System rozpoznawania mowy z ograniczonym słownikiem

Speech recognition system with limited dictionary

Dawid Grabowski[1], Monika Kwiatkowska[2] i Łukasz Świerczewski[1]

Treść. Motywacją w pisanej pracy jest omówienie i porównanie popularnych algorytmów rozpoznawania mowy na różnych systemach. Zebrane informacje są przedstawione w stosunkowo krótkiej formie, bez wnikliwej analizy dowodów matematycznych, do których przedstawienia i tak potrzebne jest odniesienie się do odrębnych specjalistycznych źródeł. Omówione zostały tutaj problemy pewne związane z ASR (ang. Automatic Speech Recognition) i perspektywy na rozwiązanie ich. Na podstawie dostępnych rozwiązań stworzony został moduł aplikacji umożliwiający porównywanie zebranych nagrań pod kątem podobieństwa sygnału mowy i przedstawienie wyników w formie tabelarycznej. Stworzona biblioteka w celach prezentacyjnych została użyta do pełnej aplikacji umożliwiającej wykonywanie rozkazów na podstawie słów wypowiadanych do mikrofonu. Wyniki posłużą nie tyle za ostateczne wnioski w tematyce rozpoznawania mowy, co za wskazówki do kolejnych analiz i badań. Mimo postępów w badaniach nad ASR, nadal nie ma algorytmów o skuteczności przekraczającej 95%. Motywacją do dalszych działań może być np. społeczne wykluczenie ludzi nie mogących posługiwać się komunikacją polegającą na wzroku.

Słowa kluczowe: Rozpoznawanie mowy, ASR, MFCC.

Abstract. Motivation of this thesis is discussion about popular ASR algorithms and comparision on various architectures. Collected results are presented in relatively short shape. It's done without math argumentation because it could depend on complicated equations. Here are discussed some problems associated with ASR (Automatic Speech Recognition) and the prospects for a solution to their. On the basis of available solutions it was developed application module that allows comparison of collected recordings in respect of similarity of the speech signal and present the results in tabular form. For presentation purposes it has been created a library and it was used in complete application that allows execution of commands based on the words spoken to microphone. The results will be used not only for the final conclusions about ASR, what clues for further analysis and research. Despite the advances in research on ASR, still there are no algorithms for effectiveness in excess of 95%. The motivation for further actions may be, eg, the social exclusion of people who can not use the communication involving the eye.

Keywords: speech recognition, ASR, MFCC.

1. Wstęp

Celem rozwoju technologii rozpoznawania mowy jest umożliwienie człowiekowi komunikacji z maszyną przy pomocy mowy. W ostatnich latach nastąpił postęp rozwoju dziedziny przetwarzania mowy, między innymi za sprawą dostępnych zasobów w postaci słowników. To spowodowało, że ASR (ang Automatic Speech Recognition) stało się jedną z ważniejszych dziedzin DSP (ang. Digital Sound Processing). Mowa posiada pewne charakterystyczne cechy. Jedną z nich było zawieranie się częstotliwości sygnału mowy w pewnym spektrum, ograniczonym w stosunku do słyszalnego. Pozwoliło to na opracowanie algorytmów kompresji danych specyficznych dla konkretnych dźwięków.

Okazuje się, że optymistycznie zapowiadający się postęp cyfryzacji zaczął ograniczać np. niewidomych. Rozpowszechnienie komputerów i innych wszelkiego rodzaju urządzeń elektronicznych a także wzrost ich mocy obliczeniowej [9, 12] otworzyły nowe perspektywy dla systemów rozpoznawania mowy, które pomagają np. osobom, które mają bardzo utrudnione wprowadzanie danych do systemu.

Choć niektóre algorytmy, np. ukryte modele Markowa pozostały niemal bez zmian, stanowią bazę dla innych, wykorzystujących bardziej wyrafinowane metody postępowania. Bardzo ważnym elementem projektowania systemów automatycznego rozpoznawania mowy jest

1. Państwowa Wyższa Szkoła Informatyki i Przedsiębiorczości, Instytut Automatyki i Robotyki, ul. Akademicka 14, 18-400 Łomża
2. Uniwersytet Marii Curie-Skłodowskiej w Lublinie, Wydział Matematyki, Fizyki i Informatyki, pl. Marii Curie-Skłodowskiej 5, 20-031 Lublin

złożoność obliczeniowa. Projektowanie algorytmu stawia wybór pomiędzy dokładnością a szybkością. Sieci neuronowe i technologie typu GPGPU (*ang General-Purpose computing on Graphics Processing Units*) [8], pozwalające programować szybkie procesory graficzne, wydają się przybliżać możliwość posiadania dokładności i szybkości w jednej aplikacji.

Głos różny dla każdego człowieka, może być jednym z elementów systemów zabezpieczeń. Ponieważ te systemy polegają na zezwalaniu dostępu co najwyżej kilku osobom, nie ma potrzeby weryfikacji danych wejściowych pod względem innych osób. To znacząco obniża koszt czasowy i pamięciowy. Podobnymi, prostszymi mechanizmami dysponuje większość telefonów komórkowych posiadających opcję wybierania głosowego [11].

Powyższe rozwiązania nie są pozbawione wad. Sekwencje słów wprowadzone jako klucze w systemach zabezpieczeń mogą być ciężkie do zapamiętania i do powtórzenia. Do tego należy wziąć pod uwagę nastrój osoby w danym momencie, albo choroby związane z układem artykulacji (np. chrypka). Z jednej strony algorytmy rozpoznawania mowy są projektowane tak, aby radzić sobie z tego typu rozbieżnościami, z drugiej jednak architekci systemów zabezpieczeń nie mogą pozwolić sobie na zbytnią dowolność, gdyż to umożliwiało by włamania.

1.1. Cel pracy

Motywacją w pisanej pracy jest omówienie ważniejszych aspektów z dziedziny automatycznego rozpoznawania mowy i przedstawienie popularnych algorytmów. Zebrane informacje są przedstawione w stosunkowo krótkiej formie, bez wnikliwej analizy dowodów matematycznych, do których przedstawienia i tak potrzebne jest odniesienie się do odrębnych specjalistycznych źródeł. Omówione zostały tutaj problemy pewne związane z ASR i perspektywy na rozwiązanie ich.

Na podstawie rozpatrywanych rozwiązań i algorytmów zostanie zrealizowany system automatycznego rozpoznawania mowy z ograniczonym słownikiem oraz aplikacja graficzna wykorzystująca mechanizmy rozpoznawania mowy do sterowania. Następnie zostanie dokonany krótki test i zebranie wyników w formie tabelarycznej.

2. Charakterystyka języka mówionego

Nie ulega wątpliwości, iż mowa jest czymś bardziej naturalnym dla człowieka, niż dla maszyny. Dlatego aby umożliwić interpretację głosu z zadowalającym efektem przez maszynę, należy zrozumieć działanie narządu głosu (mowy i słychu) człowieka, następnie wiedzę tę należy usystematyzować. W tym rozdziale przedstawiono podstawowe informacje niezbędne podczas tworzenia interfejsu umożliwiającego komunikację człowieka z komputerem.

2.1. Dźwięk

Dźwięk jest skutkiem drgań powietrza o charakterze falowym. Innymi słowy to seria określonych zmian ciśnienia powietrza [1, 15]. Charakteryzuje je kilka cech:

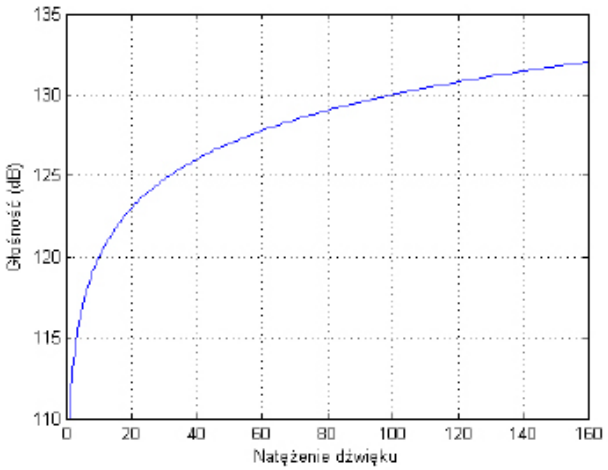
- amplituda – wychylenie z położenia równowagi – ciśnienie akustyczne,
- długość fali – czas trwania pojedynczego wychylenia,
- częstotliwość – ilość drgań w ciągu jednej sekundy.

Długość fali jest mierzona w ułamkach sekund i odbierana niezależnie od obserwatora. Pozostałe cechy mają pewne różnice w pomiarze i odbiorze przez człowieka. Zarówno amplituda jak i częstotliwość mierzone są w skali liniowej, w przeciwieństwie do subiektywnej skali odbiorcy. Ciśnienie akustyczne odczuwalne jest jako głośność i wyraża się ja w belach (decybelach). Nie każda zmiana amplitudy powoduje wyczuwalną zmianę głośności. Odbywa się to w skali logarytmicznej zgodnie z prawem Webera-Fechnera – różnice przy małej głośności są wyraźnie wyczuwalne, a różnice przy dużej głośności są praktycznie nierozróżnialne [1, 15]. Zależność tę wyraża się wzorem:

$$dB_x = 10\log_{10}\left(\frac{x}{10^{-12}}\right) W/m^2 \tag{1}$$

gdzie: *x – częstotliwość w Hz, W – moc wyrażona w Watt, m - metr*

Reprezentuje ją Ryc 1.



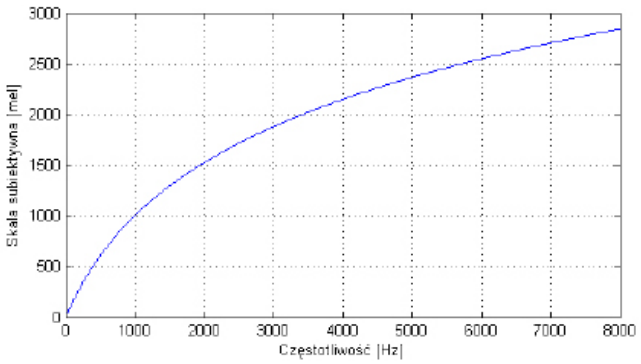
Ryc. 1 Zależność głośności w skali dB między natężeniem dźwięku.
Fig. 1. *The dependence between the volume in dB scale and the intensity of a sound.*

Częstotliwość odbierana przez ludzkie ucho wyrażana jest w skali mel (*ang mel- scale*). Polega ona na zależności między częstotliwością czystego tonu harmonicznego i częstotliwością postrzeganą przez człowieka. Jest ona wyrażana w skali mel. Zależność między Hz i mel jest nieliniowa [1,15].

$$f_{mel}(f_{Hz}) = 2595\log_{10}\left(1+\frac{f_{Hz}}{700}\right)$$
$$f_{Hz}(f_{mel}) = 700\left(10^{\left(\frac{f_{mel}}{2595}\right)}-1\right) \tag{2}$$

gdzie: f_{mel} – *częstotliwość w skali mel,* f_{Hz} – *częstotliwość w* Hz

Według wyników doświadczeń, nieliniowe przetwarzanie częstotliwości zwiększa skuteczność systemów automatycznego rozpoznawania mowy. Powyższy związek między obiema skalami reprezentuje Ryc. 2 [1].



Ryc. 2. Zależność skali wyrażonej w dB i skali mel.
Fig. 2. *The dependece in dB scale and mel scale.*

2.2. Narząd mowy i fonetyka

Mowa w ogólnym pojęciu oznacza usystematyzowane dźwięki wydawane przez narząd mowy u człowieka. Powstawanie tych dźwięków polega na modyfikacji przepływu wydychanego powietrza za pomocą ust, języka i nosa. Zestawienie samogłoski i spółgłoski stanowi sylabę i tym samym część słowa.

2.3. Narząd słuchu i percepcja mowy

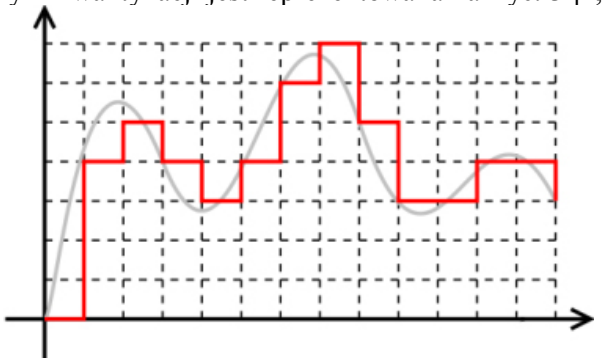
Ucho ludzkie odpowiada za odbieranie dźwięków z otoczenia oraz za utrzymanie równowagi. Przekształca fale dźwiękowe na impulsy nerwowe. Składa się z trzech sekcji: ucha zewnętrznego, środkowego i wewnętrznego. Przekształcenie fal dźwiękowych na sygnał elektryczny dzisiaj nie stanowi większego problemu dla projektujących wszelką elektronikę. Zdecydowanie najpoważniejszym wyzwaniem jest ustalenie kontekstu dźwięku (np. odróżnienie wypowiedzi zależnie od intonacji, dialektyzacja). To, co wydaje się siłą algorytmów implementowanych w programach komputerowych, a mianowicie pewna „odporność" na zmiany dźwięku, staje się też ich wadą. Jeśli przetwarza się słowo jako całość, należy wziąć pod uwagę jego warianty. Dobrym rozwiązaniem wydaje się nadanie sensu dla każdej możliwej wymowy słowa i zapamiętanie. To jednak stwarza kolejne problemy związane z pamięcią [8, 10, 11].
Praktycznym podejściem do problemu jest interpretacja mowy jako ciąg głosek. Mając do dyspozycji difony, czyli przejścia pomiędzy sąsiadującymi głoskami, można próbować dopasowania do kontekstu. Najczęściej wypowiedź traktuje się jako serię difonów, czyli stan przejść w trakcie wymowy tak, jak ma to miejsce w przypadku rozróżniania kontekstu przez człowieka [8, 10, 11].

3. Cyfrowe przetwarzanie sygnałów

Cyfrowe przetwarzanie sygnałów stanowi dziedzinę nauki zajmującą się sygnałami cyfrowymi i operacjami na nich. Ważnymi zastosowaniami DSP (ang. Digital Sound Processing) są: kompresja dźwięku, kodowanie obrazu, telekomunikacja, rozpoznawanie mowy. Pierwszym etapem działania każdego układu DSP jest przetworzenie sygnału analogowego na cyfrowy – część za to odpowiadającą nazywa się przetwornikiem analogowo-cyfrowym. Od lat inżynierzy wspomnianych układów zmagają się z tym samym problemem: im lepszy układ, tym bardziej skomplikowane wykonanie [1, 2, 3, 8, 10, 11].

3.1. Reprezentacja dźwięku w komputerze

Karty dźwiękowe stały się nieodłącznym elementem każdej płyty głównej. Zintegrowany układ SPU (ang. Sound Processing Unit) pozwala na podstawowe operacje: przechwytywanie i odtwarzanie dźwięku z przyzwoitą jakością. Dźwięk może zostać przechwycony różnymi urządzeniami peryferyjnymi, jednak aby móc przechowywać sygnał w komputerze, wszystkie przekształcają go na postać cyfrową. Operacja konwersji sygnału analogowego na cyfrowy wiąże się z nieodwracalną utratą informacji. Próbkowanie danych polega na tym, że z sygnału wejściowego pobierane są próbki w stałym okresie czasu. Im częściej (gęściej) ułożone, mniejsze przerwy w nich występują, tym mniej danych zostało utracone. Jednostką pomiarową dla próbkowania jest herc. Im wyższa jest częstotliwość próbkowania tym wyższa jest maksymalna częstotliwość dźwięku jaką można zapisać. Maksymalna częstotliwość dźwięku jaką można odtworzyć bez zniekształceń z danych na jedynym kanale to 22050 Hz [1, 19].
Kwantyzacja określa skończony zbiór wartości dla pojedynczej próbki (precyzji zmian głośności). Kwantyzacja jest również informacją o maksymalnym możliwym wychyleniu amplitudy. Parametr ten wyraża się w bitach, co ma proste przełożenie na podstawowe typy danych w komputerze, np. 64 dla typu zmiennopozycyjnego podwójnej precyzji. Różnica pomiędzy sygnałem rzeczywistym a poddanym kwantyzacji jest reprezentowana na Ryc. 3 [1, 19]



Ryc. 3 Porównanie wykresu sygnału wejściowego i odpowiadającego mu sygnału dyskretnego.
Fig. 3. *The comparison of the input signal and the corresponding discrete signal.*

Na czerwono zaznaczono sygnał dyskretny. Siatka złożona z przerywanych linii oznacza kwantyzację (linie poziome) i próbkowanie (linie pionowe). Ponieważ kwantyzacja i próbkowanie - jest wartością dyskretną, powoduje bezpowrotną utratę części informacji i błędy przy próbach przywrócenia sygnału analogowego. Sama definicja sygnału dyskretnego sugeruje, iż sygnał dyskretny nie będzie idealnie oddawać sygnału analogowego. Podczas przetwarzania cyfrowego sygnału można zniwelować błędy kwantyzacji np. stosując dithering. W przypadku ilości bitów przypadającej na próbkę większej niż 16 bitów, błędy są niesłyszalne dla ludzkiego ucha. Dlatego warto stosować maksymalną możliwą wartość kwantyzacji. Dostosowując wyżej wymienione parametry można „oszukać" zmysł słuchu [1, 15]. Czasami zachodzi potrzeba zmiany parametrów dźwięku, które nie są jawnie widoczne w wykresie sygnału np. barwa dźwięku (zbiór częstotliwości). Szybka transformacja Fouriera pozwala wygodnie przekształcać dziedzinę czasu na dziedzinę częstotliwości i odwrotnie [1, 15].

3.2. Problemy związane z DSP

Mimo wzrostu mocy obliczeniowej komputerów i precyzji procesorów, cyfrowe przetwarzanie dźwięku nadal sprawia pewne problemy. Przede wszystkim dokładność odwzorowania jest niewiele poniżej granicy, której przekroczenie pozwoli stwierdzić, iż dźwięku cyfrowego nie da się odróżnić od najwyższej jakości analogowych nagrań. Problemem otwartym pozostaje kwantyzacja na poziomie maksymalnie 64 bitów. Niewiele wskazuje na to, żeby w najbliższym czasie ta sytuacja mogła ulec poprawie [1, 12, 15].

DSP stwarza również problem kosztu obliczeniowego. Obecnie z domowym sprzętem można wykonywać operacje na dźwięku o średniej jakości w czasie rzeczywistym. Stosując np. tylko zmianę barwy dźwięku, można oczekiwać efektu praktycznie natychmiast (tj. czas odpowiedzi jest niezauważalny). Dlatego programowe odtwarzacze muzyki posiadają rozszerzenia umożliwiające stosowanie efektów dźwiękowych. Dźwięk o lepszej jakości wymaga pewnego oczekiwania na zakończenie operacji, bądź też lepszego procesora, aby zakończyć się w rozsądnym czasie. Ponieważ systemy rozpoznawania mowy służą rozpoznaniu konkretnej osoby, poza pierwotnymi zadaniami wymaga się również zastosowanie jako system czasu rzeczywistego [2, 7, 10, 11].

Kolejnym problemem pozostaje jakość sprzętu, który jest używany do przetwarzania sygnału. Najtańszy sprzęt może być mało wydajny i mało precyzyjny. Charakterystyka większości układów elektronicznych powoduje zakłócenia i szumy z powodu jakości użytych materiałów. Niektóre części większych systemów są wyspecjalizowane, np. reduktory szumu białego, czerwonego itp. [2, 7, 10, 11].

Problem jaki można zauważyć najczęściej podczas przetwarzania mowy, jest umiejscowienie źródła dźwięku względem odbiorcy (mikrofonu). Większa odległość mówcy od mikrofonu powoduje, że dźwięk staje się bardziej rozproszony i podatny na zakłócenia. To rozproszenie sprawia, że przejścia między kolejnymi sekcjami dźwięku (np. między głoskami a ciszą) są płynne, więc oddzielenie głosu mówcy od otoczenia jest trudniejsze. Charakterystyka częstotliwościowa takiego dźwięku jest również nieco zbliżona do szumu. Rozwiązaniem na ten problem, jak i na rozpoznanie kierunku dźwięku, jest zastosowanie kilku urządzeń przechwytywania. Mając do dyspozycji dwa mikrofony można stwierdzić, z której strony jest mówca – odpowiednio jeśli dźwięk jest głośniejszy w prawym czy w lewym mikrofonie [22].

4. Systemy rozpoznawania mowy

Celem poniższego rozdziału jest omówienie powszechnie stosowanych systemów rozpoznawania mowy. Z charakterystyki sygnału mowy wynika podobieństwo algorytmów. Parametry opisywanych algorytmów są wynikiem długotrwałych badań na dużej liczbie zebranych wcześniej danych. Gotowe aplikacje działają lepiej jeżeli słowniki mają więcej, z którymi porównywane są nagrane słowa [7, 8, 14].

Między algorytmami automatycznego rozpoznawania mowy istnieją pewnie różnice, według których częściowo da się je sklasyfikować. Istniejące systemy podzielić możemy na takie, których zadaniem jest rozpoznawania mowy ciągłej i takie które mają wykrywać wystąpienia izolowanych słów. Wyróżnić możemy również systemy zależnie od mówcy: dla jednej osoby bądź dla wielu. W pierwszym przypadku algorytmy ukierunkowane są na rozpoznawanie z pewną z góry określoną charakterystyką. W drugim zaś musi zostać spełnione założenie wielu potencjalnych mówców [7, 8, 14].

Kolejnym istotnym podziałem jest klasyfikacja ze względu na wielkość. Są to systemy [5, 14]:

- małe – do około 100 zwrotów-największa skuteczność ze względu na małą ilość zwrotów,
- średnie – co najwyżej 3000 zwrotów-aktualnie są przedmiotem badań i rozwoju,
- duże – 20000 i więcej-cechują się licznymi problemami, szczególnie kiedy mamy zamiar tworzyć uniwersalny system będący w stanie rozpoznawać mowę wielu osób. Trudniejsze, im wydajniej algorytm ma działać. Szczególnie jeśli ma to być system czasu rzeczywistego [3, 10, 14].

4.1. MFCC

Słowo ciężko powtórzyć w taki sam sposób. To jednak nie znaczy, że powtarzane słowa są różne. Nadal noszą one tą samą treść. Dlatego warto się skupić na pewnych cechach wypowiedzi. Oszacowanie podobieństw można uzyskać za pomocą MFCC (ang *Mel Frequency Cepstral Coeficients*) [1, 7, 15, 22].

Dziedzina czasu nie zawsze niesie ze sobą tyle informacji, co dziedzina częstotliwości. To sugeruje przedmiot badań sygnału. W przypadku tego algorytmu, nazwa w pewien sposób odzwierciedla podstawowe działanie i wynik. Jego wynikiem jest wektor cech, na którego podstawie można zaobserwować zmiany sygnału [1, 7, 15, 22].

Dane poddane analizie to pliki wave PCM z typową częstotliwością próbkowania 22050 Hz. Poniżej przedstawiono ogólny schemat postępowania w przypadku MFCC [1, 7, 15, 22]:

- podział dźwięku na nachodzące na siebie ramki o długości około 25 ms,
- obliczenie transformaty Fouriera z oknem Hamminga dla każdej ramki,
- przekształcenie każdego z widm chwilowych do postaci 24 zachodzących na siebie filtrów o kształcie trójkątnym równo rozłożonych na skali mel
- przekształcenie amplitud sygnałów na wyjściach filtrów ze skali liniowej do logarytmicznej
- obliczenie współczynników cepstralnych za pomocą dyskretnej transformaty cosinusowej przeprowadzonej na sygnałach wyjściowych z filtrów, oraz energii sygnału.

Ważne jest rozmieszczenie ramek (nachodzenie) tak, żeby pokrywały istotne fragmenty sygnału. Zarówno długość ramek jak i ich nakładanie można w pewnym zakresie regulować stosownie do pozostałej części aplikacji. Aby uzyskać możliwość dokonywania obserwacji częstotliwości i obliczenia cepstrum, należy najpierw poddać sygnał szybkiej transformacji Fouriera. Specyfika zadania sugeruje użycie okna Hamminga. Wybór tego okna czasowego pozwala na eliminację maksymalnej ilości zniekształceń, co w przypadku innych nie jest możliwe w takim samym stopniu. Dźwięk w domenie częstotliwości poddany jest przekształceniu na skalę mel zgodnie ze wzorem (2). Ze względu na dalsze operacje na dźwięku w skali mel stosuje się bank filtrów. Charakterystyka banku filtrów przypomina okienkowanie trójkątne powtarzane w pewnych przedziałach częstotliwości [1, 22].

W wyniku analizy za pomocą MFCC uzyskuje się wektor około 40 cech w skali son. Wektor ten stanowi charakterystykę dźwięku. Odnajdywanie podobieństw w próbkach dźwięku realizowane jest jako porównanie odległości euklidesowych wektorów cech. Proces ten wyrażony jest wzorem [1, 22]:

$$d(p,q)=d(q,p)=\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\ldots+(p_n-q_n)^2}=\sqrt{\sum_1^n (q_i-p_i)^2}$$

(3)

gdzie p, q – wektory cech, n – długość wektorów

4.2. Ukryte modele Markowa

W opisie powyższego algorytmu wypada zacząć od definicji łańcuchów Markowa. Niech $Q \neq \emptyset$, będzie skończonym zbiorem (zbiorem stanów). Pewien stan $k_0 \in Q$ jest wyróżniony jako stan początkowy. Łańcuch Markowa zadany jest przez macierz przejść $M=(p_{k,l})k,l \in Q$, która dla $k,l \in Q$ podaje prawdopodobieństwo przejścia ze stanu *k* do stanu l. *M* musi spełniać następujący warunek: dla każdego $k \in Q$ mamy [2]:

$$\sum_{k \in Q} p_{k,l}=1 \qquad (4)$$

Pewien układ, który w każdym momencie może znajdować się w jednym ze stanów $k \in Q$ jest opisywany właśnie łańcuchem Markowa. Układ taki obserwuje się w dyskretnych chwilach czasowych (czyli np. *t* = 0, 1, … n). Przyjmuje się, iż jego początkowy stan to k_0. Jeśli w momencie *t* układ znajduje się w stanie *k*, to w momencie *t* + 1 przechodzi w stan *l* z prawdopodobieństwem *p(k,l)*. Bardzo ważną – wręcz podstawową – cechą łańcuchów Markowa jest fakt, iż obecny stan nie jest zależny od *t* ani historii przejść, tylko od poprzedniego stanu [2].

Ukryte modele Markowa (ang. *Hidden Markov Models*) mają za podstawę definicję łańcuchów Markowa. Niech Σ będzie alfabetem $k \in Q$. Rozważaniu podlegają tutaj łańcuchy Markowa mogące się komunikować z otoczeniem poprzez emitowanie ciągów liter z alfabetu Σ. Mając dany HMM, będący w stanie $k \in Q$, emituje on symbol $x \in \sum$ z prawdopodobieństwem $e_k(x)$ oraz przechodzi do stanu l z prawdopodobieństwem *p(k,l)*. Jeśli w każdym stanie $k_0 \in Q$ ma być emitowany jakiś symbol, to należy przyjąć: $\sum_{x \in \Sigma} e_k(x)=1$

Jeśli dopuszcza się (z pewnym prawdopodobieństwem), że w pewnych stanach nic nie jest emitowane, to przyjmuje się $\sum_{x \in \Sigma} e_k(x) \leqslant 1$. Można przyjąć, że to co da się obserwować, to symbole emitowane przez układ a nie stany wewnętrzne układu (stąd nazwa „ukryte") [2].

Algorytm bazujący na ukrytych modelach Markowa pozwala na uzyskanie podobnych (ewentualnie trochę lepszych) wyników, niż MFCC, jednak nie został on wykorzystany w aplikacji. Na cele demonstracyjne został wybrany prostszy algorytm, a efekty są zadowalające. Dla poprawienia osiągów albo w celu napisania produkcyjnej wersji aplikacji, można posłużyć się również HMM (ang. *Hidden Markov Models*).

5. Aplikacja z wykorzystaniem MFCC

Celem niniejszego rozdziału jest opisanie aplikacji z zaimplementowanym systemem rozpoznawania mowy i omówienie dostępnych narzędzi. System powinien umożliwiać rozpoznanie pojedynczego mówcy na podstawie uprzednio przygotowanych nagrań znajdujących się w słowniku. Aplikacja zawiera praktyczne zastosowanie matematycznego modelu rozpoznawania mowy: rysowanie zgodnie z wydanym poleceniem. Takie zastosowanie systemu automatycznego rozpoznawania mowy pozwala na wykorzystanie szerokiego zakresu poleceń. Ponieważ zbiór poleceń jest stały (tzn. komputer nie ma prawa wykonać nieznanego polecenia), nie ma potrzeby sięgania do innych źródeł po znaczenie słowa, ani próby rozpoznania sensu.

5.1. Problemy i dobór narzędzi

Nie w każdej aplikacji da się zastosować rozpoznawanie mowy. Warto rozważyć system przypominający swoim działaniem konsolę systemową: po wydaniu polecenia i podaniu ewentualnych parametrów, oczekuje się odpowiedzi. Musimy dobrać algorytmy odpowiedzialne za realizację zadań postawionych przed projektem. Należy tu rozważyć przede wszystkim wielkość systemu i wydajność. Za rozmiar informacji w aplikacji rozpoznającej mowę, przyjmujemy rozmiar słownika:

- zajmowana pamięć na dysku,
- ilość mówców do rozpoznania,
- ilość poleceń, których oczekuje aplikacja,
- ilość pozycji w słowniku.

Słownik zajmuje maksymalnie tyle, ile wszystkie jego elementy razem wzięte, ich liczba może być różna. Istnieje rozwiązanie zmniejszenia rozmiaru słownika, wiele słów ma te same rdzenie (podstawa kilku liter) np. liczebniki: „jedenasty" ostatnie 5 liter ma takie same jak „dwunasty". To znaczy, że przyrostek „nasty" można zapisać w słowniku raz, a odpowiednie formatowanie pliku może posłużyć za dopasowanie do reszty. W niniejszej pracy wykorzystano mechanizm serializacji obiektów języka Java. Istotnym czynnikiem wpływającym na rozwój aplikacji jest platforma systemowa i sprzętowa, na której ma działać projekt. Rezygnując z mechanizmów specyficznych dla niektórych systemów, można zyskać uniwersalność. Niniejszy projekt został wykonany w języku Java. Biblioteką odpowiedzialną za graficzny interfejs użytkownika jest Swing. Ten sam kod można skompilować na niemal dowolnym systemie bez uprzedniego przygotowywania projektu. Środowiskiem programistycznym użytym do stworzenia projektu jest Eclipse.

Ważnym elementem projektu są dane wejściowe. Ponieważ rozpoznawanie mowy wiąże się z zapisem dźwięku z mikrofonu, należy zastanowić się jakie parametry powinien mieć ten dźwięk. W ogólnym przypadku przetwarzania sygnału im lepsza jego jakość, tym lepiej. W przypadku rozpoznawania mowy jednak warto obniżyć częstotliwość próbkowania. Pierwszym argumentem przemawiającym za doborem niskiej wartości jest tylko jeden kanał przechwytywania: całość słyszalnego sygnału zmieści się w częstotliwości 22050 Hz. Po drugie: do sygnału mowy nie ma potrzeby używania całego pasma. Można powiedzieć więcej – jest ono zbędne, gdyż nieużywane pasmo może powodować błędy.

Kwantyzacja 8 bitów bez znaku pozwoliła znacząco uprościć schemat przetwarzania. W Javie do przechowywania takiego dźwięku wystarczy tablica typu *byte*. Klasy do strumieniowego odczytu danych z mikrofonu przechwytują „surowe" dane w porcjach po 8 bitów, gdyż to jest najmniejsza możliwa do przechowywanie porcja danych. Z charakterystyki architektury x86 i innych opartych na wymienionej wynika, że w przypadku zmiennej mniejszej, niż 8 bitów miejsce zajmowane przez adres zajmie więcej danych, niż sama zmienna. Kolejnym powodem, dla którego nie może być to mniej, niż 8 bitów, jest szybkość działania: w systemie dwójkowym wyrównanie (adresów) do liczby bitów będącej potęgą dwójki jest najbardziej optymalne. Dlatego decyzja o kwantyzacji pozwala uniknąć nadmiarowych obliczeń podczas nagrywania.

5.2. Schemat i budowa aplikacji

Pisząc aplikację od podstaw, można napotkać kilka problemów:

- jak długo projekt ma być rozwijany – ile wersji ma powstać,
- jakie biblioteki mają być dołączone,
- które części programu mogą/ mają się zmienić.

Java w pewnym sensie „wymusza" pewne usystematyzowanie kodu. Pakiety, specyfikatory dostępu, typy anonimowe i interfejsy pozwoliły na stworzenie modularnej aplikacji. Aby dodać element graficznego interfejsu użytkownika, wystarczy dodać do okna „dziecko", które jest rozszerzeniem dla jednej z podstawowych klas Swing/ AWT. Projekt został podzielony na następujące pakiety, z czego każdy zawiera kilka klas. Poniżej przedstawiono strukturę kodu projektu:

- silentium
  - AudioPreProcessor.java
  - Console.java
  - Dictionary.java
  - DrawCanvas.java
  - FFT.java
  - MFCC.java
  - NewCommandDialog.java
  - Range.java
  - ReducedAudioInputStream.java
  - Silentium.java
  - SoundSystem.java
  - Utils.java
  - Wave.java
  - XMLSerializable.java
- comirva.audio
  - AudoPlayer.java

  - AudioPlaylistPlayer.java
  - XMLSerializable.java
- comirva.audio.util.math
  - CholeskyDecomposition.java
  - EigenvalueDecomposition.java
  - LUDecomposition.java
  - Maths.java
  - Matrix.java
  - NormalizedConvolution.java
  - QRDecomposition.java
  - SingularValueDecomposition.java

Pakiet silentium zawiera główne klasy aplikacji. Pozostałe zawierają klasy biblioteki zwanej CoMIRVA (Collection of Music Information Retrieval and Visualization Applications). Ta biblioteka stanowi zbiór kilku algorytmów pomocnych przy obróbce danych multimedialnych. Zawiera np. algorytm do szybkiej transformacji Fouriera. Niektóre klasy i metody wywoływane przez inne, są niejako pomocnicze. Inne są zależne od systemu: na przykład szybka transformacja Fouriera jest implementowana na różne sposoby, pozwalające na maksymalne wykorzystanie możliwości architektury, więc niektóre metody mogą mieć mniej wspólnego z samym algorytmem, a więcej z danym systemem komputerowym. W tej pracy skupiono się na opisaniu tylko najważniejszych klas i metod. Funkcjonalność taka jak konwersja z typu byte do typu double jest sprawą drugorzędną, nie mającą wpływu na działanie algorytmu (w zasadzie to mają wpływ na szybkość działania, ale bardzo małe ułamki sekund są tu widziane jako zupełny brak różnicy).

Biblioteka Swing domyślnie ma swój specyficzny, niezależny od systemu, wygląd. Kolejnymi zaletami wartymi wspomnienia są:

- duża liczba komponentów,
- dobry MVC (ang. *Model-View-Controller*)
- stabilność, „dojrzałość” projektu (w wersji 1.3.1 Java SE biblioteka została przeniesiona do biblioteki standardowej [21])
- rozszerzalność: budowa bliblioteki pozwala na rozszerzenie niemal dowolnego komponentu (np. JComponent, JButton).

Bibliotece Swing podobnie jak każdej innej, można zarzucić kilka rzeczy, np. dłuższy czas odpowiedzi kontrolek i większe zużycie pamięci RAM niż w przypadku systemowych rozwiązań.

5.3. Realizacja programowa aplikacji

Aplikacja została podzielona na kilka plików. Celem jest opisanie programowej realizacji wymienionych elementów. Pierwszym opisanym elementem jest graficzny interfejs użytkownika i powiązanie go z operacjami wykonywaniami przez aplikację. Kolejnymi wymienianymi elementami będą struktury funkcji odpowiedzialnych za samo wywołanie i operacje. W listingach obejmowano tylko najważniejsze części kodu. Kod niemal całego graficznego interfejsu użytkownika zawarty jest w klasie silentium. W tym pliku znajdują się funkcje realizujące odpowiedź aplikacji na zdarzenia przyciśnięcia przycisku, inicjując tym samym odpowiednie operacje, np. nagrywanie dźwięku. Wykonano je z pomocą obiektów typu ActionListener. W przeciążonej funkcji actionPerformed sprawdzane jest pochodzenie sygnału i wywoływana jest funkcja odpowiednia do przycisku. W poniższych listingach przedstawiono kod realizujący przypisanie zdarzeń do metod i wywoływane funkcje. Kolejne fragmenty kodu przedstawiają operacje wykonywane po naciśnięciu przycisków.

Listing 1. Dowiązanie zdarzeń przyciśnięcia przycisków do funkcji.
Listing 1. *Linking the events of pressing buttons to functions.*

Listing 2. Wczytywanie słownika.
Listing 2. *Loading of the dictionary.*

```
this.buttonsDictionaryListener = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
                if (e.getSource() == _parent.buttonLoadDictionary) {
                        _parent.loadDictionary();
                } else if (e.getSource() == _parent.buttonFindWord) {
                        _parent.recordAndFind();
                } else if (e.getSource() == _parent.buttonAddNewWord) {
                        _parent.addDictionaryPosition();
                } else if (e.getSource() ==        _parent.buttonAddNewWordFromFile) {
                        _parent.addDictionaryPositionFromFile();
                        }
                }
        };
this.buttonsMainListener = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
                if (e.getSource() == _parent.buttonRecordSound) {
                        _parent.recordCommand();
                } else if (e.getSource() == _parent.buttonClearCanvas) {
                        _parent.canvas.lines.clear();
                }
```

```
public void loadDictionary() {
        int retVal = this.fileChooser.showOpenDialog(this);
        if (retVal == JFileChooser.APPROVE_OPTION) {
                String name = this.fileChooser.getSelectedFile().getAbsolutePath();
                if (this.dictionary == null) {
                        this.dictionary = new silentium.Dictionary(name, 20, 20);
                }
                this.dictionaryPath = name;
...
                this.deserializeDictionary(name);
                for (String current : this.dictionary.wordList) {
                        this.dictionaryPositionsListModel.addElement(current);
                }
                JoptionPane.showMessageDialog(this, "Wczytane pozycje " +
this.dictionary.wordList.size(), "Informacja", JOptionPane.INFORMATION_MESSAGE);
        }
}
```

Listing 3. Użycie serializacji do wczytywania słownika.
Listing 3. *The use of serialization to load the dictionary.*

```
public void deserializeDictionary(String path) {
        try {
                FileInputStream fis = new FileInputStream(path);
                ObjectInputStream ois = new ObjectInputStream(fis);
                this.dictionary = (Dictionary) ois.readObject();
                fis.close();
                ois.close();
        } catch (FileNotFoundException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        } catch (ClassNotFoundException e) {
                e.printStackTrace();
        }
}
```

Listing 4. Użycie mechanizmu serializacji do zapisu słownika.
Listing 4. The use of serialization mechanism to save the dictionary.

```
public void serializeDictionary() {
        try {
                if (this.dictionary == null) {
                        JOptionPane.showMessageDialog(this, "Nie można zapisać słownika",
                                        "Informacja", JOptionPane.ERROR_MESSAGE);
                        return;
                }
                FileOutputStream fos = new FileOutputStream(System.getProperty("user.home")
                                + System.getProperty("file.separator") +
"SilentiumDictionaryDump.bin");
                ObjectOutputStream oos = new ObjectOutputStream(fos);
                oos.writeObject(this.dictionary);
                oos.flush();
                fos.close();
                oos.close();
        } catch (FileNotFoundException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
}
```

Listing 5. Uzyskanie wektora cech ze strumienia dźwięku.
Listing 5. Obtaining of the feature vector from the stream of sound.

```
public double[] processWindow(double[] window, int start) throws IllegalArgumentException {
        int fftSize = (windowSize / 2) + 1;
        if (start < 0)
                throw new IllegalArgumentException("start must be a positve value");
        if (window == null || window.length - start < windowSize)
                throw new IllegalArgumentException("the given data array must not be a null value
and must contain data for one window");
        for (int j = 0; j < windowSize; j++)
                buffer[j] = window[j + start];
        normalizedPowerFFT.transform(buffer, null);
        Matrix x = new Matrix(buffer, windowSize);
        x = x.getMatrix(0, fftSize - 1, 0, 0);
        x = melFilterBanks.times(x);
        double log10 = 10 * (1 / Math.log(10));
        x.thrunkAtLowerBoundary(1);
        x.logEquals();
        x.timesEquals(log10);
        x = dctMatrix.times(x);
        return x.getColumnPackedCopy();
}
```

5.4. Wyniki działania aplikacji

Aby aplikacja spełniała stawiane przed nią zadania, należy sprawdzać zgodność jej działania z oczekiwaniami. Musi to się odbywać niemal na każdym etapie pisania. Testowanie aplikacji podczas pisania dotyczyło głównie na badaniu poprawności reakcji programu na polecenia użytkownika. Test działania najważniejszego elementu, czyli rozpoznawania mowy, odbywał się pod niemal na końcu projektu, ponieważ wymagało to przygotowania paru innych elementów np. zbieranie próbek głosu z mikrofonu.
W procedurze testowej stworzono słownik różnymi nagraniami – dostępnymi w aplikacji poleceniami. Następnie sprawdzona została funkcja wyszukiwania słowa, które powinno istnieć w słowniku. Wyszukiwanie polegało na wyznaczeniu odległości euklidesowych między wektorami MFCC dla każdego słowa i wyborze najmniejszej odległości.

Tab. 1. Skuteczność działania mechanizmu rozpoznawania.
Tab. 1. The effectiveness of the recognition mechanism.

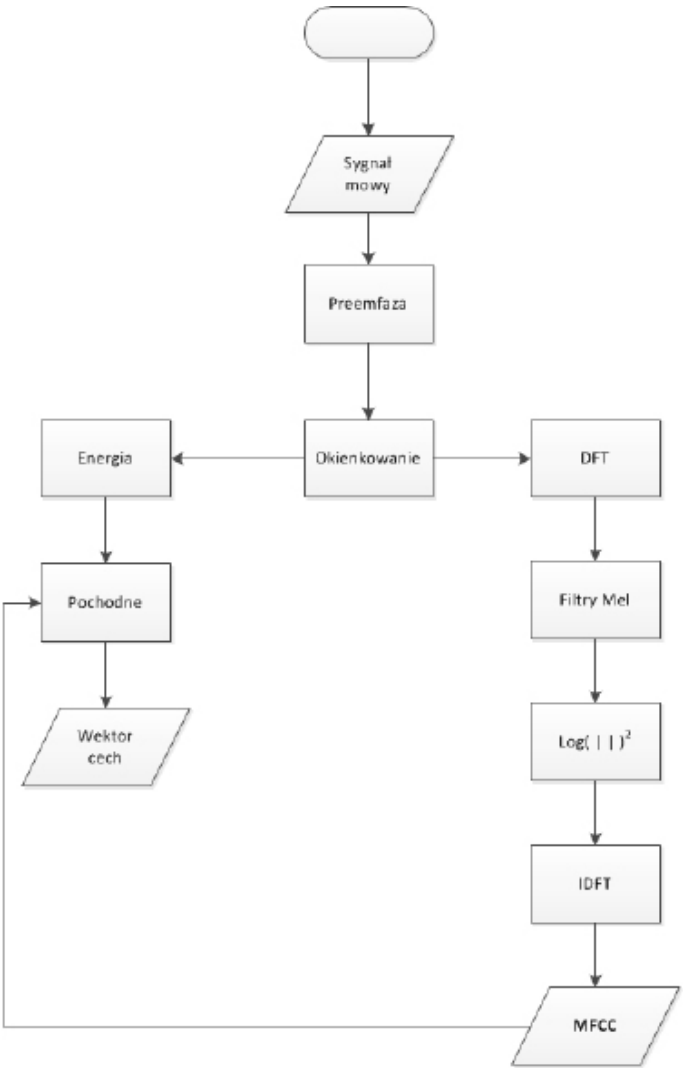
Słowo	Próby	Rozpoznania prawidłowe
prawa	25	20 (80%)
lewa	40	31 (~77%)
cofnij	35	22 (~63%)
góra	20	13 (65%)
wyczyść	36	23 (~63%)
zamknij	45	40 (~88%)

5.5 Zestawienie wydajności MFCC - Działanie programu

Do przeprowadzonych testów został stworzony słownik w kilku rozmiarach. Pomiary zostały dokonane dla operacji wyszukiwania w słowniku słowa nagranego mikrofonem. Na wyszukiwanie składają się następujące operacje:

- skalowanie dźwięku
- wyliczanie MFCC dla nagranego słowa
- porównanie z pozycjami słownika: odległości euklidesowe

Schemat działania programu opartego o MFCC przedstawiono na Ryc. 4.



Ryc. 4. Schemat działania algorytmu wyznaczania MFCC.
Fig. 4. *The diagram of the MFCC determining algorithm.*

Porównanie według ilości wątków

W poniższej tabeli przedstawiono czas wykonania algorytmu w zależności od zasobów użytych na komputerze Sun Constellation System Halo2. Dla przykładu operację porównania powtórzono kilka razy.

Tab. 2. Czas realizacji zrównoleglonego algorytmu MFCC na komputerze Sun Constellation System Halo2 (czasy podane w mikrosekundach).
Tab. 2. Duration time of the parallelized MFCC algorithm on a Sun Constellation System Halo2 computer (times in microseconds).

Próba	1 wątek	2 wątki	4 wątki	16 wątków	32 wątki
1	45913039	22899451	9981095	2849774	1481066
2	48639417	23181627	10573786	3648998	1569013
3	51054355	24811597	11298702	3668891	1646915
4	52799836	25542779	11478225	3277231	1703221
5	54441505	25924526	11865109	3471128	1756178
6	101170839	48676590	21993660	6279569	3263575
7	116008978	55242370	28219343	7200957	3742225
8	120149721	57214152	26119504	7457569	3875797
9	124332854	59206120	27028881	7717212	4019737

Porównanie według maszyn

Zestawienie zawiera średni czas działania programu w zależności od rozmiaru słownika i użytego systemu.

Tab. 3. Porównanie wydajności trzech różnych komputerów: IBM Blue Gene/Q, Sun Constellation System Halo2 oraz IBM Power 775 podczas realizacji zrównoleglonego algorytmu MFCC. Wykorzystano dwa wątki (czasy podane w mikrosekundach).
Tab. 3. *Compare of the performance of three different computers: IBM Blue Gene/Q, Sun Constellation System Halo2 and IBM Power 775 during the realisation of the MFCC parallelized algorithm. Uses two threads* (*times in microseconds*)

Rozmiar słownika	IBM Blue Gene/Q	Sun Constellation System Halo2	IBM Power 775
256	55968900	34810729	45638649
512	96382944	59793289	84824320
1024	147881027	104780294	132123888
2048	313507777	201178164	265569014
4096	732040659	362120695	589563211

6. Wnioski

W niniejszej pracy zaprezentowano działanie tylko jednego spośród algorytmów rozpoznawania mowy. Wybór odpowiedniego algorytmu nie zawsze jest prosty. Mimo znacznych postępów idealny algorytm nie istnieje. Z tego względu warto rozważać wybór pod konkretny projekt i konkretne zastosowania. Jak z każdym większym projektem, należy przed rozpoczęciem działań zastanowić się nad próbami ich rozwiązania. Może się okazać, że nie wszystkie problemy wystąpią naraz i nie każdy będzie tak dotkliwy, żeby znacząco utrudnić tworzenie bądź też używanie aplikacji. Jeśli system ma rozpoznawać konkretne słowa bądź konkretnego mówcę, to nie trzeba martwić się rozróżnianiem mówców [1, 15, 22].
Nowe technologie mogą budzić zarówno podziw i fascynację, jak i obawy. Komputery i telefony są najczęstszymi środkiem komunikacji. Ludzie starsi, chorzy czy w jakikolwiek inny sposób pozbawieni możliwości korzystania z komputerów nie mogąc komunikować się z innymi za ich pomocą, tacy ludzie tracą kontakt ze znajomymi i rodziną. Projektując nowe algorytmy należy brać pod uwagę takie problemy i starać się dostosować aplikację do jak największej liczby korzystających [13, 25, 26].
Aplikacja spełniła swoje założenia. Graficzny interfejs użytkownika pozwolił na wygodne ukazanie działania takiej aplikacji. Wykorzystany język programowania pozwolił na możliwe szerokie udostępnienie aplikacji. Modularna budowa projektu stanowi tutaj dobrą podstawę do dalszego rozwoju.

Podziękowania

Obliczenia wykonano w Interdyscyplinarnym Centrum Modelowania Matematycznego i Komputerowego (ICM) Uniwersytetu Warszawskiego w ramach grantu obliczeniowego nr G55-11.

Literatura (References)

[1] X. Huang, Spoken Language Processing - A Guide to Theory, Algorithm, and System Development, Prentice Hall PTR 2001.
[2] A. M. Wiśniewski, Automatyczne rozpoznawanie mowy bazujące na ukrytych modelach Markowa – problemy i metody, Biuletyn Instytutu Automatyki i Robotyki WAT nr 12, 2001.
[3] A. Viterbi, Eror bounds for convolutional codes and an asymptotically optimum decoding algorithm, IEEE Trans. Information Theory, vol IT-13, pp 260-269, 1967
[4] J. Schalkwyk, P. Hoson, E. Kasier, K. Shobaki, CSLU -HMM: The CSLU Hidden Markov Modelling Environment, Center of Spoken Language Understanding, Oregon Graduate Institute of Science & Technology
[5] F. Fissore, E. Giachin, P. Laface, P. Massafra, Using Grammars in forward and backward search, Proceedings Eurospeech 9, Berlin 1993.
[6] M. Schedl, http://www.cp.jku.at/people/schedl/Research/Development/CoMIRVA/webpage/CoMIRVA.html, stan z dnia 25.01.2013.
[7] Carnegie Mellon University, http://cmusphinx.sourceforge.net/, stan z dnia 25.01.2013.
[8] Carnegie Mellon University, http://www.speech.cs.cmu.edu/, stan z dnia 25.01.2013.
[9] NVIDIA Corporation, http://www.nvidia.pl/object/cuda_home_new_pl.html, stan z dnia 25.01.2013
[10] Massachusetts Institute of Technology, http://mit.edu, stan z dnia 25.01.2013.
[11] Alcatel- Lucent (Bell Labs) http://www.alcatel-lucent.com, stan z dnia 25.01.2013.
[12] G. Moore, Cramming more components onto integrated circuits, Electronics, Volume 38, Number 8, April 19, 1965.
[13] On Board PR Ecco Network, http://www.onboard.pl/data/file/pdf/raport__swiadomosc_polakow_w_rzeczywistosci_cyfrowej.pdf, stan z dnia 25.01.2013
[14] K. R. Farrell, R. J. Mammone, K. T. Assaleh, Speaker Recognition Using Neural Networks and Conventional Classifiers, IEEE TRANSACTIONS ON SPEECH AND AUDIO PROCESSING, VOL. 2, NO. 1, PART 11, JANUARY 1994.
[15] T. P. Zieliński, Cyfrowe Przetwarzanie Sygnałów – Od Teorii do Zastosowań, Wydawnictwa Komunikacji i Łączności, Wydanie 1, Warszawa 2005
[16] Rada Języka Polskiego, http://www.rjp.pan.pl, stan z dnia 25.01.2013.
[17] Eclipse Fundation, http://www.eclipse.org, stan z dnia 04.02.2013.
[18] Oracle Corporation, http://www.oracle.com, stan z dnia 04.02.2013.
[19] R.J. Marks II, Introduction to Shannon Sampling and Interpolation Theory, Spinger-Verlag, New York 1991.
[20] W. Chen, https://blogs.oracle.com/Swing/entry/awt_swt_swing_java_gui3, stan z dnia 05.02.2013.
[21] Oracle Corporation, Swing 1.3 features http://docs.oracle.com/javase/1.3/docs/relnotes/features.html, stan z dnia 05.02.2013.
[22] M. L. Seltzer, Microphone Array Processing for Robust Speech Recognition, Carnegie Mellon University, Pittsburgh 2003.
[23] Microsoft, http://msdn.microsoft.com/pl-pl/ms348103.aspx, stan z dnia 07.02.2013.
[24] Microsoft, MSDN http://msdn.microsoft.com/en-us/vstudio/hh341490.aspx, stan z dnia 07.02.2013.
[25] Social Press, http://socialpress.pl/2013/02/sotrender-podsumowuje-2012-rok-na-polskim-facebooku-popularnosc-marek-rosnie-w-bardzo-szybkim-tempie/#, stan z dnia 08.02.2013.
[26] Gazeta Wyborcza, http://wyborcza.biz/biznes/1,101562,6289081,Nokia_bierze_Skype_a_na_poklad.html, stan z dnia 08.02.2013.