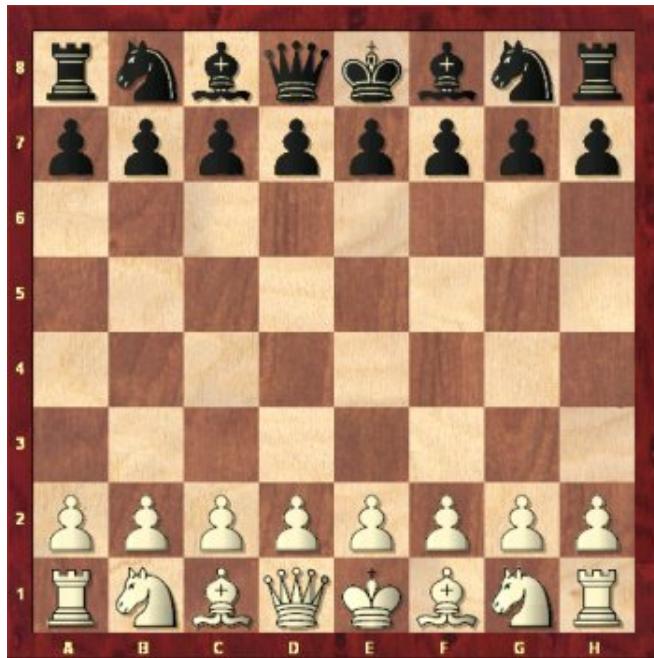
Kinematyka prosta oraz odwrotna została zaimplementowana w kontrolerze robota. Istnieją dwa układy współrzędnych po których może poruszać się ramię robota, umożliwiające bardziej precyzyjne sterowanie ruchami robota. Są to współrzędne globalne oraz układ współrzędnych narzędzia.

5.4. Stockfish

Stockfish jest to nazwa darmowego silnika wykorzystanego w niniejszej pracy jako inteligencja robota podczas rozgrywki szachowej. Jest to projekt wieloplatformowy, dostępny zarówno na systemy z rodziny Windows, Linux jak również na urządzenia mobilne. W niniejszej pracy wykorzystywana jest wersja Linuxowa programu.

Silnik uruchamiany jest jako aplikacja konsolowa, z którą można się komunikować po protokole UCI (ang. Universal Chess Interface). Protokół ten został stworzony aby umożliwić komunikację pomiędzy programami szachowymi a interfejsem użytkownika. Jego dokumentacja znajduje się w pozycji [4].

Pozycje figur na szachownicy mogą być zapisywane w notacji FEN (ang. Forsyth–Edwards Notation). Więcej szczegółów na temat notacji FEN można znaleźć w pozycji [5]. Przy założeniu, że startowe pozycje figur są takie jak na rysunku 5.7, zapis ich pozycji pokazano na listingu 5.3.



Rysunek 5.7: Pozycje startowe figur

Listing 5.3: Zapis początkowych pozycji figur

```
position startpos
```

W niniejszej pracy wykorzystano domyślne ustawienia silnika Stockfish, które zakładają, że pierwszy ruch wykonują figury białe a ilość ruchów została ograniczona do pięćdziesięciu. W przypadku wykonania maksymalnej liczby ruchów i nie doprowadzenia do sytuacji w której jest mat (wygrana jednego z graczy) lub pat (kończy partię remisem), wygrywa gracz, który ma lepszy bilans strat, lub jeśli straty są takie same, ustalany jest remis.

Dla przykładu, jeśli wykonano ruch białego piona z E2 na E4 a następnie czarnego skoczka z B8 na C6 zapis pozycji figur na szachownicy przedstawiono na listingu 5.4.

Listing 5.4: Zapis pozycji figur po wykonaniu dwóch ruchów

```
position startpos moves e2e4 b8c6
```

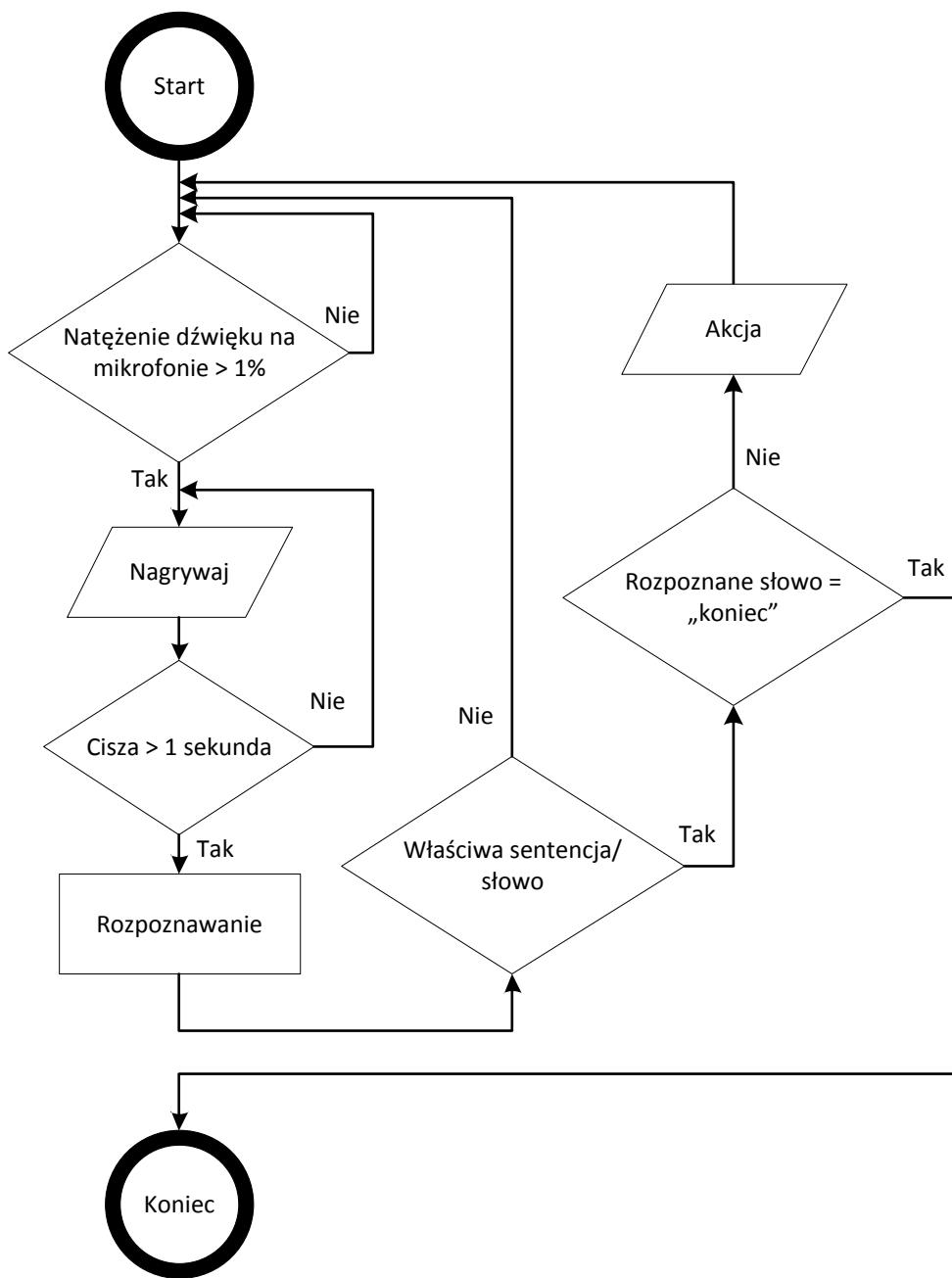
W takim formacie pozycja figur przesyłana jest do programu Stockfish. Następnie uruchamiana jest analiza możliwych ruchów a program zwraca nową pozycję figury.

6. System sterujący robotem za pomocą komend głosowych

6.1. Schemat działania aplikacji

Początkowo główna aplikacja nadzorująca działanie systemu rozpoznawania mowy oraz łącząca w sobie funkcjonalności wybranych narzędzi została napisana w języku C++ jako aplikacja okienkowa (typu desktop) z użyciem QT framework. Ostatecznie jednak autor zdecydował, że skoro tematem przewodnim niniejszej pracy jest komunikacja głosowa, to można zrezygnować z graficznego interfejsu użytkownika. Stąd finalna wersja aplikacji została napisana w języku programowania o nazwie Python i jest uruchamiana w trybie konsolowym. Python jest często określany jako język skryptowy, posiada w pełni dynamiczny system typów a skrypty w nim napisane są wykonywane przez interpreter zainstalowany w systemie.

Schemat działania głównej aplikacji można przedstawić na rysunku 6.1.



Rysunek 6.1: Schemat blokowy opisujący działanie systemu

Do wykonania pierwszych trzech etapów a mianowicie, detekcji sygnału mowy, nagrywania oraz detekcji ciszy zostało wykorzystane narzędzie linuxowe o nazwie Sox [14]. Nagrywanie zostaje rozpoczęte, gdy na mikrofonie zostanie wykryty dźwięk o natężeniu większym niż 1% całosci oraz trwający dłużej niż 0.1 s, a zakończone, gdy cisza (czyli natężenie mniejsze niż 1%) trwa dłużej niż 1 s. Po zakończeniu nagrywania uruchomione zostaje rozpoznanie do którego wykorzystany jest pakiet HTK, którego biblioteki są integralną częścią systemu rozpoznawania mowy. Wynikiem rozpoznawania jest najbardziej

prawdopodobne słowo lub sekwencja słów. Jeżeli zostało rozpoznane słowo kluczowe *koniec* aplikacja kończy swoje działanie (informując o tym użytkownika) a następnie wyłącza urządzenie na którym pracuje. Natomiast w przypadku rozpoznania innej komendy, przeprowadzane jest sprawdzenie jej poprawności. Jeżeli rozpoznana komenda zostanie uznana za poprawną (z punktu widzenia gramatyki na danym etapie rozgrywki), to wtedy zostaje wykonana jakaś akcja. Po wykonaniu akcji aplikacja powtarza swoje czynności i oczekuje na kolejną komendę użytkownika. Akcją może być np. ruch robota przenoszącego figurę na szachownicy lub przejście do innego etapu rozgrywki. Aplikacja uruchamiana jest automatycznie, zaraz po załadowaniu systemu operacyjnego urządzenia na którym pracuje. Szczegóły poszczególnych etapów rozgrywki zostaną wyjaśnione w jednym z kolejnych podrozdziałów. Aplikacje tego typu zwane są w literaturze *command and control*, czyli służą do sterowania za pomocą komend.

6.2. System mikroprocesorowy

W początkowej fazie prac system stworzony na potrzeby niniejszej pracy działał na laptopie z systemem operacyjnym Fedora 17 (dystrybucja Linuxa). Autor doszedł jednak do wniosku, że jeśli podstawą komunikacji systemu z użytkownikiem jest mowa, to klawiatura oraz monitor (matryca laptopa) są zbędne. Do działania systemu rozpoznawania mowy nie jest również potrzebna duża moc obliczeniowa jaką oferował laptop. Zdecydowano się zamiast laptopa użyć systemu mikroprocesorowego o nazwie Raspberry Pi, którego główną zaletą są małe rozmiary urządzenia. Jego parametry zostały opisane w rozdziale 5 niniejszej pracy.

Urządzenie pracuje na systemie Linuxowym o nazwie Raspbian opartym o dystrybucję Debian.

Niestety Raspberry Pi nie posiada wejścia dźwięku wymaganego do podłączenia mikrofonu, dlatego zostało wyposażone w dodatkowy moduł USB (firmy LogiLink) pełniący funkcję karty dźwiękowej, do której zostały podpięte słuchawki z mikrofonem (firmy Sennheiser, model PC 3 Chat) umożliwiające nagrywanie dźwięku. Korzystanie ze słuchawek z mikrofonem ma dodatkową zaletę w postaci stałej, niewielkiej odległości mikrofonu od ust użytkownika. Głośność nagrań oraz SNR sygnału mowy są wtedy zależne jedynie od natężenia głosu użytkownika a nie od odległości mikrofonu od ust mówiącego. Dzięki temu ogranicza się wówczas zmienność parametrów sygnału mowy, będącą potencjalnym źródłem błędów rozpoznawania. Ponadto ze względu na bliską odległość mikrofonu od ust mówiącego można ustawić stosunkowo niski poziom nagrywania, dzięki temu tor audio znaczco tłumii lub nawet eliminuje hałasy dochodzące z otoczenia. Dodatkowo niski poziom nagrywania pozwala zminimalizować niekorzystny wpływ przesterowania sygnału na skuteczność rozpoznawania.

6.3. Szczegóły użycia HTK

System rozpoznawania mowy stworzony z użyciem HTK na potrzeby niniejszej pracy jest zależny od mówcy (w tym przypadku od głosu autora), działa dla połączonych słów (ang. connected words) wypowiadanych ciągiem a jego najmniejszą jednostką fonetyczną są całe słowa, zamiast pojedynczych fonemów. Strategia ta, zwana jest w literaturze *connected speech recognition*. [32]

Podczas rozpoznawania system zawsze dopasowuje najbardziej prawdopodobną sekwencję jednostek mowy zgodną z gramatyką systemu. Co za tym idzie, nawet w przypadku wypowiedzi zawierających przypadkowe słowa lub dźwięki zostanie rozpoznane jakieś słowo lub słowa ze słownika systemu. Jest to tzw. system rozpoznawania komend ze zbioru zamkniętego. System taki wymaga pewnej dyscypliny od użytkownika polegającej na tym, że użytkownik nie może wypowiadać zbędnych słów, które mogłyby być rozpoznane jako komendy sterujące. Niemniej jednak, aby uodpornić system na niektóre sytuacje, do słownika systemu dodano sztuczne słowa takie jak cisza, kaszel, oddech oraz słuchawki (szmer słuchawek), których modele zostały wytrenowane próbami nagrań różnych szumów. W przypadku nagrania podobnego szumu, system z dużym prawdopodobieństwem przyporządkuje mu wymieniony wcześniej model zamiast słów uruchamiających sterowanie, co byłoby niezgodne z intencją użytkownika.

6.3.1. Przygotowanie danych

Jednym z początkowych kroków w trakcie pracy nad systemem rozpoznawania mowy było stworzenie słownika, w którym zapisane są wszystkie używane przez system słowa wraz z ich etykietami.

W początkowej fazie prac, nazwy pól szachownicy określane były bardziej naturalnie w tzn. notacji algebraicznej, jako kombinacja litery oraz cyfry (np. A2, D8). Jako jednostka rozpoznawania są to słowa krótkie. Skuteczność rozpoznawania takich wyrazów nie była zadowalająca (na poziomie około 70%). Dosyć często zdarzały się przypadki, że pola zaczynające się od litery A były błędnie rozpoznawane jako pola zaczynające się od litery H. Podobne błędy występowały w przypadku liter E oraz F. Ich źródłem jest fakt, że te litery różnią się od siebie tylko częścią bezdźwięczną, co utrudnia rozpoznawanie. Paradoksalnie krótkie słowa są trudniejsze do rozpoznawania od słów długich, ponieważ różnią się między sobą na mniejszej liczbie pozycji (bo są krótsze). W celu poprawienia skuteczności oraz wyeliminowania błędów rozpoznawania zastosowano zamiast zwykłego alfabetu z pojedyńczymi literami alfabet fonetyczny ICAO. Stosowany jest on m.in. w komunikacji radiowej w celu poprawienia jednoznaczności wypowiedzi oraz odporności na zakłócenia. Zastosowano wymowę polską. W tabeli 6.1 oraz 6.2 wypisano wykorzystywane litery oraz cyfry oraz ich odpowiedniki z alfabetu ICAO. Zmiana wymowy w przypadku cyfr nastąpiła jedynie dla cyfry 1 oraz 5.

Tablica 6.1: Alfabet fonetyczny ICAO wykorzystywany w systemie - litery na podstawie [1]

Litera	Kod	Wymowa polska
A	Alpha	alfa
B	Bravo	brawo
C	Charlie	czarli
D	Delta	delta
E	Echo	eko
F	Fox potrà	fokstrot
G	Golf	golf
H	Hotel	hotel

Tablica 6.2: Alfabet fonetyczny ICAO wykorzystywany w systemie - cyfry na podstawie [1]

Cyfra	Wymowa polska
1	jedynka
2	dwa
3	trzy
4	cztery
5	piątka
6	sześć
7	siedem
8	osiem

Na listingu 6.1 przedstawiono wszystkie słowa składające się na komendy rozpoznawane przez system.

Przy czym takie komendy jak CISZA, KASZEL, ODDECH oraz SLUCHAWKI nie są akustycznie słowami lecz nagranymi odgłosami (środowiska pracy robota, kaszlu, oddechu użytkownika oraz szmeru przestawianych słuchawek).

Kolejnym krokiem po określaniu słownika, było ustalenie gramatyki systemu, czyli formatu wypowiedzi które będą rozpoznawane.

HTK dostarcza język definicji gramatyki. Składa się on z zestawu definicji zmiennych zawierających wyrażenia regularne będące słowami, które są rozpoznawane.

Przykładowa gramatyka dla wypowiedzi (listing 6.2) sterującymi ruchami robota podczas rozgrywki została zaprezentowana na listingu 6.3.

Listing 6.2: Przykładowe wypowiedzi sterujące podczas rozgrywki z robotem

Alfa jedynka na bravo dwa
 Delta cztery na hotel piątka
 Charlie trzy na foxtrot osiem

Listing 6.3: Przykład definicji gramatyki

```
$cyfra = JEDYNKA | DWA | TRZY | CZTERY | PIATKA | SZESC | SIEDEM | OSIEM;
$litera = ALFA | CHARLIE | BRAVO | DELTA | ECHO | FOXTROT | GOLF | HOTEL;
( SENT-START ($litera $cyfra NA $litera $cyfra) SENT-END )
```

Znak „|” oznacza alternatywę. Jeżeli jakieś słowo ma być opcjonalne, trzeba je wstawić pomiędzy kwadratowe nawiasy („[„, oraz „]”) natomiast nawiasy ostre („<” oraz „>”) oznaczają występowanie powtórzeń.

Listing 6.1: Słownik systemu

ALFA
BRAVO
CHARLIE
CIEMNY
CISZA
COFNIJ
CZTERY
DELTA
DWA
ECHO
FOXTROT
GOLF
HOTEL
JASNY
JEDYNKA
JESTEM
KASZEL
KONIEC
NA
ODDECH
OSIEM
OWCAMI
PIATKA
PIOTR
RESET
SIEDEM
SLUCHAWKI
START
STOP
SZESC
TRZY
TWÓJRUCH
WILKIEM
WITAJ
WSZACHY
WWILKIORCE
ZAGRAJMY
ZATRZYMAJ

Do poprawnego wytrenowania modeli HMM potrzeba wielu próbek zawierających powtarzające się nagrania wszystkich słów ze słownika oraz powiązane z nimi etykiety. Etykiety mogą być zapisane w oddzielnego plikach powiązanych z każdą próbką, lub w jednym głównym pliku zawierającym etykiety wszystkich próbek wykorzystanych do trenowania systemu. Zapis etykiet musi mieć odpowiedni format, dla HTK jest to format *MLF*. Przykład dla pliku dźwiękowego *sample1.wav* zawierającego nagranie wypowiedzi *Alfa jedynka na bravo dwa* pokazano na listingu 6.4.

Listing 6.4: Przykład etykiety w formacie MLF

```
#!MLF!#
" */ sample1 . wav"
ALFA
JEDYNKA
NA
BRAVO
DWA
.
```

W niniejszym systemie została zastosowana strategia zwana w literaturze jako *flat start monophones* [32]. Oznacza to, że zostały jedynie określone etykiety każdej próbki treningowej. Inną możliwą strategią jest określenie dokładnie w jakim odcinku czasowym próbki dana etykieta występuje. W niektórych przypadkach ta strategia zwiększa skuteczność rozpoznawania systemu, niemniej jednak zastosowanie jej w przypadku rozpoznawania mowy ciągłej, jest dosyć trudne do zrealizowania, ponieważ w wielu próbkach treningowych niełatwo jest odnaleźć granice pomiędzy poszczególnymi etykietami (klasami słów) dlatego, że są wypowiadane bez znaczących przerw między nimi i często się na siebie nakładają. Trudności sprawiają również słowa kończące się głoskami bezdźwięcznymi. W tym przypadku odsłuchując daną próbkę i próbując ustalić koniec takiego słowa jest to również bardzo problematyczne.

Jeżeli tworzy się system oparty o fonemy, narzędzie *HLED* konwertuje pliki etykiet, do plików zawierających transkrypcje fonemowe, czyli zamienia zwykły alfabet języka na alfabet fonetyczny. W przypadku niniejszej pracy, która nie jest oparta na fonemach lecz na połączonych słowach został utworzony jedynie plik zawierający etykiety, określające klasy rozpoznawanych słów.

Do nagrania próbek sygnału mowy może zostać użyte narzędzie *HSLab* z pakietu HTK lub inny rejestrator dźwięku. W przypadku niniejszej pracy zostało wykorzystane darmowe narzędzie linuxowe o nazwie *Sox*. Rozdzielcość bitowa podczas nagrywania została ustalona na 16 bitów, natomiast częstotliwość próbkowania na 16 kHz.

Otrzymane nagrania muszą zostać odpowiednio zparametryzowane (musi zostać wykonana ekstrakcja cech). Dokonuje się tego przy użyciu narzędzia *HCopy*, które konwertuje pliki dźwiękowe do plików zawierających odpowiednie cechy sygnału mowy. W niniejszej pracy zostały wykorzystane cechy MFCC, opisane w rozdziale 3 oraz w wielu pozycjach literaturowych m.in. w [32]. Przed użyciem narzędzia *HCopy* musi zostać utworzony plik konfiguracyjny przechowujący parametry w dużym stopniu

wpływające na poprawność działania systemu rozpoznawania mowy. Na listingu 6.5 został pokazany plik konfiguracyjny wykorzystany w niniejszej pracy a następnie została wyjaśniona jego konstrukcja.

Listing 6.5: Plik konfiguracyjny narzędzia *HCopy*

```
SOURCEFORMAT = WAV
SOURCERATE = 625
TARGETKIND = MFCC_0_D
TARGETRATE = 100000.0
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
```

Wartość parametru *SOURCEFORMAT* określa format plików dźwiękowych, z których zostaną wyekstrahowane cechy, w tym przypadku jest to format WAV.

Parametr *SOURCERATE* przechowuje częstotliwość próbkowania nagrani dźwiękowych wykorzystanych do trenowania.

Wartość parametru *TARGETKIND* określa jakie cechy sygnału mowy zostaną wykorzystane (w tym przypadku MFCC) oraz kolejno:

- cyfra *0* oznacza, że do wektora cech zostaje dołączona energia pierwszego współczynnika cepstralnego (jeśli cyfra *0* zastąpiona byłaby literą *E*, wtedy zostałaby obliczona sumaryczna energia wszystkich współczynników cepstralnych),
- litera *D* oznacza, że wektor cech zostanie rozszerzony o pierwsze pochodne (czyli współczynniki delta), jeśli zostałaby dopisana kolejno litera *A*, wtedy byłyby dołączone również drugie pochodne (delta-delta).

Wartość parametru *TARGETRATE* określa ilość wektorów cech stworzonych przetwarzając jedną sekundę próbki dźwiękowej.

Parametr *SAVECOMPRESSED* ustawiony na *T* (oznacza true) włącza kompresję plików przechowujących wyekstrahowane cechy.

Natomiast parametr *SAVEWITHCRC* ustawiony również na *T*, sprawdza poprawność wykonanej kompresji.

Wartość parametru *WINDOWSIZE* determinuje długość trwania pojedynczej ramki sygnału mowy.

Wartości parametrów *SOURCERATE*, *TARGETRATE* oraz *WINDOWSIZE* podane są w skali 100 ns.

Dla parametru *SOURCERATE* wartość 625 oznacza częstotliwość próbkowania 16 kHz, w przypadku

TARGETRATE wartość 100000.0 oznacza, że z jednej sekundy nagrania zostanie utworzonych sto wektorów cech, natomiast dla parametru *WINDOWSIZE* wartość 250000.0 oznacza, że pojedyncza ramka sygnału mowy trwa 25 ms.

Parametr *USEHAMMING* ustawiony na wartość T powoduje użycie okna Hamminga w trakcie wyznaczania cech MFCC, *PREEMCOEF* określa współczynnik preemfazy, *NUMCHANS* określa rozmiar banku filtrów wykorzystywanych podczas wyznaczania cech MFCC, *CEPLIFTER* określa ilość współczynników poddanych liftrowaniu natomiast ostatni parametr pliku konfiguracyjnego, *NUMCEPS*, determinuje ilość współczynników cepstralnych.

Podsumowując z każdej ramki sygnału o czasie trwania 25 ms oraz zachodzących na siebie na odcinku 15 ms, zostały wyekstrahowane 25 elementowe wektory cech z deltą energii.

6.3.2. Trenowanie

Przed rozpoczęciem trenowania z użyciem HTK musi zostać utworzony prototyp modelu HMM, zapisany jako zwykły plik tekstowy, gdzie definiuje się topologię modelu (określa ilość stanów, dopuszczalne przejścia między nimi oraz kilka innych parametrów). HTK zawiera skrypty generujące gotowe prototypy. Modele HMM są tworzone dla każdego słowa ze słownika. Procedura reestymacji przebiega równolegle dla każdego modelu w przypadku mowy ciągłej, lub indywidualnie w przypadku izolowanych słów.

Inicjalizacja niektórych parametrów HMM jest wykonywana za pomocą narzędzia zwanego *HInit*. Narzędzie to, najpierw przydziela po równo wektory obserwacji do każdego ze stanów modelu (prócz stanów nieemitujących) a następnie wyznacza dla nich początkowe wartości parametrów za pomocą wzorów:

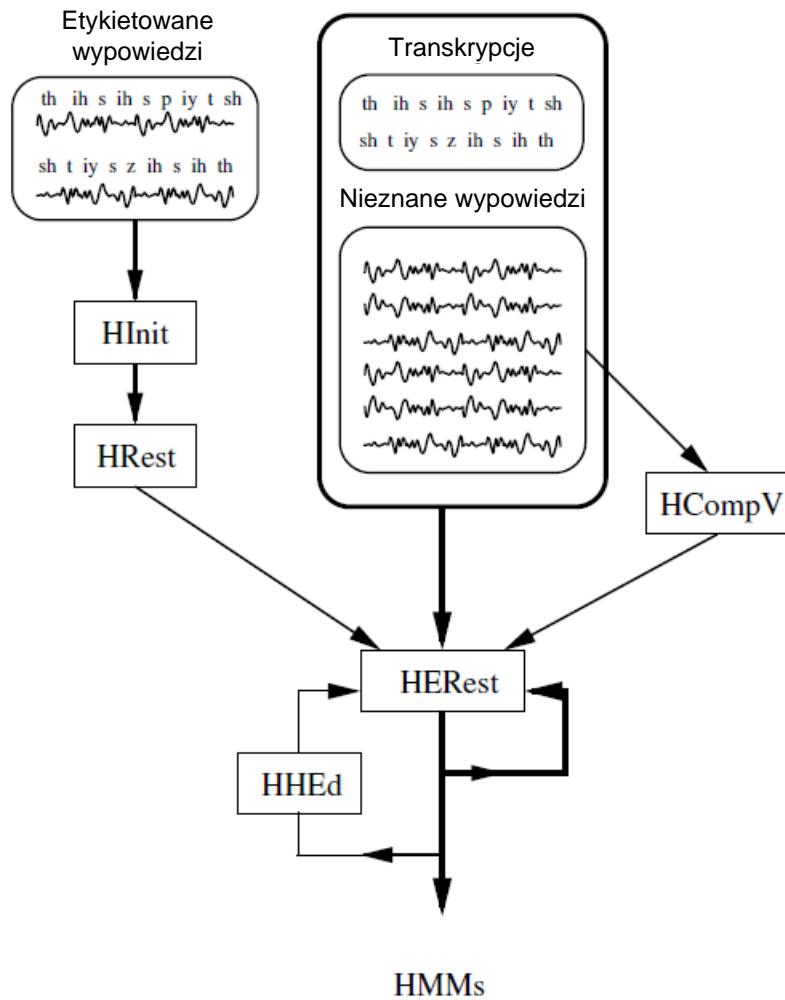
$$\hat{\mu}_j = \frac{1}{T} \sum_{t=1}^T o_t \quad (6.1)$$

$$\hat{\Sigma}_j = \frac{1}{T} \sum_{t=1}^T (o_t - \hat{\mu}_j)(o_t - \hat{\mu}_j)' \quad (6.2)$$

Gdzie $\hat{\mu}_j$ oraz $\hat{\Sigma}_j$ to maksymalne estymaty wektora średniego obserwacji oraz macierzy kowariancji.

Tak dzieje się w przypadku izolowanych słów. Dla mowy ciągłej zamiast narzędzia *HInit* wykorzystywane jest narzędzie *HCompV* a początkowe parametry są jednakowe dla wszystkich modeli. [32]

HTK do reestymacji parametrów HMM używa algorytmu Bauma-Welcha. Ten algorytm jest zaimplementowany w narzędziu zwanym *HERest* oraz *HRest*. Narzędzie *HRest* wykorzystywane jest tylko w przypadku izolowanych słów. Na rysunku 6.2 pokazano jak przebiega proces trenowania HMM za pomocą narzędzi HTK.



Rysunek 6.2: Proces trenowania na podstawie [32]

Narzędzie *HHEd* przygotowuje modele HMM w zależności od kontekstu wypowiedzi. W niniejszej pracy ten etap został pominięty, ponieważ system oparty jest na połączonych słowach, których kontekst nie jest rozważany.

Wynikiem trenowania jest utworzenie modelu akustycznego, który jest podstawą działania rozpoznawania. Model akustyczny zawiera statystyczną reprezentację jednostek (słów) rozpoznawanych w systemie.

6.3.3. Testowanie

Testowanie czyli właściwie faza rozpoznawania, polega na znalezieniu najbardziej prawdopodobnej sekwencji przejść między stanami modelu HMM za pomocą algorytmu Viterbiego zaimplementowanego w narzędziu *HVite*. Narzędzie to na wejściu przyjmuje sieć opisującą dostępne sekwencje słów, słownik

systemu oraz zestaw wytrenowanych modeli HMM. Jako wynik działania narzędzia *HVite* tworzony jest plik zawierający rozpoznane etykiety (klasy słów) nieznanej wypowiedzi w formacie *MLF* (listing 6.6).

Listing 6.6: Przykładowy wynik rozpoznawania w formacie MLF

```
#!MLF!#
"*/sample1.rec"
0 5000000 SENT-START -2374.491211
5000000 22100000 ALFA -8464.383789
22100000 24200000 JEDYNKA -2054.89383
24200000 31200000 NA -804.563265
31200000 42100000 BRAVO -1054.473267
42100000 51200000 DWA -5322.543212
51200000 62100000 SENT-END -2334.412611
.
```

W wygenerowanym pliku podane są etykiety rozpoznanej wypowiedzi, czas (przedział czasowy) w którym poszczególne słowo (etykieta) zostało wykryte, oraz prawdopodobieństwo z jakim zostało wykryte podane w skali logarytmicznej.

6.3.4. Analiza

Po stworzeniu systemu rozpoznawania mowy warto przetestować jego skuteczność. HTK udostępnia narzędzie zwane *HResults*, które na wejściu przyjmuje testowe próbki wypowiedzi wraz z ich etykietami a następnie porównuje z nimi etykiety otrzymane w wyniku rozpoznawania i określa procentową skuteczność działania systemu (listing 6.7).

Listing 6.7: Przykładowy wynik narzędzia *HResults*

```
=====
HTK Results Analysis =====
Date: Sun Oct 22 16:14:45 2012
Ref : testrefs.mlf
Rec : recout.mlf
----- Overall Results -----
SENT: %Correct=98.50 [H=197, S=3, N=200]
WORD: %Corr=99.77, Acc=99.65 [H=853, D=1, S=1, I=1, N=855]
=====
```

Gdzie w linii z *SENT*:

- H oznacza liczbę sentencji rozpoznanych,

- S sentencji nieroznalezionych,
- a N oznacza liczbę wszystkich sentencji.

W przypadku linii z *WORD* analogicznie tylko, że tutaj chodzi o pojedyncze słowa i tak kolejno:

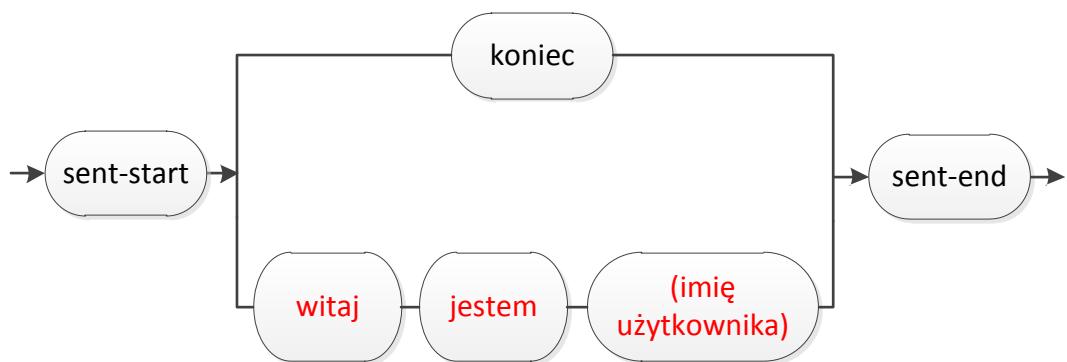
- H oznacza liczbę słów nieroznalezionych,
- S słów rozpoznanych,
- N oznacza liczbę wszystkich słów,
- a D oraz I są to błędy rozpoznawania.

Natomiast *Acc* oznacza skuteczność rozpoznawania podaną w procentach. [32]

6.3.5. Etapy rozgrywki

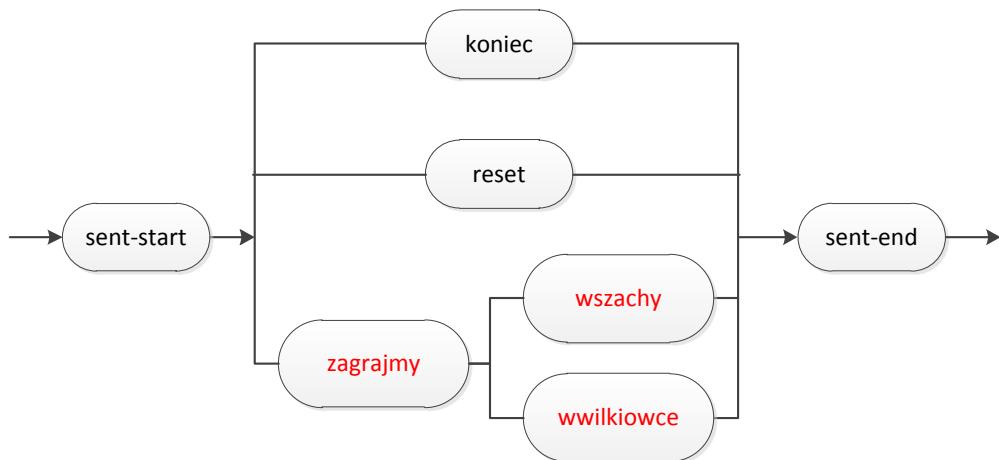
Działanie systemu można podzielić na sześć etapów, podczas których oczekiwana jest ustalona sekwencja (lub słowo) i jej rozpoznanie determinuje wykonanie jakiejś akcji lub jej brak. O braku akcji w przypadku komendy niezgodnej z gramatyką systemu na danym etapie, użytkownik zostaje poinformowany za pomocą komunikatu głosowego. Na rysunkach od 6.3 do 6.8 przedstawiono gramatyki systemu na każdym etapie jego działania w postaci sieci słów. Wypowiedzi których rozpoznanie przechodzi do kolejnego etapu zaznaczono na czerwono. Wyjątkiem jest jedynie etap szósty, gdyż jest to ostatni etap rozgrywki, wypowiedź *Twój ruch* informuje system, że on jako przeciwnik może wykonać swój ruch, natomiast komenda *cofnij* uruchamia akcję cofającą ostatni wykonany ruch użytkownika i umożliwia ponowne jego wykonanie wracając do etapu piątego.

Na rysunku 6.3 przedstawiono gramatykę etapu pierwszego. System oczekuje tutaj od użytkownika przedstawienia się w wybranym formacie.



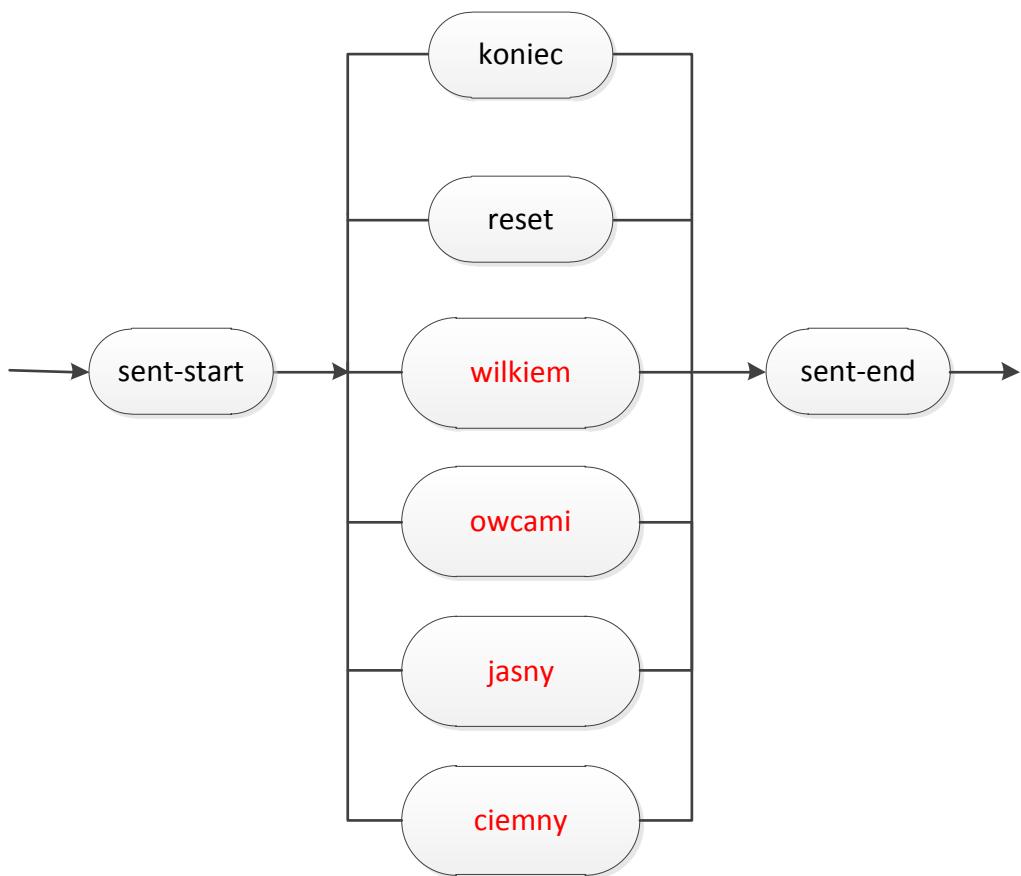
Rysunek 6.3: Gramatyka dla etapu pierwszego

Na rysunku 6.4 widoczna jest gramatyka dla etapu drugiego. Na tym etapie użytkownik wybiera rodzaj rozgrywki.



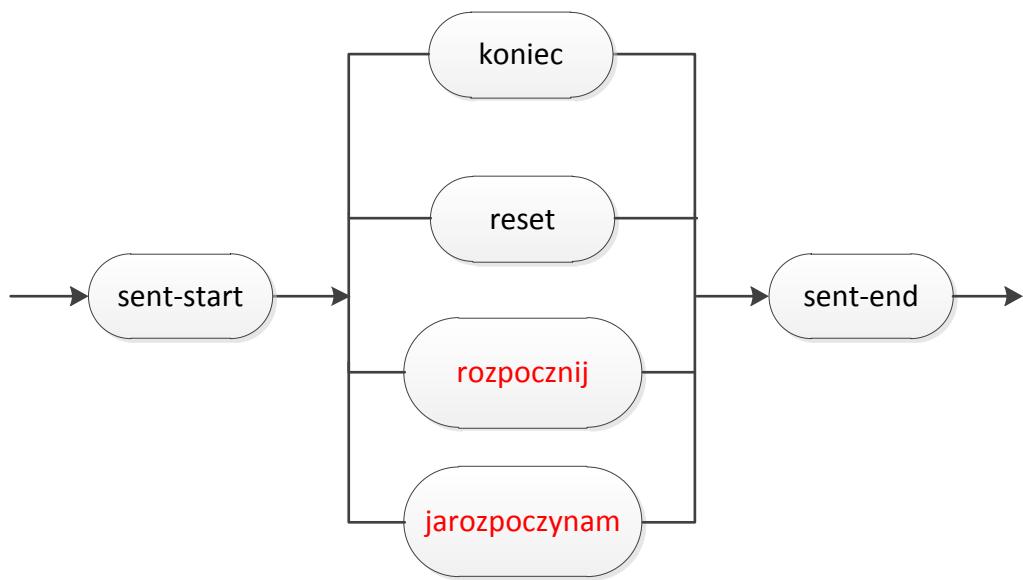
Rysunek 6.4: Gramatyka dla etapu drugiego

W kolejnym etapie o numerze trzy, którego gramatyka jest przedstawiona na rysunku 6.5 użytkownik wybiera kolor figur w zależności od rodzaju rozgrywki, który wybrał w etapie poprzednim.



Rysunek 6.5: Gramatyka dla etapu trzeciego

Etap numer cztery (gramatyka jest widoczna na rysunku 6.6) ma za zadanie określić gracza rozpoczęjącego rozgrywkę. Ten etap jest pomijany w przypadku gry w szachy, ponieważ zgodnie z przyjętymi zasadami, rozpoczyna graczy, który wybrał jasny kolor figur.



Rysunek 6.6: Gramatyka dla etapu czwartego

Etap piąty (gramatyka widoczna na rysunku 6.7) jest najważniejszym etapem działania systemu, ponieważ tutaj odbywa się właściwa rozgrywka. Etap ten wraz z kolejnym są powtarzane iteracyjnie aż do momentu wykrycia przez aplikację ustawień figur na szachownicy oznaczających koniec rozgrywki (wygraną lub przegrana jednego z graczy) dla danej gry lub przerwania przez gracza rozgrywki za pomocą komendy *koniec*. Pole szachownicy (zawierającej 64 pól) jest opisane literą oraz cyfrą. Wypowiedź ma za zadanie określić ruch figury, jej pozycję źródłową oraz docelową. Format wypowiedzi jest pokazany na listingu 6.8. Zmienne *pole_źródłowe* oraz *pole_docelowe* określają wybrane pole na szachownicy.

Zgodnie z przykładem pokazanym na listingu 6.8, wykonany zostanie ruch figury z pola A1 na pole H8.

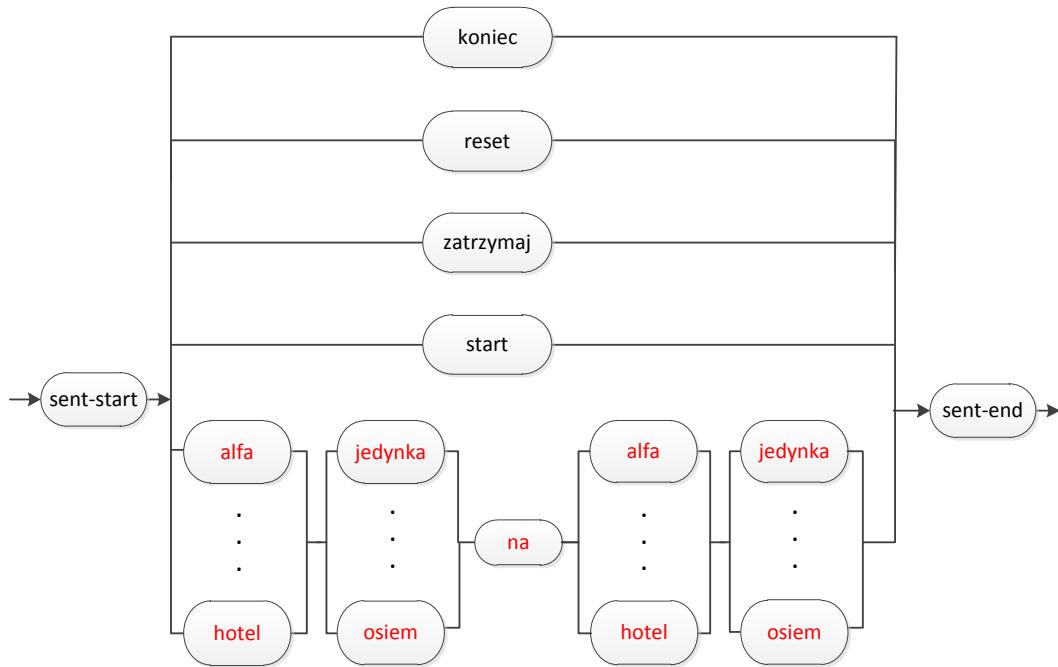
Na tym etapie istnieje również możliwość zatrzymania rozgrywki na maksymalny czas (zdefiniowany w aplikacji) około pięciu minut (komenda *zatrzymaj*). Po tym czasie użytkownik zostaje poinformowany, że aplikacja przerwa pracę i nastąpi jej wyłączenie. Zanim jednak to nastąpi, po upływie około trzech minut użytkownik dostaje ostrzeżenie (komunikat głosowy), że niebawem rozgrywka zostanie przerwana. Wznowienie rozgrywki jest możliwe korzystając z komendy *start*.

Listing 6.8: Format wypowiedzi określającej przemieszczenie figury na szachownicy

`pole_źródłowe na pole_docelowe`

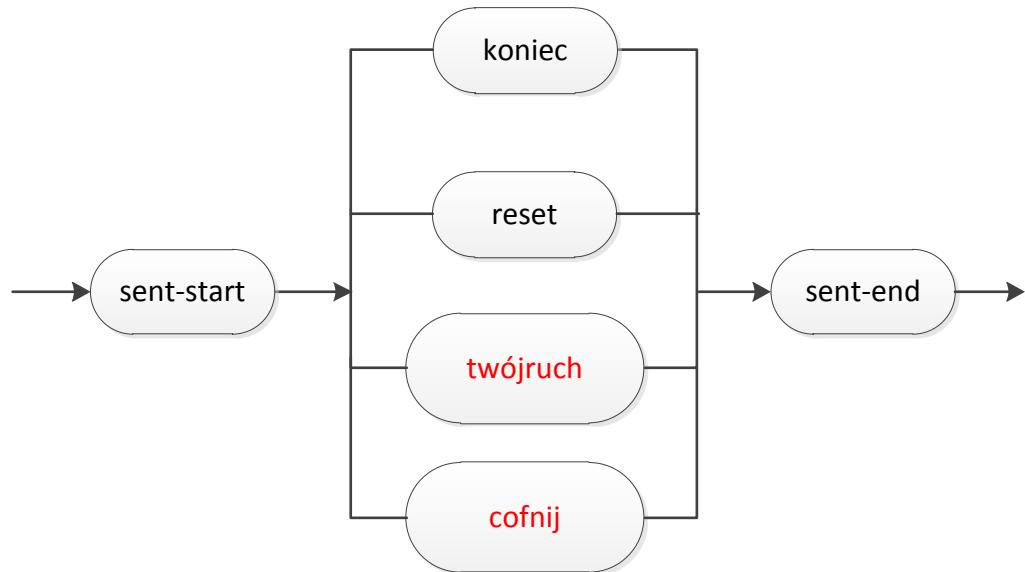
Przykład :

`alfa jedynka na hotel osiem`



Rysunek 6.7: Gramatyka dla etapu piątego

W etapie szóstym (gramatyka widoczna na rysunku 6.8) użytkownik informuje swojego przeciwnika, że ten może wykonać swój ruch lub ma możliwość cofnięcia posunięcia wykonanego na poprzednim etapie tj. piątym.



Rysunek 6.8: Gramatyka dla etapu szóstego

Na każdym z dostępnych etapów istnieje możliwość przerwania rozgrywki poprzez komendę *koniec*. Natomiast komenda *reset* jest dostępna na wszystkich etapach rozgrywki prócz pierwszego, pozwala ona powrócić do etapu początkowego (pierwszego).

6.4. Robot

Robot przemysłowy z którym współpracuje system mikroprocesorowy wraz z systemem rozpoznawania mowy jest wykorzystywany do podnoszenia figur położonych na szachownicy. Do tego celu zostały wycięte metalowe uchwyty przyjmocowane do chwytaka robota. Wszystkie pozycje figur zostały zapisane na stałe w pamięci kontrolera robota. Zdjęcie stanowiska robota wraz z całym osprzętem pokazano na rysunku 6.9.

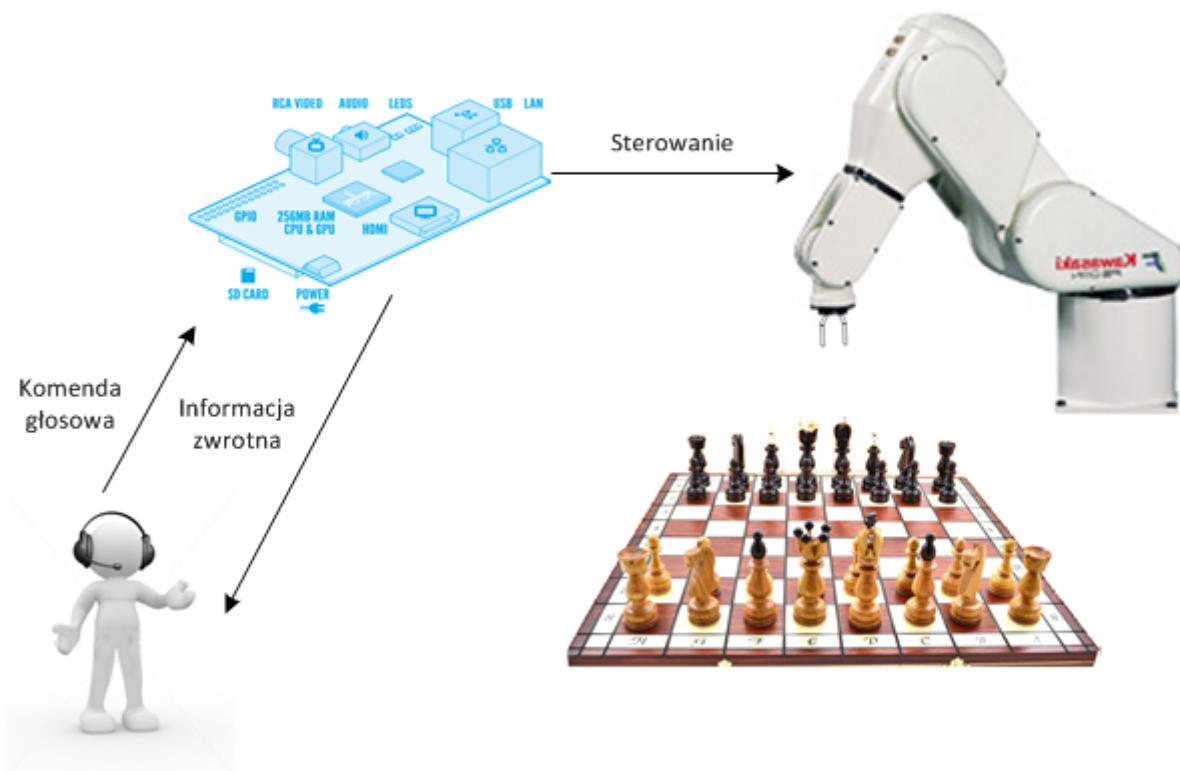


Rysunek 6.9: Zdjęcie stanowiska robota

Uproszczony schemat sterowania robotem przedstawiono na rysunku 6.10. Użytkownik po wydaniu komendy dostaje od systemu odpowiedź zwrotną (która jest ustalonym sygnałem dźwiękowym lub nagraną wcześniej wypowiedzią) potwierdzającą uruchomienie rozpoznawania oraz informującą czy roz-

poznana komenda jest zgodna z gramatyką rozgrywki na danym etapie. Nagrania odtwarzane przez aplikację mają za zadanie symulować rozmowę z robotem. Zostały nagrane z wykorzystaniem syntezatora mowy Ivona na stronie domowej projektu [15].

Komunikacja ze sterownikiem robota odbywa się za pomocą kabla sieciowego Ethernet po protokole Telnet.



Rysunek 6.10: Uproszczony schemat sterowania robotem

Do kontrolowania ruchów robota podczas przenoszenia figur na szachownicy został wykorzystany program napisany w języku AS. Na listingu 6.9 pokazano program wykorzystywany przez kontroler do sterowania robotem podczas rozgrywki. Jest on zapisany w pamięci trwałej kontrolera robota.

Listing 6.9: Program napisany w języku AS, służący do sterowania robotem podczas rozgrywki

```
CLOSEI
JAPPRO #source,100
LMOVE #source
OPENI
TWAIT 2
LDEPART 120
JAPPRO #target,100
```

```
LMOVE #target  
CLOSEI  
TWAIT 2  
LDEPART 120  
LMOVE #center
```

Polecenie CLOSEI otwiera chwytkę robota, natomiast OPENI zamyka (trocze mało intuicyjne na zewnątrz w tym przypadku). Instrukcja JAPPRO służy do przesunięcia chwytnika robota na podaną odległość powyżej celu, instrukcja LMOVE przechodzi ruchem liniowym do celu, instrukcja LDEPART unosi chwytkę nad szachownicę (na podaną wysokość) natomiast instrukcja TWAIT opóźnia wykonanie programu o podaną wartość w sekundach.

W trakcie rozgrywki z robotem, aplikacja działająca na urządzeniu Raspberry Pi wysyła polecenie po protokole Telnet przedstawione na listingu 6.10.

Listing 6.10: Polecenie uruchamiające program wykorzystywany do rozgrywki

```
ex pg753
```

Umożliwia ono wykonanie przez kontroler robota programu sterującego ruchami robota podczas przenoszenia figur na szachownicy. Wcześniej jednak ustawiane są zmienne globalne *source* oraz *target* przechowujące pozycję wybranych figur na szachownicy, które podobnie jak program sterujący są zapisane w pamięci trwałej kontrolera robota. Przykład ustawiania powyższych zmiennych pokazano na listingu 6.11.

Listing 6.11: Polecenia ustawiające zmienne *source* oraz *target*

```
POINT #source=#b1  
POINT #target=#a3
```

Po ustawieniu zmiennych zgodnie z listingiem 6.11 oraz wykonaniu polecenia uruchamiającego program sterujący ruchami robota, przeniesie on figurę z pola B1 na pole A3.

6.5. Gry planszowe

Aplikacja wynik rozpoznanej komendy głosowej określającej ruch figury na szachownicy zamienia na notację algebraiczną (np. *alfa jedynka* zostaje zamienione na *A1*), która jest łatwiejsza w interpretacji. Otrzymana informacja jest wysyłana w odpowiednim formacie do programu Stockfish lub funkcji realizującej algorytm gry Wilk i owce.

6.5.1. Szachy

Zasady gry w szachy są powszechnie znane oraz szeroko opisane w literaturze, dlatego autor ograniczy się jedynie do podania przykładowej pozycji internetowej w której można znaleźć szczegółowy opis zasad rozgrywki, mianowicie [17].

Podczas komunikacji z programem szachowym Stockfish, aplikacja wysyła oraz odbiera pozycje figur na szachownicy oraz uruchamia analizę umożliwiającą ruch przeciwnika komputerowego. W przypadku szachów, domyślnie pierwszy ruch wykonywany jest figurami jasnymi, stąd jeśli użytkownik wybierze kolor jasny aplikacja automatycznie ustawi ciemny kolor figur graczu komputerowemu. Natomiast w odwrotnym wypadku, gracz komputerowy wykona pierwszy ruch a więc najpierw zostanie uruchomiona analiza, której efektem będzie otrzymanie nowej pozycji figury jasnej. Jeżeli użytkownik wybierze kolor jasny i wykona ruch aplikacja najpierw wysyła programowi Stockfish pozycję figur na szachownicy zawierającą ruch użytkownika, a następnie uruchamia analizę i odbiera wynik zawierający nową pozycję figury ciemnej gracza komputerowego.

Ruch gracza komputerowego można otrzymać wysyłając programowi Stockfish komendę pokazaną na listingu 6.12.

Listing 6.12: Polecenie uruchamiające analizę dostępnych ruchów

```
go depth 20
```

Wartość po słowie *depth* ustala głębokość do której wykonywana jest analiza przez program Stockfish. Im mniejsza wartość, tym ruchy będą mniej efektywne, można na tej zasadzie ustawić poziom trudności rozgrywki z graczem komputerowym.

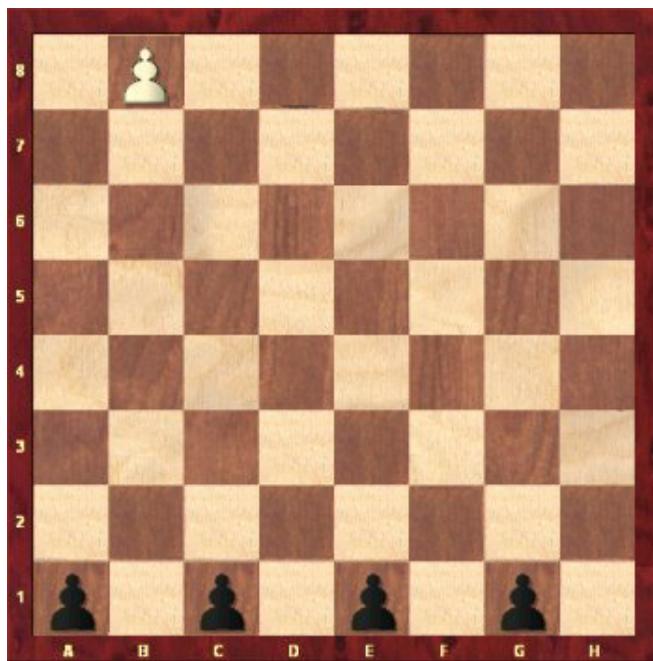
W trakcie rozgrywki wykonywane jest sprawdzanie poprawności ruchu użytkownika, który w razie próby wykonania ruchu niezgodnego z zasadami prowadzonej gry otrzymuje ostrzeżenie w formie komunikatu głosowego o niedozwolonym ruchu.

6.5.2. Wilk i owce

Gra *Wilka i owiec* znana jest również pod nazwą *Wilka i psów*. Jest to gra o typie warcabowym. Jeden z graczy dysponuje jednym jasnym pionem (wilkiem) natomiast przeciwnik ma do dyspozycji cztery piony czarne (owce).

Ruchy odbywają się jedynie po ciemnych polach. Owce zajmują pozycje w pierwszym rzędzie szachownicy, natomiast wilk zajmuje jedną z czterech dostępnych pozycji w rzędzie ostatnim.

Pozycja początkowa figur na szachownicy przyjęta w pracy podczas gry Wilk i owce została przedstawiona na rysunku 6.11.



Rysunek 6.11: Pozycja początkowa figur w grze Wilk i owce

Celem wilka jest przedostanie się na pierwszy rzad pól szachownicy, natomiast gracz mający do dyspozycji owce wygrywa jeśli uda mu się zablokować wilka tak aby nie mógł on wykonać żadnego posunięcia. Wilk w jednym ruchu może przesunąć się o jedno pole po przekątnej w dowolną stronę, natomiast owce mogą przesuwać się również o jedno pole po przekątnej, lecz tylko do przodu. Gracz dysponujący wilkiem może się cofać, w przeciwnieństwie do gracza dysponującego owcami.

Jako inteligencja robota podczas rozgrywki został napisany prosty algorytm polegający z punktu widzenia wilka, na ruchach do przodu jeśli istnieje taka możliwość a w przeciwnym wypadku na cofaniu się. W sytuacji gdy owce przesuwają się równomiernie, to znaczy, że wybrana owca nie wykona więcej niż jednego ruchu przed pozostałymi trzema, to gracz dysponujący wilkiem jest skazany na porażkę. Dlatego w stworzonym algorytmie owce nie poruszają się w sposób bezbłędny aby umożliwić wilkowi wygraną w niektórych przypadkach.

Tak jak podczas gry w szachy, tutaj również wykonywana jest walidacja ruchów użytkownika.

7. Podsumowanie

W niniejszej pracy został opisany proces przygotowania systemu rozpoznawania mowy. Zostały przedstawione narzędzia potrzebne do sterowania robotem w grach planszowych. Opisane zostały także podstawowe informacje o sygnale mowy oraz podstawa systemów rozpoznawania mowy.

Część praktyczna niniejszej pracy składa się z silnika rozpoznawania mowy wytrenowanego z użyciem HTK oraz aplikacji napisanej w Pythonie, która łączy w sobie wszystkie założone funkcjonalności takie jak detekcję i nagrywanie dźwięku, rozpoznawanie mowy z użyciem bibliotek HTK, sterowanie robotem oraz gry planszowe (wilk i owce, szachy). Aplikacja ta jest typu *proof of concept*, dowodzi spełnienia postawionych założeń oraz prezentuje przykładowe zastosowanie rozpoznawania mowy do sterowania urządzeniem elektronicznym, w tym przypadku robotem.

Działanie systemu stworzonego na potrzeby niniejszej pracy zostało przetestowane a uzyskana skuteczność jest zgodna z założeniami i pozwala na poprawne sterowanie robotem. Po przygotowaniu wszystkich narzędzi i skryptów, można niewielkim nakładem pracy zbudować w pełni działający oraz skuteczny system rozpoznawania mowy o małym słowniku.

7.1. Jakość systemu

HTK do poprawnego wytrenowania systemu wymaga conajmniej 3 próbek dla każdego słowa ze słownika. Do uzyskania zadowalającej skuteczności systemu z małym słownikiem oraz po wyeliminowaniu podobieństw słów wystarczy po 10 próbek dla każdego słowa. System stworzony na potrzeby niniejszej pracy posiada słownik składający się z 39 słów, z których każde występuje conajmniej po 20 razy we wszystkich nagraniach treningowych. Słowa zostały dobrane tak aby były długie oraz jak najbardziej różniły się od siebie, co pomaga osiągnąć wysoką skuteczność rozpoznawania na poziomie około 99%. Uzyskanie takiej skuteczności wymaga od użytkownika pewnej dyscypliny. Komendy powinny być wypowiadane w normalnym tempie, wyraźnie oraz z zachowaniem odpowiedniej głośności (komendy nie muszą być wypowiadane podniesionym głosem, lecz nie mogą być również wypowiadane zbyt ciicho). Użytkownik musi także uważać, aby nie wypowiadać przypadkowych słów, ponieważ zostaną im przyporządkowane słowa ze słownika systemu, które mogą być komendami sterującymi. Ilość stanów w modelu HMM została dobrana eksperymentalnie. HTK domyślnie tworzy modele 5-cio stanowe, zostały również przetestowane modele 7, 9, 11 oraz 15-sto stanowe. Największą skuteczność uzyskano w przypadku 15-sto stanowych modeli HMM, z tym, że różnice w poprawie skuteczności w zależności od ilości stanów w modelu były niewielkie. Początkowym wyznacznikiem tej skuteczności było przetesto-

wanie systemu na próbkach wykorzystanych do jego wytrenowania. Po przeprowadzeniu takiego testu skuteczność wyniosła blisko 100% jeśli chodzi o rozpoznawanie pojedynczych słów oraz około 95% w przypadku rozpoznawania całych sentencji. W początkowej fazie prac, gdy słownik zawierał mniej słów oraz było mniej nagrani treningowych rozpoznawanie próbek treningowych dawało 100% poprawność.

7.2. Możliwości rozwoju

W chwili składania tej pracy, dostępne gry to wilk i owce oraz szachy. Sterowanie w nich przebiega w podobny, niezbyt skomplikowany sposób. Warto byłoby przetestować jakąś grę, w której sposób sterowania jest w większym stopniu skomplikowany, co za tym idzie, słownik systemu mógłby ulec powiększeniu. Wymagałoby to również nagrania większej ilości danych treningowych.

Ciekawym kierunkiem rozwoju w opinii autora byłoby oparcie systemu rozpoznawania mowy na fonemach jako najmniejszych jednostkach fonetycznych. Umożliwiłoby to rozpoznawanie komend ze zbioru otwartego co jest przeciwieństwem strategii zastosowanej w niniejszej pracy. Pociąga to za sobą konieczność nagrania dużej ilości próbek treningowych aby umożliwić poprawne wytrenowanie modeli HMM. Niestety skuteczność rozpoznawania w takim systemie, byłaby zapewne trochę niższa od osiągniętej w niniejszej pracy.

Kolejnym interesującym kierunkiem rozwoju, byłoby stworzenie systemu rozpoznawania mowy niezależnego od mówcy. Wymagałoby to wytrenowania systemu na nagraniach pochodzących od różnych osób, których głosy mają odmienną tonację (dzieci, kobiet, osób starszych). Brak dostępu do bazy nagrani języka polskiego znacznie utrudnia to zadanie, bo stworzenie takowej byłoby bardzo czasochłonne oraz mogłoby być osobnym projektem, niezwiązanym z niniejszą pracą. Taka baza byłaby również przydatna do poprawnego wytrenowania systemu opartego na fonemach. Pierwszym pomysłem jaki nasunął się autorowi, umożliwiającym pokonanie tej przeciwności, byłoby skorzystanie z darmowych nagrani znalezionych np. w Internecie, lub wykorzystanie książki w wersji audiobook i za pomocą takich nagrani możnaby wytrenować system rozpoznawania mowy. Kolejną ideą związaną z tworzeniem systemu niezależnego od mówcy jest rozwinięcie aplikacji o możliwość automatycznej adaptacji do mówcy. Polegałoby to na poinstruowaniu użytkownika, który po raz pierwszy korzysta z systemu oraz nagraniu jego głosu podczas ustalonych wypowiedzi, następnie ponownym wytrenowaniem systemu na próbkach nowego głosu. Taka czynność byłaby uruchamiana tylko podczas pierwszego korzystania z systemu oraz mogłaby być opcjonalna, uruchamiana jedynie w przypadku niskiej skuteczności rozpoznawania dla danego głosu w celu jej poprawienia.

W dobie coraz większej popularności urządzeń mobilnych takich jak smartfony lub tablety, działających na systemie Android, który posiada ponad połowę światowego rynku urządzeń mobilnych powinnym wątem uwagi wydaje się stworzenie wersji mobilnej aplikacji. Biliteki HTK są niezbędne do poprawnego działania aplikacji stworzonej na potrzeby niniejszej pracy. Zatem przeniesienie wszystkich funkcjonalności na system Android wiąże się ze skompilowaniem bibliotek HTK na tym systemie. W razie problemów z uruchomieniem HTK w systemie Android, alternatywą pozostaje skorzystanie z urządzenia Raspberry Pi jako serwera na którym odbywa się rozpoznawanie mowy za pomocą HTK. Aplikacja działająca na urządzeniu mobilnym z systemem Android podłączonym do sieci, nagrywałaby

wypowiedzi użytkownika, wysyłała do serwera skąd otrzymywałaby wyniki rozpoznawania a następnie odpowiadała za rozgrywkę oraz sterowanie robotem.

Podsumowując, systemy rozpoznawania mowy dają duże spektrum możliwości rozwoju oraz w opini autorów, mogą zostać ciekawym oraz przyszłościowym obiektem zainteresowań.

Bibliografia

- [1] *Alfabet fonetyczny ICAO*. http://pl.wikipedia.org/wiki/Alfabet_fonetyczny_ICAO.
- [2] *AS language reference manual*. <http://pl.scribd.com/doc/92216565/D-Controller-As-Language-Reference-Manual-deb>.
- [3] *CMUSphinx*. <http://cmusphinx.sourceforge.net>.
- [4] *Dokumentacja protokołu UCI*. <http://download.shredderchess.com/div/uci.zip>.
- [5] *Forsyth-Edwards Notation*. http://en.wikipedia.org/wiki/Forsyth-Edwards_Notation.
- [6] *HTK (Hidden Model Markov Toolkit)*. <http://htk.eng.cam.ac.uk/>.
- [7] *Julius*. http://julius.sourceforge.jp/en_index.php.
- [8] *Kawasaki FS03N Robot with D76 Controller*. <http://www.kawasakirobot.com/PDFs/FS03REV06-06.pdf>.
- [9] *MagicScribe*. <http://www.mAGICSCRIBE.pl>.
- [10] *Primespeech*. <http://www.primespeech.pl>.
- [11] *Raspberry Pi Foundation*. <http://www.raspberrypi.org/>.
- [12] *SARMATA*. <http://www.dsp.agh.edu.pl/doku.php?id=pl:resources:asr>.
- [13] *Schemat Raspberry Pi*. <http://www.raspberrypi.org/archives/382>.
- [14] *SoX - Sound eXchange*. <http://sox.sourceforge.net/>.
- [15] *Syntezator mowy Ivona*. <http://www.ivona.com/pl/>.
- [16] *WAVE PCM soundfile format*. <https://ccrma.stanford.edu/courses/422/projects/WaveFormat/>.
- [17] *Zasady gry w szachy*. http://pl.wikipedia.org/wiki/Zasady_gry_w_szachy.

- [18] Ostaszewska D. *Fonetika i Fonologia współczesnego języka polskiego*. Wydawnictwo Naukowe PWN, Warszawa, 2000.
- [19] Apple Inc. *Siri*. <http://www.apple.com/ios/siri>.
- [20] Google Inc. *Google Mobile App*. <http://www.google.com/mobile/google-mobile-app>.
- [21] Google Inc. *Google Now*. <http://www.google.com/landing/now>.
- [22] Szostek K. *Rozpoznawanie mowy metodami niejawnych modeli Markowa*. PhD thesis, AGH Kraków, 2006.
<http://winntbg.bg.agh.edu.pl/rozprawy/9792/full9792.pdf>.
- [23] Gałka J. *Optymalizacja parametryzacji sygnału w aspekcie rozpoznawania mowy polskiej*. PhD thesis, AGH Kraków, 2008.
<http://winntbg.bg.agh.edu.pl/rozprawy2/10009/full10009.pdf>.
- [24] Kowalski A. B. Kasprzak W. Analiza sygnału mowy sterowana danymi dla rozpoznawania komend głosowych. *Postępy Robotyki, WKiŁ*, 2006. http://www.ia.pw.edu.pl/~wkasprza/PAP/kkr06_rkg.pdf.
- [25] Grad L. Obrazowa reprezentacja sygnału mowy. *Biuletyn Instytutu Automatyki i Robotyki WAT nr. 8*, 1997. http://www.ita.wat.edu.pl/~l.grad/sieci%20neuronowe/ekstrakcja_cech_mowy.pdf.
- [26] Microsoft. *TellMe*. <http://www.microsoft.com/en-us/Tellme>.
- [27] Nuance. *Dragon NaturallySpeaking*. <http://www.nuance.com/dragon/index.htm>.
- [28] Halicki P. Język as - duże możliwości programowania robotów kawasaki. *Biuletyn Automatyki nr 54*, 4/2007. <http://www.astor.com.pl/biuletyn/archiwum/rocznik-2007/nr-54/jazyk-as-duze-mozliwosci-programowania-robotow-kawasaki.html>.
- [29] Zieliński T. P. *Cyfrowe przetwarzanie sygnałów od teorii do zastosowań*. WKiŁ, Warszawa, 2005.
- [30] Wielgat R. Wykłady z przedmiotu Techniki Multimedialne. PWSZ Tarnów., 2012.
- [31] Juang B-I Rabiner L. *Fundamentals of Speech recognition*. Prentice Hall, Warszawa, 1993.
- [32] Young S. *The HTK Book*. Cambridge, 2006. <http://htk.eng.cam.ac.uk/docs/docs.shtml>.
- [33] Kasprzak W. *Rozpoznawanie obrazów i sygnałów mowy*. Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2009.

Data ostatniej aktualizacji adresów: 3.01.2013 r.

Spis rysunków

2.1	Ogólny podział systemów przetwarzania mowy	8
2.2	Schemat blokowy systemu rozpoznawania mowy	11
3.1	Warstwowy model języka [23]	13
3.2	Reprezentacja słowa <i>trzy</i> w dziedzinie czasu	14
3.3	Reprezentacja słowa <i>trzy</i> w dziedzinie częstotliwości [30]	16
3.4	Segment bezdźwięczny oraz dźwięczny słowa „ <i>trzy</i> ” [30]	16
3.5	Ekstrakcja cech na podstawie [30]	18
3.6	Bank filtrów w skali melowej na podstawie [30]	19
3.7	Bank filtrów w skali hercowej na podstawie [32]	20
4.1	Model Markowa 5-cio stanowy typu left-to-right	23
4.2	Trenowanie modeli Markowa na podstawie [30]	26
4.3	Model HMM dla ciszy	27
4.4	Model HMM dla krótkiej pauzy	27
4.5	Zapis jednofonowy na podstawie [30]	28
4.6	Zapis trójfonowy [30]	28
4.7	Reestymacja Bauma-Welcha [30]	29
4.8	Schemat blokowy rozpoznawania [30]	33
4.9	Przykładowy model HMM	34
4.10	Algorytm Viterbiego [30]	34
5.1	Organizacja bibliotek HTK na podstawie [32]	36
5.2	Enkodowanie/dekodowanie sygnału mowy na podstawie [32]	37
5.3	Cztery główne fazy budowy systemu rozpoznawania mowy z HTK na podstawie [32]	38
5.4	Architektura HTK na podstawie [32]	39
5.5	Schemat Raspberry Pi [13]	41
5.6	Osie robota	42
5.7	Pozycje startowe figur	44
6.1	Schemat blokowy opisujący działanie systemu	46

6.2 Proces trenowania na podstawie [32]	54
6.3 Gramatyka dla etapu pierwszego	56
6.4 Gramatyka dla etapu drugiego	57
6.5 Gramatyka dla etapu trzeciego	58
6.6 Gramatyka dla etapu czwartego	59
6.7 Gramatyka dla etapu piątego	60
6.8 Gramatyka dla etapu szóstego	60
6.9 Zdjęcie stanowiska robota	61
6.10 Uproszczony schemat sterowania robotem	62
6.11 Pozycja początkowa figur w grze Wilk i owce	65
A.1 Struktura folderu z plikami HTK	77

Spis tabelic

5.1	Specyfikacja urządzenia [11]	41
5.2	Parametry modelu FS03N [8]	42
6.1	Alfabet fonetyczny ICAO wykorzystywany w systemie - litery na podstawie [1]	48
6.2	Alfabet fonetyczny ICAO wykorzystywany w systemie - cyfry na podstawie [1]	49

Spis listingów

5.1	Przykład wywołania narzędzia <i>HFoo</i>	40
5.2	Przykład wywołania narzędzia HTK wraz z plikiem konfiguracyjnym	40
5.3	Zapis początkowych pozycji figur	44
5.4	Zapis pozycji figur po wykonaniu dwóch ruchów	44
6.2	Przykładowe wypowiedzi sterujące podczas rozgrywki z robotem	49
6.3	Przykład definicji gramatyki	49
6.1	Słownik systemu	50
6.4	Przykład etykiety w formacie MLF	51
6.5	Plik konfiguracyjny narzędzia <i>HCopy</i>	52
6.6	Przykładowy wynik rozpoznawania w formacie MLF	55
6.7	Przykładowy wynik narzędzia HResults	55
6.8	Format wypowiedzi określającej przemieszczenie figury na szachownicy	59
6.9	Program napisany w języku AS, służący do sterowania robotem podczas rozgrywki	62
6.10	Polecenie uruchamiające program wykorzystywany do rozgrywki	63
6.11	Polecenia ustawiające zmienne <i>source</i> oraz <i>target</i>	63
6.12	Polecenie uruchamiające analizę dostępnych ruchów	64
A.1	Słownik systemu	78
A.2	Gramatyka systemu	79
A.3	Przykład wywołania skryptu <i>main.sh</i>	79
A.4	Główny skrypt <i>main.sh</i> uruchamiający proces trenowania	80
A.5	Podgląd pliku <i>step1.sh</i>	82
A.6	Podgląd pliku <i>step2.sh</i>	82
A.7	Podgląd pliku <i>step3.sh</i>	82
A.8	Podgląd pliku <i>step4.sh</i>	83
A.9	Podgląd pliku <i>step5.sh</i>	83
A.10	Podgląd pliku <i>step6.sh</i>	84
A.11	Podgląd pliku <i>step7.sh</i>	84
A.12	Podgląd pliku <i>step8.sh</i>	85

A.13 Podgląd pliku <i>recognize_live.sh</i>	85
---	----

A. Instrukcja wykorzystania skryptów do zautomatyzowania pracy z HTK

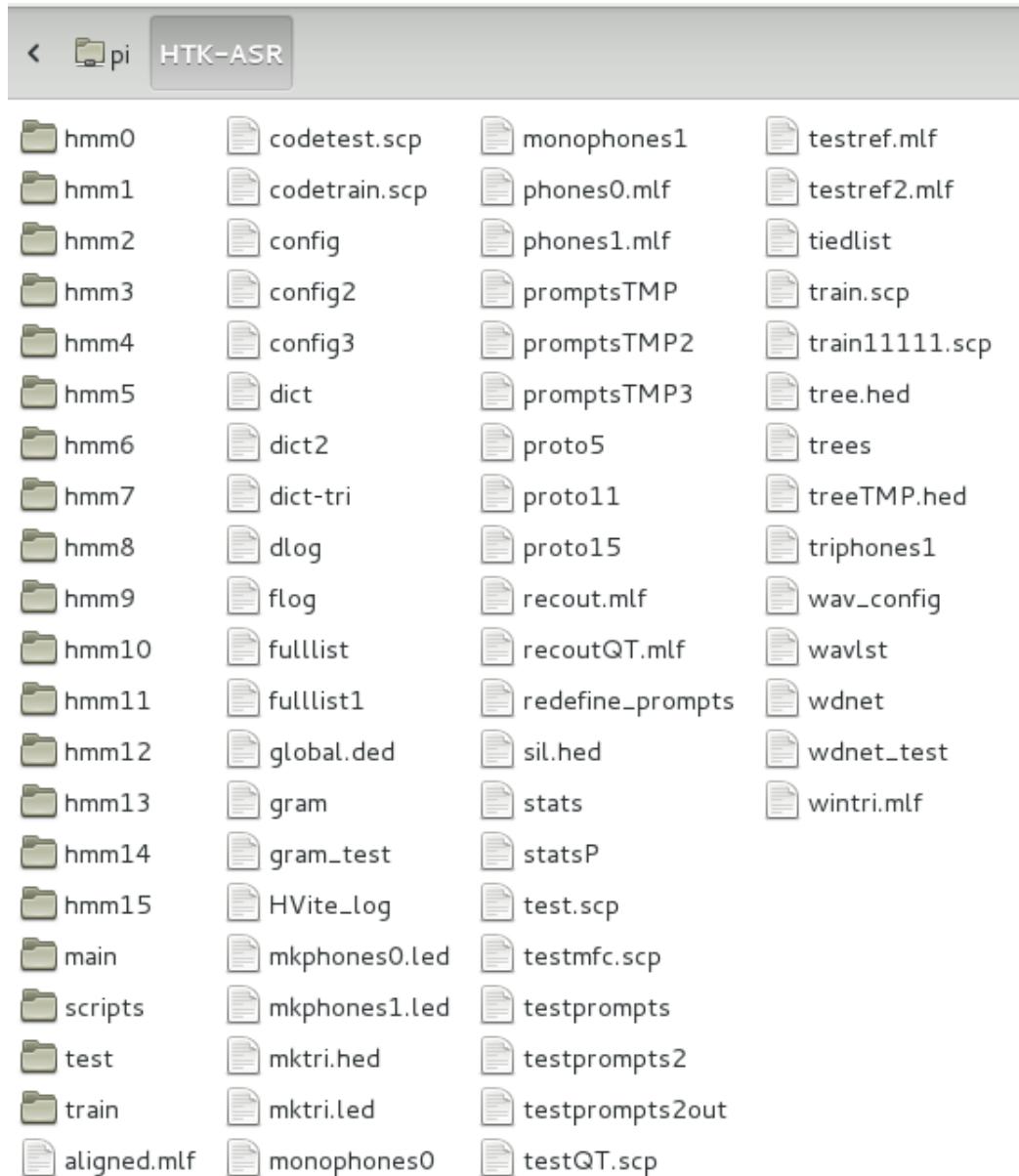
A.1. Wstęp

Praca z HTK wymaga często powtarzania tych samych żmudnych czynności, takich jak kopiowanie plików, tworzenie plików konfiguracyjnych lub plików zawierających ścieżki dostępu do innych plików. Testowanie skuteczności systemu rozpoznawania mowy oraz zmiana różnych jego parametrów wiąże się z ponownym wytrenowaniem tego systemu a więc ponownym przejęciem przez te same kroki. Chcąc zaoszczędzić czas potrzebny na powtarzanie tych samych czynności autor postanowił przygotować skrypty wykonujące różne funkcje z użyciem bibliotek HTK, parsujące pliki tekstowe, lub tworzące odpowiednie pliki konfiguracyjne wymagane do pracy z HTK. Skrypty te zostały napisane w języku Perl (na wzór gotowych skryptów z pakietu HTK) oraz w języku powłoki bash (powłoka systemowa UNIX), która jest rozszerzeniem powłoki sh.

Zostaną tutaj opisane najważniejsze skrypty oraz pliki pozwalające stworzyć silnik systemu rozpoznawania mowy.

Pakiet HTK jest dostępny pod adresem <http://htk.eng.cam.ac.uk/download.shtml> zarówno dla systemu Windows jak i Linux. Opis komplikacji bibliotek również dostępny jest w w/w źródle.

Struktura folderu, zawierającego pliki potrzebne do pracy z HTK podczas tworzenia systemu rozpoznawania mowy została pokazana na rysunku A.1.



Rysunek A.1: Struktura folderu z plikami HTK

Folder *hmm15* zawiera wytrenowane modele HMM, wykorzystywane podczas rozpoznawania. Z punktu widzenia rozpoznawania, istotne są również pliki zawierające gramatykę systemu (plik *gram*), sieć słów (plik *wdnet*) utworzoną na podstawie gramatyki oraz słownik systemu (plik *dict*), ścieżkę pliku dźwiękowego którego zawartość jest rozpoznawana (plik *testQT.scp*) oraz plik w którym zapisany jest wynik rozpoznawania (*recoutQT.mlf*). W folderze *train/wav* znajdują się próbki treningowe, natomiast folder *train/mfcc* zawiera pliki z wyekstrahowanymi cechami próbek treningowych. W folderze *scripts* znajdują się skrypty wykorzystywane w procesie trenowania systemu. Z kolei folder *main* zawiera główne skrypty przeprowadzające (automatyzujące) proces trenowania oraz uruchamiające rozpoznawanie.

A.2. Trenowanie oraz rozpoznawanie

Zgodnie z tym co zostało napisane w rozdziale 6 niniejszej pracy, jednym z pierwszych kroków podczas pracy nad systemem jest utworzenie słownika oraz gramatyki widocznych na listingach A.1 oraz A.2.

Listing A.1: Słownik systemu

ALFA	[ALFA]	alfa sp
BRAVO	[BRAVO]	bravo sp
CHARLIE	[CHARLIE]	charlie sp
CIEMNY	[CIEMNY]	ciemny sp
CISZA	[CISZA]	cisza sp
COFNIJ	[COFNIJ]	cofnij sp
CZTERY	[CZTERY]	cztery sp
DELTA	[DELTA]	delta sp
DWA	[DWA]	dwa sp
ECHO	[ECHO]	echo sp
FOXTROT	[FOXTROT]	foxtrot sp
GOLF	[GOLF]	golf sp
HOTEL	[HOTEL]	hotel sp
JAROZPOCZYNAM	[JAROZPOCZYNAM]	jarozoczynam sp
JASNY	[JASNY]	jasny sp
JEDYNKA	[JEDYNKA]	jedynka sp
JESTEM	[JESTEM]	jestem sp
KASZEL	[KASZEL]	kaszel sp
KONIEC	[KONIEC]	koniec sp
NA	[NA]	na sp
ODDECH	[ODDECH]	oddech sp
OSIEM	[OSIEM]	osiem sp
OWCAMI	[OWCAMI]	owcamy sp
PIATKA	[PIATKA]	piatka sp
PIOTR	[PIOTR]	piotr sp
RESET	[RESET]	reset sp
ROZPOCZNIJ	[ROZPOCZNIJ]	rozpocznij sp
SENT-END	[SENT-END]	sil
SENT-START	[SENT-START]	sil
SIEDEM	[SIEDEM]	siedem sp
SLUCHAWKI	[SLUCHAWKI]	słuchawki sp
START	[START]	start sp

SZESC	[SZESC]	szesc sp
TRZY	[TRZY]	trzy sp
TWOJRUCH	[TWOJRUCH]	twojruch sp
WILKIEM	[WILKIEM]	wilkiem sp
WITAJ	[WITAJ]	witaj sp
WSZACHY	[WSZACHY]	wszachy sp
WWILKIEWCE	[WWILKIEWCE]	wwilkiewce sp
ZAGRAJMY	[ZAGRAJMY]	zagrajmy sp
ZATRZYMAJ	[ZATRZYMAJ]	zatrzymaj sp

Skrót *sp* oznacza krótką pauzę, natomiast skrót *sil* ciszę.

Listing A.2: Gramatyka systemu

```
$number = JEDYNKA | DWA | TRZY | CZTERY | PIATKA | SZESC | SIEDEM |
OSIEM;
$field = ALFA | CHARLIE | BRAVO | DELTA | ECHO | FOXTROT | GOLF |
HOTEL;
$noise = ODDECH | CISZA | SLUCHAWKI | ODDECH; $word = ALFA | CHARLIE |
BRAVO | DELTA | ECHO | FOXTROT | GOLF | HOTEL | JEDYNKA | DWA | TRZY |
CZTERY | PIATKA | SZESC | SIEDEM | OSIEM | START | KONIEC | RESET |
ZATRZYMAJ | COFNIJ | TWOJRUCH | WILKIEM | OWCAMI | JASNY | CIEMNY |
ROZPOCZNIJ | JAROZPOCZYNAM | WITAJ | JESTEM | ZAGRAJMY | WSZACHY |
WWILKIEWCE | PIOTR;

( SENT-START (( $field $number NA $field $number) | $word | $word
$word | $word $word $word | $word $noise) SENT-END )
```

Gdy pliki zawierające słownik oraz gramatykę są już gotowe, potrzeba nagrać próbki treningowe zgodne z gramatyką systemu, zawierające wielokrotne powtórzenia wszystkich słów ze słownika.

Do poprawnego zadziaływania skryptu (listing A.4) przeprowadzającego trenowanie systemu, kolejne próbki dźwiękowe zawierające wypowiedzi treningowe muszą zostać nazwane wg. odpowiedniej konwencji. Dla pierwszej próbki będzie to *sample1.wav*, w nazwach każdej kolejnej próbki zmienia się tylko jej numer, tzn. *sample2.wav*, *sample3.wav*, aż do *sampleN.wav*, gdzie *N* jest to liczba wszystkich próbek treningowych. Musi być również zachowana struktura folderów widoczna na rysunku A.1. Na listingu A.3 pokazano przykład wywołania skryptu *main.sh*.

Listing A.3: Przykład wywołania skryptu *main.sh*

```
sh main/main.sh 673 15 > main/main_log
```

Listing A.4: Główny skrypt *main.sh* uruchamiający proces trenowania

```
#!/bin/bash
#$1 - number of prompt files
#$2 - number of states
#$3 - how many steps we omit counting from the first

ARGC="$#"
if [ $ARGC -eq 2 ]; then
    NOSTEP="0"
else
    NOSTEP=$3
fi
if [ $NOSTEP -eq 0 ]; then
    sh main/step1.sh
    echo -----
    echo " ----- STEP1 ----- "
    echo -----
fi
if [ $NOSTEP -eq 0 ] || [ $NOSTEP -eq 1 ]; then
    sh main/step2.sh
    echo -----
    echo " ----- STEP2 ----- "
    echo -----
fi
if [ $NOSTEP -eq 0 ] || [ $NOSTEP -eq 1 ] || [ $NOSTEP -eq 2 ]; then
    sh main/step3.sh $1
    echo -----
    echo " ----- STEP3 ----- "
    echo -----
fi
if [ $NOSTEP -eq 0 ] || [ $NOSTEP -eq 1 ] || [ $NOSTEP -eq 2 ] ||
[ $NOSTEP -eq 3 ]; then
    sh main/step4.sh proto$2 $1
    echo -----
    echo " ----- STEP4 ----- "
    echo -----
fi
if [ $NOSTEP -eq 0 ] || [ $NOSTEP -eq 1 ] || [ $NOSTEP -eq 2 ] ||
[ $NOSTEP -eq 3 ] || [ $NOSTEP -eq 4 ]; then
    sh main/step5.sh $2
```

```

echo -----
echo " ----- STEP5 ----- "
echo -----
fi
if [ $NOSTEP -eq 0 ] || [ $NOSTEP -eq 1 ] || [ $NOSTEP -eq 2 ] ||
[ $NOSTEP -eq 3 ] || [ $NOSTEP -eq 4 ] || [ $NOSTEP -eq 5 ]; then
    sh main/step6.sh
    echo -----
    echo " ----- STEP6 ----- "
    echo -----
fi
if [ $NOSTEP -eq 0 ] || [ $NOSTEP -eq 1 ] || [ $NOSTEP -eq 2 ] ||
[ $NOSTEP -eq 3 ] || [ $NOSTEP -eq 4 ] || [ $NOSTEP -eq 5 ] ||
[ $NOSTEP -eq 6 ]; then
    sh main/step7.sh
    echo -----
    echo " ----- STEP7 ----- "
    echo -----
fi
if [ $NOSTEP -eq 0 ] || [ $NOSTEP -eq 1 ] || [ $NOSTEP -eq 2 ] ||
[ $NOSTEP -eq 3 ] || [ $NOSTEP -eq 4 ] || [ $NOSTEP -eq 5 ] ||
[ $NOSTEP -eq 6 ] || [ $NOSTEP -eq 7 ]; then
    sh main/step8.sh
    echo -----
    echo " ----- STEP8 ----- "
    echo -----
fi
#cp train/wav/* test/
#perl scripts/prompts2mlf testref2.mlf test/prompts
#perl scripts/fixtestref.pl testref2.mlf testref.mlf
#perl scripts/create_test.pl test.scp $1
#sh main/recognize_tests.sh
#perl scripts/create_stats.pl train/wav/wlist recout.mlf testref.mlf
statsP
#cat statsP
#leafpad main/main_log

```

Skrypt *main.sh* jako dwa główne argumenty przyjmuje ilość wszystkich próbek treningowych zawartą w pliku *prompts* znajdującym się w folderze *train/wav* oraz ilość stanów modelu HMM.

Zakomentowane linie na końcu skryptu (z użyciem znaku #) służą do przeprowadzenia testu skuteczności systemu na próbkach wykorzystanych do jego wytrenowania.

W pliku *main_log* zapisane są wyniki lub błędy działania niektórych funkcji oraz skryptów wykorzystanych w procesie trenowania.

Proces trenowania systemu został podzielony na osiem kroków, co odpowiada ośmiu skryptom wywoływanym z głównego *main.sh*.

W kroku pierwszym (listing A.5) zostaje utworzona sieć słów na podstawie gramatyki systemu a następnie przygotowywany jest plik (*words.mlf*) zawierający etykiety próbek treningowych w formacie MLF oraz plik (*wlist*) zawierający listę wszystkich słów zawierających się w tych etykietach (bez powtórzeń). Lista ta musi się zgadzać ze słownikiem systemu.

Listing A.5: Podgląd pliku *step1.sh*

```
#!/bin/bash

HParse gram wdnet
perl scripts/prompts2mlf train/wav/words.mlf train/wav/prompts
perl scripts/prompts2wlist train/wav/prompts train/wav/wlist
perl scripts/fixwlist.pl train/wav/wlist
```

W kroku drugim (listing A.6) zostają utworzone pliki (*monophones0*, *monophones1*) zawierające wszystkie słowa użyte w systemie (jeśli system opierałby się na fonemach byłoby to pojedyncze fonemy, tzw. jednofony).

Listing A.6: Podgląd pliku *step2.sh*

```
#!/bin/bash

HDMan -A -D -T 1 -m -w train/wav/wlist -n monophones1 -i -l dlog dict
dict2
#perl scripts/remove_double_sp.pl dict2 dict
perl scripts/create_monophones0.pl
```

Wynikiem kroku trzeciego (listing A.7) jest plik zawierający ścieżki do próbek treningowych oraz pliki MFCC zawierające wykstrahowane cechy próbek treningowych.

Listing A.7: Podgląd pliku *step3.sh*

```
#!/bin/bash
```

```
HLED -A -D -T 1 -l '*' -d dict -i phones0.mlf mkphones0.led
train/wav/words.mlf
HLED -A -D -T 1 -l '*' -d dict -i phones1.mlf mkphones1.led
train/wav/words.mlf
perl scripts/create_codetrain.pl codetrain.scp $1
HCopy -A -D -T 1 -C wav_config -S codetrain.scp
```

W kolejnym kroku tj. czwartym (listing A.8) wygenerowane zostają prototypy modeli Markowa a następnie rozpoczyna się ich reestymacja.

Listing A.8: Podgląd pliku *step4.sh*

```
#!/bin/bash

perl scripts/create_train.pl train.scp $2
HCompV -A -D -T 1 -C config -f 0.01 -m -S train.scp -M hmm0 $1
perl scripts/create_hmmdefs.pl monophones0 $1 hmm0/hmmdefs
perl scripts/create_macros.pl hmm0/$1 hmm0/vFloors hmm0/macros
HERest -A -D -T 1 -C config -I phones0.mlf -t 250.0 150.0 1000.0
-S train.scp -H hmm0/macros -H hmm0/hmmdefs -M hmm1 monophones0
HERest -A -D -T 1 -C config -I phones0.mlf -t 250.0 150.0 1000.0
-S train.scp -H hmm1/macros -H hmm1/hmmdefs -M hmm2 monophones0
HERest -A -D -T 1 -C config -I phones0.mlf -t 250.0 150.0 1000.0
-S train.scp -H hmm2/macros -H hmm2/hmmdefs -M hmm3 monophones0
```

Podczas kroku piątego (listing A.9) na podstawie modelu ciszy (*sil*) utworzony zostaje model krótkiej pauzy (*sp*) oraz kontynuowana jest reestymacja.

Listing A.9: Podgląd pliku *step5.sh*

```
#!/bin/bash

perl scripts/create_hmm4.pl hmm3/macros hmm4/macros hmm3/hmmdefs
hmm4/hmmdefs $1
HHED -A -D -T 1 -H hmm4/macros -H hmm4/hmmdefs -M hmm5 sil.hed
monophones1
HERest -A -D -T 1 -C config -I phones1.mlf -t 250.0 150.0 3000.0
-S train.scp -H hmm5/macros -H hmm5/hmmdefs -M hmm6 monophones1
```

```
HERest -A -D -T 1 -C config -I phones1.mlf -t 250.0 150.0 3000.0
-S train.scp -H hmm6/macros -H hmm6/hmmdefs -M hmm7 monophones1
```

W kroku szóstym (listing A.10) oraz siódmym (listing A.11) przebiega kolejna reestymacja parametrów modeli HMM.

Listing A.10: Podgląd pliku *step6.sh*

```
#!/bin/bash

HVite -A -D -T 1 -l '*' -o SWT -b SENT-END -C config
-H hmm7/macros -H hmm7/hmmdefs -i aligned.mlf
-m -t 250.0 150.0 1000.0 -y lab -a -I train/wav/words.mlf
-S train.scp dict monophones1> HVite_log
HERest -A -D -T 1 -C config -I aligned.mlf
-t 250.0 150.0 3000.0 -S train.scp -H hmm7/macros
-H hmm7/hmmdefs -M hmm8 monophones1
HERest -A -D -T 1 -C config -I aligned.mlf
-t 250.0 150.0 3000.0 -S train.scp -H hmm8/macros
-H hmm8/hmmdefs -M hmm9 monophones1
```

Listing A.11: Podgląd pliku *step7.sh*

```
#!/bin/bash

HLED -A -D -T 1 -n triphones1 -l '*' -i wintri.mlf mktri.led
aligned.mlf
perl scripts/maketrihed monophones1 triphones1
HHEd -A -D -T 1 -H hmm9/macros -H hmm9/hmmdefs -M hmm10 mktri.hed
monophones1
HERest -A -D -T 1 -C config -I wintri.mlf -t 250.0 150.0 3000.0
-S train.scp -H hmm10/macros -H hmm10/hmmdefs -M hmm11 triphones1
HERest -A -D -T 1 -C config -I wintri.mlf -t 250.0 150.0 3000.0
-s stats -S train.scp -H hmm11/macros -H hmm11/hmmdefs
-M hmm12 triphones1
```

Podczas ostatniego króku, tj. ósmego (listing A.12), zostaje zakończony proces reestymacji a otrzymane modele HMM (folder *hmm15*) tworzą model akustyczny wykorzystywany przez narzędzie *HVite* na etapie rozpoznawania.

Listing A.12: Podgląd pliku *step8.sh*

```
#!/bin/bash

HDMAn -A -D -T 1 -b sp -n fulllist -g global.ded -l flog dict-tri
dict
perl scripts/remove_double_sp.pl dict-tri dict-tri2
mv dict-tri2 dict-tri
perl scripts/create_treeHed_part1.pl treeTMP.hed tree.hed
perl scripts/mkclsript.prl TB 350 monophones0 >> tree.hed
perl scripts/create_treeHed_part2.pl tree.hed
HHEd -A -D -T 1 -H hmm12/macros -H hmm12/hmmdefs -M hmm13 tree.hed
triphones1
HERest -A -D -T 1 -T 1 -C config -I wintri.mlf -s stats
-t 250.0 150.0 3000.0 -S train.scp -H hmm13/macros
-H hmm13/hmmdefs -M hmm14 tiedlist
HERest -A -D -T 1 -T 1 -C config -I wintri.mlf -s stats
-t 250.0 150.0 3000.0 -S train.scp -H hmm14/macros
-H hmm14/hmmdefs -M hmm15 tiedlist
```

Po pomyślnym wykonaniu wszystkich ośmiu skryptów skrypt *recognize_live.sh* (listing A.13) pozwala przetestować skuteczność rozpoznawania. Można go uruchomić w konsoli polecienniem *sh recognize_live.sh*, po uruchomieniu nagrywana jest wypowiedź użytkownika a po wykryciu ciszy trwającej dłużej niż jedna sekunda uruchamiane jest rozpoznawanie z użyciem narzędzia *HVite*, którego wyniki zapisywane są do pliku *recoutQT.mlf* oraz wyświetlane w konsoli.

Listing A.13: Podgląd pliku *recognize_live.sh*

```
#!/bin/bash

sh main/record.sh
HVite -C config3 -H hmm15/macros -H hmm15/hmmdefs -S testQT.scp
-l '*' -i recoutQT.mlf -w wdnet -p 0.0 -s 5.0 dict tiedlist
cat recoutQT.mlf
```

B. Zawartość płyty

Na płycie dołączonej do pracy znajdują się następujące elementy:

- *praca.pdf* - wersja elektroniczna niniejszej pracy,
- *raspbian.iso* - obraz zawierający kopię systemu Raspbian z urządzenia Raspberry Pi wraz z bibliotekami HTK oraz wszystkimi plikami niezbędnymi do poprawnego działania systemu rozpoznawania mowy,
- *ASR-PY* - folder zawierający skrypt głównej aplikacji,
- *HTK-ASR* - folder w którym znajduje się silnik rozpoznawania mowy, próbki treningowe oraz pozostałe pliki potrzebne do pracy z HTK,
- *Literatura* - folder zawierający wersje elektroniczne niektórych pozycji wymienionych w bibliografii.