AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI,
ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA TELEKOMUNIKACJI

Praca dyplomowa inżynierska

Aplikacja mobilna ułatwiająca eliminację nawyku mówienia „yyyy" podczas prezentacji.

Autor: *Tomasz Balawajder*
Kierunek studiów: *Elektronika i Telekomunikacja*
Opiekun pracy: *dr inż. Jarosław Bułat*

Kraków, 2014

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Spis treści

Wstęp

Cele pracy

Wszyscy mieliśmy okazję przemawiać do ludzi, jak również wysłuchiwać przemówień innych. Często problemem mówców nie jest nieznajomość tematu czy brak przygotowania, a technika mowy. Problemem stają się wrodzone oraz nabyte wady wymowy, nalot gwary regionalnej czy nieprawidłowa dykcja. Wiele złych nawyków można wyeliminować poprzez ćwiczenia. Jednym z błędów wymowy jest nawyk mówienia „yyy" czy „eee" w czasie prowadzenia prezentacji. Jest to przyzwyczajenie, które potrafi łatwo i szybko zniechęcić słuchacza do nawet najlepiej merytorycznie przygotowanej prezentacji. Takie zawieszanie głosu może mieć uzasadnienie w tym, że mówca jest nieprzygotowany do prezentacji i „zapełnia" w ten sposób czas. Najczęściej jednak jest to nieświadomy odruch, pozwalający prelegentowi na chwilę przerwy w merytorycznych rozważaniach. To właśnie poczucie nieświadomości stało się powodem zajęcia się tym tematem w niniejszej pracy. Jednym ze sposobów uświadomienia prelegenta o błędach wymowy jest informowanie go o nich w trakcie prezentacji. Kiedy prelegent jest na bieżąco informowany o swoich błędach wymowy, może je szybko i skutecznie skorygować.

Celem pracy jest opracowanie aplikacji umożliwiającej eliminację nawyku mówienia ‚yyy" oraz „eee" podczas prezentacji. Założono, że aplikacja ma zostać zaimplementowana dla systemu Android. Jej celem będzie rejestrowanie dźwięku w czasie rzeczywistym, analizowanie go pod kątem wystąpienia „yyy" lub „eee" oraz prezentacja wyników tej analizy w sposób zdefiniowany przez użytkownika. Zaprojektowana aplikacja powinna, oprócz przetwarzania dźwięku, dostarczać użytkownikowi prosty i przejrzysty interfejs, który umożliwi mu szybkie oraz dyskretne posługiwanie się nią w czasie prelekcji. Dodatkowo w programie znajdzie sie zegar, który będzie odliczał czas prezentacji w taki sposób, aby prelegent mógł go swobodnie kontrolować. Następną funkcją programu powinna być możliwość gromadzenia przez niego statystyk o rejestrowanej prezentacji. Zebrane dane należy wyświetlić użytkownikowi po zakończeniu przez niego prelekcji.

Niniejsza praca stanowi opis procesu tworzenia aplikacji opisanej powyżej. Zawiera cztery rozdziały, które szczegółowo obrazują jak przebiegało projektowanie oraz implementacja programu. Rozdział pierwszy zawiera podstawy teoretyczne fizycznych własności dźwięku, oraz opis ludzkiego aparatu mowy. Znajdują się w nim też podstawy teorii cyfrowego przetwarzania sygnałów wraz z najważniejszymi twierdzeniami. Na końcu przedstawiono i porównano sukcesy z dziedziny przetwarzania dźwięku jakie do tej pozy zostały osiągnięte.

Rozdział 2 zawiera specyfikację programu opisywanego w pracy. Szczegółowo zostały sprecyzowane wymagania, jakie powinna spełniać aplikacja, która została nazwana *yyyKiller*. Zobrazowano w nim również algorytm, wykorzystywany później w implementacji kodu źródłowego aplikacji. Znajduje się tu zarówno jego opis, jak też charakterystyka algorytmów będących częściami składowymi głównej procedury. Ponadto opisano w nim przebieg procesu badawczego poprzedzającego implementację oraz testy aplikacji.

Kolejny rozdział charakteryzuje platformę na jakiej przeprowadzono implementację aplikacji. Rozpoczynając od teoretycznych rozważań na temat systemów mobilnych opisuje proces tworzenia aplikacji, zastosowanych technik i wzorców oraz prezentuje diagramy i fragmenty kodu źródłowego programu. Skupia się też na sposobie przeniesienia algorytmu do środowiska programistycznego Android.

Czwarty rozdział pracy zawiera opis przeprowadzonych testów aplikacji. Prezentuje możliwe konfiguracje programu oraz stanowi opis uzyskanych rezultatów. W zakończeniu pracy są zamieszczone wnioski oraz podsumowanie osiągniętych efektów.

1. Charakterystyka mowy ludzkiej

W niniejszym rozdziale przedstawiono podstawowe informacje dotyczące cech fonetycznych oraz podstawy teorii przetwarzania mowy. Znajdują się w nim również informacje na temat fizycznych właściwości fal dźwiękowych.

1.1. Fizyczne właściwości dźwięku

Wytwarzanie głosu jest bardzo złożonym zjawiskiem akustycznym i mechanicznym. Dźwięk, w kategoriach fizycznych jest wrażeniem słuchowym spowodowanym drganiami fal akustycznych rozchodzących się w ośrodku sprężystym. Fala akustyczna to zaburzenie gęstości w formie fali podłużnej rozchodzące się w powietrzu lub innym ośrodku. Są to zarówno te fale, które powodują wrażenia słuchowe, jak również fale o częstotliwościach wykraczających poza pasmo słyszalności ludzkich zmysłów. Ze względu na częstotliwość dźwięki możemy podzielić w następujący sposób [5]:

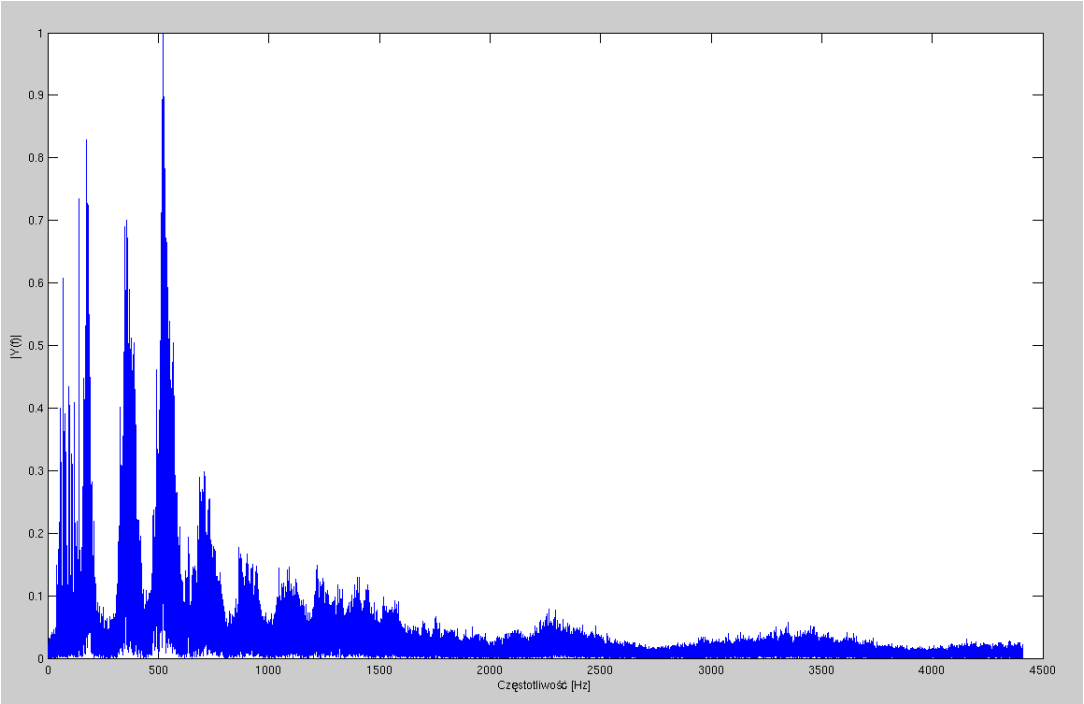
- infradźwięki - o częstotliwości poniżej 16 Hz,
- dźwięki słyszalne - o częstotliwości w paśmie 16 Hz–20 kHz,
- ultradźwięki - powyżej 20 kHz,
- hiperdźwięki - powyżej 10^{10} Hz.

Jak można zauważyć, że zakres dźwięków słyszalnych stanowi niewielki wycinek całego pasma dźwięków otaczających człowieka. Fizycznymi aspektami dźwięków są:

- widmo,
- natężenie,
- czas trwania dźwięku.

Wszystkie cechy dźwięku możemy łączyć w analizie wykorzystując fakt zmienności wybranych cech w trakcie trwania dźwięku. Wykorzystuje się je do poszukiwania złóż ropy naftowej czy badań skorupy ziemskiej [7]. Fale dźwiękowe służą też jako narzędzie do prowadzenia badań medycznych oraz materiałowych. Widmem dźwięku nazywamy rozkład natężenia składowych sinusoidalnych w funkcji ich

częstotliwości. Widmo dźwięku uzyskujemy za pomocą metod spektroskopii lub w wyniku analizy częstotliwościowej, np. za pomocą transformaty Fouriera sygnału dźwiękowego. Zwykle przedstawia się je jako wykres prążkowy, który na osi poziomej zawiera częstotliwości, natomiast na osi pionowej amplitudy lub energię wyrażoną w decybelach dla poszczególnych składowych. Przykłady takich widm przedstawia rysunek 1.1.



Rysunek 1.1: Przykładowe widmo sygnału mowy

Natężenie dźwięku jest wielkością opisującą energię fali akustycznej. W fizyce wyrażana jest ona w $\frac{W}{m^2}$ jako powierzchniowa gęstość mocy fali akustycznej. Natężenie I fali dźwiękowej na pewnej powierzchni to średnia szybkość w przeliczeniu na jednostkę powierzchni, z jaką fala dostarcza energię do tej powierzchni. Bardziej powszechne jest jednak podawanie natężenia dźwięku w skali logarytmicznej, czyli wartości poziomu natężenia dźwięku (głośności). Dlatego też zamiast mówić o natężeniu dźwięku możemy używać terminu **głośności dźwięku** β, który zdefiniowano jako

$$\beta = 10log_{10}\left(\frac{I}{I_0}\right) \tag{1}$$

gdzie I_0 jest standardowym natężeniem odniesienia. Wybrane zostało w taki sposób, aby było jak najbliższe dolnej granicy słyszalności ludzkiego ucha. I_0 wynosi $10^{-12}\frac{W}{m^2}$.

Podawanie natężenia dźwięku w takiej skali jest bliższe opisowi zachowania ludzkich narządów słuchu. Odpowiada bowiem zmianom wartości natężenia w odniesieniu do ustalonego poziomu odniesienia. Inną skalą, która uwzględnia fizjologię ludzkiego ucha jest skala głośności mierzona w fonach. Ważną rolę w analizie natężenia dźwięku odgrywa jej zmiana wraz z odległością. Sposób w jaki ulega ona zmianie jest zwykle skomplikowany. Dźwięk jest emitowany przez źródła sferycznie. Jednak niektóre źródła

rzeczywiste mogą wytwarzać dźwięk rozchodzący się w jednym wybranym kierunku. Dodatkowo należy wziąć pod uwagę zjawiska jakie zachodzą w trakcie emisji dźwięku. Otoczenie zwykle wytwarza echo odbijając dźwięki co powoduje wzmocnienie lub zmniejszenie natężenia w miejscach oddalonych od źródła. Do podstawowej analizy można założyć, że dźwięk rozchodzi się izotropowo od źródła, tzn. ze stałym natężeniem we wszystkich kierunkach. Zgodnie z zależnością

$$I = \frac{P_s}{4\pi r^2} \tag{2}$$

gdzie P_s to moc akustyczna źródła przy założeniu, że zachowana zostaje energia mechaniczna, można określić natężenie dźwięku w dowolnym punkcie. Należy zwrócić szczególną uwagę, że jest ono odwrotnie proporcjonalne to kwadratu odległości od źródła, zatem niewielka zmiana odległości może spowodować dużą zmianę natężenia dźwięku.

Widmo sygnału składa się z sumy składowych sinusoidalnych. Jeśli uznać widmo za szereg harmoniczny to wyróżniamy spośród nich jedną, zwaną tonem podstawowym. Składowa główna f_0 oznacza falę harmoniczną o najniższej częstotliwości. Ton podstawowy występuje m.in. w instrumentach muzycznych. Nie każdy dźwięk rejestrowany przez ucho jest falą harmoniczną o konkretnej częstotliwości. Źródła dźwięku wytwarzają, oprócz tonu podstawowego dźwięki o wyższych częstotliwościach będących wielokrotnością tonu podstawowego. Poniżej przedstawiono zależność na *n*-tą harmoniczną.

$$f_n = nf_0, \quad n \in N \tag{3}$$

Właściwość dźwięku, pozwalająca odróżniać dźwięki o tym samym tonie podstawowym to barwa dźwięku. Barwa dźwięku jest zależna od liczby i częstotliwości składowych harmonicznych, oraz od ich amplitud.

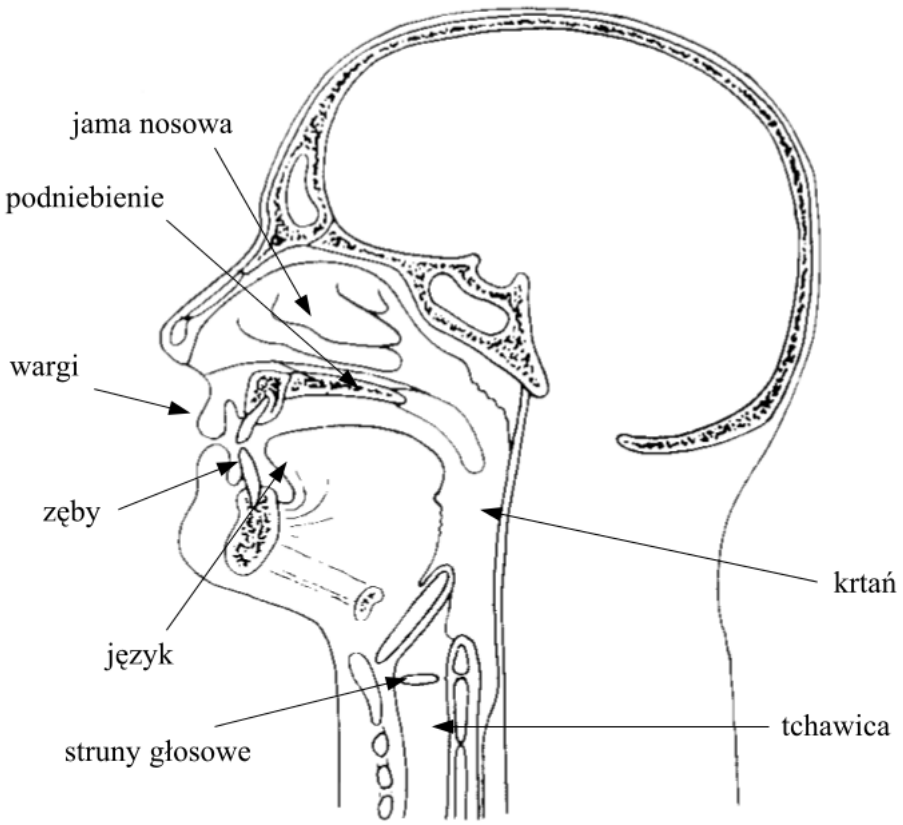
1.2. Opis ludzkiego aparatu mowy

Jednym z najważniejszych źródeł informacji o otaczającym nas świecie jest dźwięk. Jest on obecny wszędzie i zawsze nam towarzyszy. Szacuje się, że około 10% populacji ludzkiej cierpi na niedosłuch [11]. Mowa pozwala przekazywać informacje, wyrażać uczucia, emocje oraz artystyczne zachwyty. Mowa stała się fundamentem cywilizacji i kultury człowieka.

W procesie wytwarzania głosu biorą udział następujace elementy układu oddechowego: wargi i zęby, jama nosowa, podniebienie, język, struny głosowe (głośnia), krtań, tchawica i oskrzela oraz płuca i przepona. Cały aparat mowy odpowiedzialny za wytwarzanie dźwięku można podzielić na 3 główne części:

1. Aparat oddechowy.
2. Aparat fonacyjny.
3. Aparat artykulacyjny.

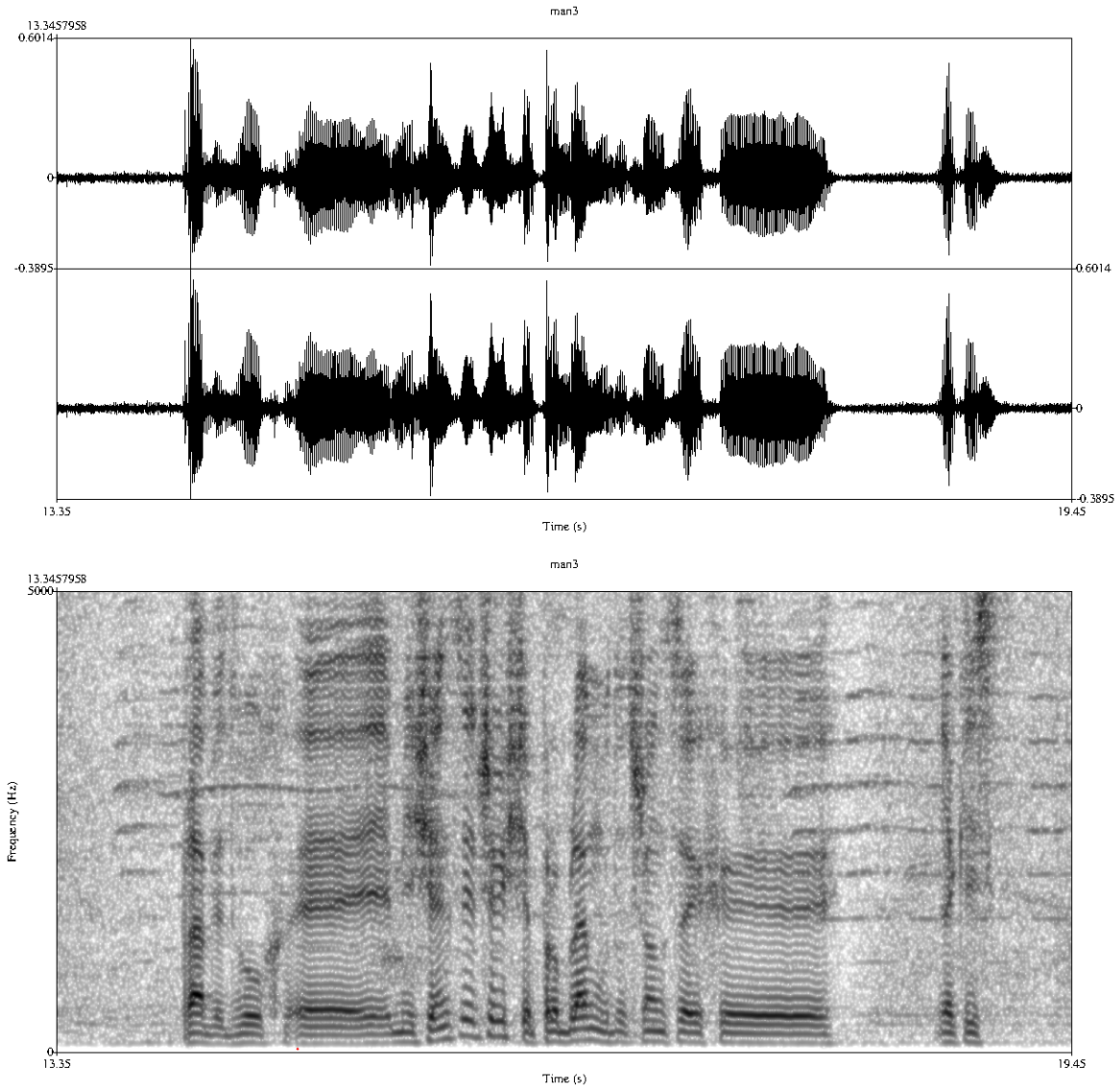
Źródłem energii dla układu głosowego jest część krtani poniżej strun głosowych oraz tchawica, płuca i przepona. Aparat oddechowy wytwarza falę dźwiękową, która następnie jest modulowana w górnych warstwach układu głosowego. Powietrze przechodząc z tchawicy trafia do układu fonacyjnego, gdzie przechodząc przez głośnię pobudza struny głosowe. W zależności od ułożenia więzadeł głosowych, od tego czy są one rozchylone na całej długości czy okresowo się rozchylają, powstaje fala dźwiękowa mająca postać ciągu impulsów lub szumu. To właśnie tam powstaje ton podstawowy głosu (ang. *pitch*). Wysokość tonu podstawowego jest związana z takimi cechami jak płeć rozmówcy, jego wiek oraz jego wrodzone lub nabyte cechy charakterystyczne, np. oddziaływanie miejsca zamieszkania. Wpływ na zmianę wysokości pierwszej składowej ma również intonacja głosu, widoczna np. przy artykulacji zdań pytających czy oznajmujących. U człowieka ton podstawowy może zmieniać sie w zakresie od 80 do 960 Hz. Bardzo ważną rolę w tym wypadku odgrywa płeć. U mężczyzn ton podstawowy zawiera się w paśmie ok. 80–480 Hz, dla kobiet pasmo jest około dwa razy większe ok. 160–960 Hz. Schemat budowy traktu głosowego przedstawiono na rysunku 1.2. Na schemacie pominięte zostały elementy niespełniające istotnej roli w procesie formowania sygnału pobudzenia, a biorące udział w generowaniu mowy.



Rysunek 1.2: Schemat traktu głosowego

Przy przechodzeniu fali dźwiękowej przez trakt głosowy kształtowane jest widmo sygnału krtaniowego, które powstaje po przejściu fali przez głośnię. Tak powstały sygnał dźwiękowy można poddać

analizie częstotliwościowej. Takie przekształcenie jest możliwe m.in. za pomocą programu `Matlab` oraz `Praat`. Na rysunku 1.3 przedstawiony został wynik analizy czasowo–częstotliwościowej krótkiego fragmentu mowy dla ramek o długości 20 ms.



Rysunek 1.3: Wykres czasowo–częstotliwościowej analizy głosu

Analiza i przetwarzanie głosu jest procesem niezwykle złożonym i skomplikowanym. Biorąc pod uwagę bezwładność ludzkich narządów słuchu możemy dokonać pewnych uproszczeń. Jednym z najczęstszych założeń jest przyjęcie, że dla krótkich fragmentów sygnału mowy, trakt głosowy jest liniowo niezmiennym w czasie układem. Przyjmuje się więc, że dla fragmentów mowy o długości ok. 20 ms parametry opisujące głos są takie same. Pozwala to na zautomatyzowanie mechanizmów analizy dźwięku oraz ułatwia ich implementację.

1.3. Cele i korzyści cyfrowego przetwarzania głosu

Rejestrowanie i odtwarzanie dźwięków od zawsze interesowało ludzi. Dopiero w 1857 roku udało się to Francuzowi Édouard-Léonowi Scottowi de Martinville'owi. Nagrał on dźwięk, ale nie potrafił go odtworzyć. Pełna rekonstrukcja nagranego dźwięku udała się dopiero za pomocą fonografu cylindrycznego opatentowanego przez Thomasa Edisona w 1877 roku. Następnie bardzo szybko rozwinął się przemysł nagrywania dźwięku. Powstawały coraz nowsze i lepszej jakości nośniki, takie jak płyty winylowe czy taśmy magnetyczne. Zapisywanie dźwięku w postaci sygnału analogowego znacznie ograniczało możliwości jego przetwarzania. Dopiero rozwój komputerów i technik procesorów sygnałowych doprowadził do zapisu dźwięku w postaci cyfrowej. Jakość zapisu cyfrowego oraz możliwości jego przetwarzania i przesyłania spowodowały bardzo dynamiczny rozwój tej dziedziny nauki, która bardzo szybko wyparła zapis analogowy. Aby przetworzyć dźwięk z postaci analogowej na cyfrową z możliwością późniejszego odtworzenia muszą zostać spełnione pewne założenia. Twierdzenie Shannona nakłada dwa najważniejsze założenia:

1. Widmo sygnału jest ograniczone, tzn. istnieje $f_m > 0$, takie że $|f| > f_m$
2. Próbki $\{s(n\Delta t)\}_{n=-\infty}^{\infty}$ sygnału są pobierane w odstępach czasu Δt takich, że $\frac{1}{\Delta t} \stackrel{df}{=} f_p \geq 2f_m$

Twierdzenie mówi, że jeśli zostaną one spełnione to możliwe jest odtworzenie dźwięku zgodnie z zależnością 4.

$$s(t) = \sum_{n=-\infty}^{\infty} s(n\Delta t) \frac{\sin(\pi(t - n\Delta t)/\Delta t)}{\pi(t - n\Delta t)/\Delta t} \tag{4}$$

W porównaniu do przetwarzania analogowego, cyfrowe przetwarzanie sygnałów zastepuje dotychczasowe przekształcenia ich odpowiednikami w dziedzinie cyfrowej. Dostępne są cyfrowe filtry oraz wzmacniacze. W związku z rozwojem elektroniki wielkość urządzeń przetwarzających sygnały cyfrowe uległa znacznemu zmniejszeniu. Przetworniki i filtry cyfrowe są mniejszych rozmiarów oraz pobierają mniej energii zapewniając lepsze charakterystyki przetwarzania dźwięku. Aby zapewnić dostatecznie dobrą jakość dźwięku należy zagwarantować dostateczną rozdzielczość bitową. Dla podstawowego przetwarzania dźwięków wystarcza przetwornik 13–15 bitowy. W chwili, kiedy synał analogowy zamieniany jest na postać cyfrową jego przetwarzanie staje się zadaniem matematycznym. Możliwe jest zaimplementowanie dowolnego algorytmu czy struktury matematycznej pozwalającej na otrzymanie zamierzonych efektów. Przeprowadzanie takich doświadczeń jest o wiele mniej kosztowne niż w przypadku przetwarzania analogowego, kiedy to każdy test musi zostać przeprowadzony na rzeczywistym sprzęcie. Cyfrowe przetwarzanie sygnałów pozwala na programowanie efektów dźwiękowych dowolnego rodzaju, zapewnia lepszy zakres dynamiki, ogranicza zniekształcenia sygnałów. Zapewnia też znaczące ograniczenie szumów wewnętrznych układu oraz istotnie poprawia dokładność filtrów oraz zwiększa precyzję układów automatycznej regulacji wzmocnienia.

2. Specyfikacja programu yyyKiller

W rozdziale tym zamieszczono założenia użytkowe oraz funkcjonalne programu. Znajduje się tu również opis zaimplementowanych funkcjonalności użytkowych oraz algorytmów wykorzystywanych w aplikacji.

2.1. Zaimplementowane funkcjonalności

Aplikacja tworzona w ramach pracy powinna działać na platformie Android w wersji pozwalającej większości użytkowników na korzystanie z niej. Program ma dostarczać trzy podstawowe funkcjonalności:

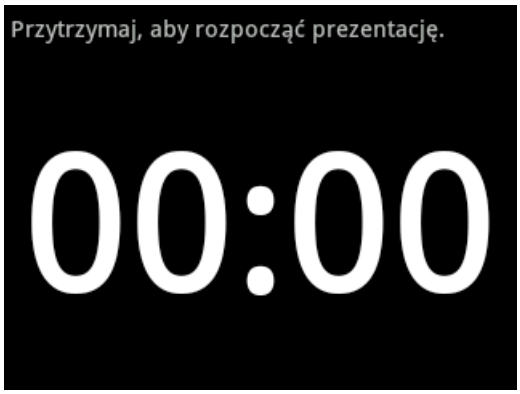
1. Możliwość wykrywania i sygnalizacji artefaktów „yyy", „eee" w trakcie prezentacji.
2. Kontrolę czasu przebiegu prezentacji.
3. Możliwość podsumowania wystąpienia, poprzez zaprezentowanie statystyk oraz czasu jego trwania.

Aplikacja powinna rejestrować wypowiedź użytkownika oraz w czasie rzeczywistym informować go o pojawieniu się niekorzystnych dźwięków - artefaktów. Ważne jest też zapewnienie dyskretnego przekazywania użytkownikowi informacji o wykrytych zdarzeniach. Użytkownik powinien być też informowany o czasie pozostałym do końca prezentacji.

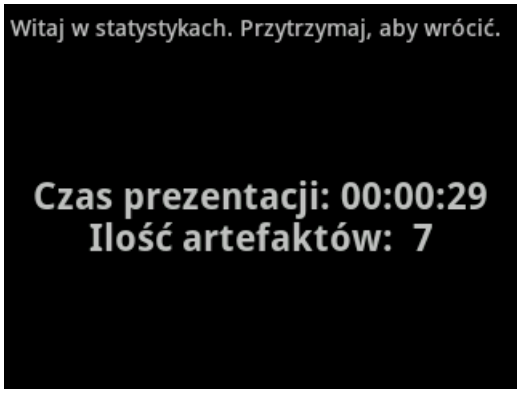
Możliwość wykrywania artefaktów stanowi główną funkcję projektu. Aby zapewnić tą funkcjonalność należało wykonać badania pozwalające stwierdzić jaki algorytm wybrać oraz jak dostroić go do tego rodzaju zagadnienia. Algorytm detekcji artefaktów udało się opracować w oparciu o istniejące sposoby przetwarzania mowy. Należy nadmienić, że został on opracowany w dwóch wariantach. Testowy algorytm przygotowany został w środowisku `Matlab`, następnie przeniesiono go z pewnymi modyfikacjami na platformę Android. Problem wykorzystanych algorytmów oraz sposobu ich konfiguracji w programie został opisany w rozdziałach 2.2 oraz 3.

Użytkownikowi został dostarczony prosty interfejs zapewniający kontrolę czasu prezentacji. Na rysunku 2.1 został przedstawiony ekran startowy aplikacji. Po uruchomieniu programu użytkownik ma możliwość rozpoczęcia analizy prelekcji poprzez przytrzymanie palca na ekranie. Zaimplementowany

został prosty zegar, który odlicza czas od początku prelekcji. Zakończenie odliczania następuje przez ponowne przytrzymanie palca na ekranie. Użytkownik ma wtedy dostęp do statystyk prezentacji, które przedstawiono na rysunku 2.2. Prezentowany jest w nich czas przemówienia oraz liczba zliczonych artefaktów. Szczegółowy opis interakcji użytkownika z aplikacją został przedstawiony w rozdziale 3.2.



Rysunek 2.1: Ekran startowy aplikacji



Rysunek 2.2: Prezentacja statystyk

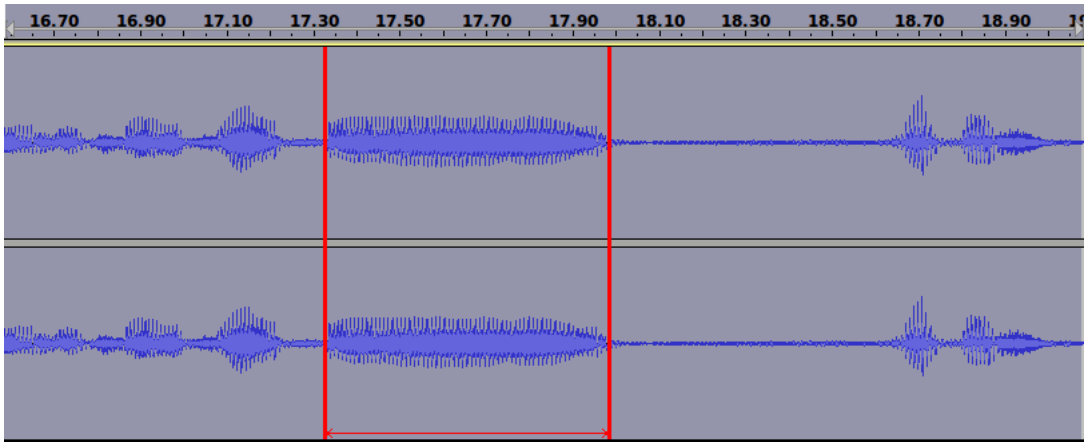
W każdym momencie użytkownik ma możliwość powrotu do ekranu startowego poprzez dotknięcie ekranu. W każdym widoku prezentowane są w górnym pasku instrukcje o możliwych do podjęcia akcjach. Dzięki takiemu zabiegowi interfejs został pozbawiony zbędnych przycisków oraz stał się prostszy i bardziej czytelny. Biorąc pod uwagę fakt, że aplikacja powinna działać w sposób niewidoczny dla słuchaczy, był to pożądany zabieg. Aplikacja stała się przez to bardziej czytelna i przejrzysta oraz prosta w obsłudze.

Detekcja artefaktów możliwa jest po rozpoczęciu prezentacji. Sygnalizacja wykrytych artefaktów może odbywać się na 3 sposoby - za pomocą wibracji, dźwięku oraz zmiany koloru ekranu. Użytkownik ma możliwość dowolnego łączenia ze sobą sposobów sygnalizacji wykrytych dźwięków. W konfiguracji istnieje również możliwość określenia minimalnego czasu trwania dźwięków, które będą wykrywalne przez algorytm. Możliwe wartości zawierają się w przedziale od 400 milisekund do 1,2 sekundy.

2.2. Algorytmy detekcji artefaktów

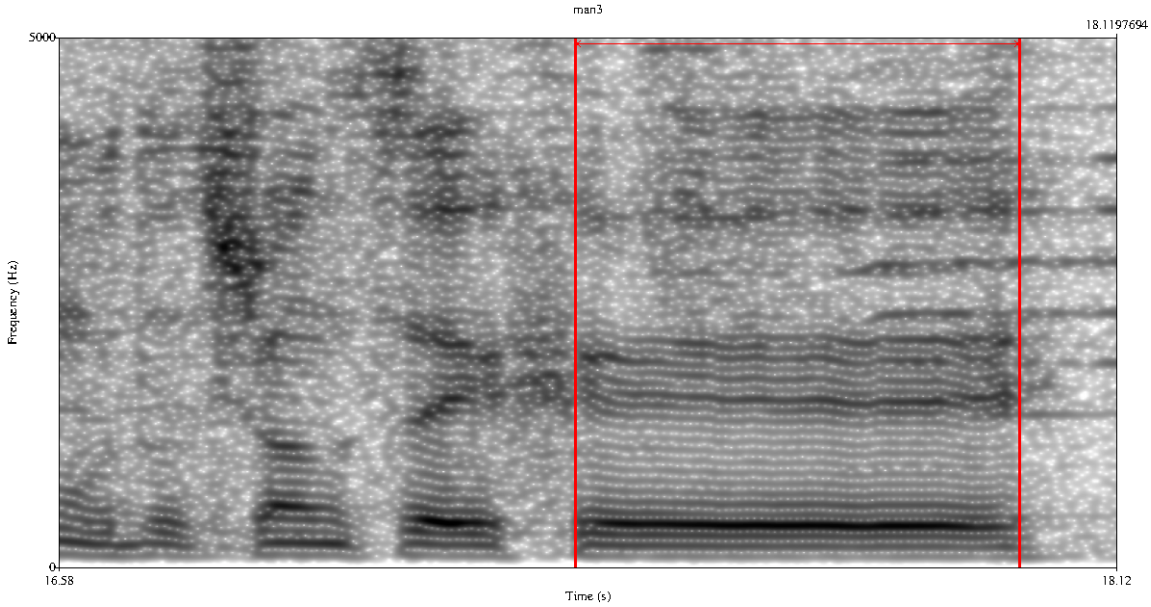
Jak wspomniano w rozdziale 2.1 najważniejszą częścią aplikacji jest algorytm wykrywający artefakty „yyy” oraz „eee”. Zaprojektowanie tego algorytmu wymagało szeregu analiz próbek dźwiękowych zawierających szukane dźwięki, wielu testów oraz następnie przeniesienie go z wersji testowej do produkcyjnej, działającej w aplikacji.

Pierwszym etapem prac było zebranie próbek dźwięków zawierających szukane artefakty. Ważne było w tym wypadku zapewnienie niezależności mówców od siebie. Niespełnienie tego warunku mogłoby doprowadzić do dostrojenia algorytmu do cech jednej osoby, a to spowodowałoby bezużyteczność algorytmu dla innych użytkowników. Zebranych zostało ponad 80 próbek od 10 niezależnych mówców. Wśród nich było 6 mężczyzn oraz 4 kobiety. Każda wypowiedź trwała około 4-5 minut oraz dostarczała ok. 8 artefaktów. Taka baza dźwięków została uznana za wystarczającą do przeprowadzenia badań oraz opracowania algorytmu. Na początku ze zgromadzonych próbek zostały wybrane te, które były interesujace z punktu widzenia algorytmu. Już w tej fazie pracy nad algorytmem udało się zaobserwować na przebiegach, że wartości amplitud fragmentów dźwięku utrzymują się w większości na stałym poziomie (rysunek 2.3). Następnie próbki zostały poddane analizie czasowo-częstotliwościowej. Na rysunku 2.4 został zaprezentowany spektrogram fragmentu wypowiedzi zawierajacy artefakty. Zostały one zaznaczone kolorem czerwonym.



Rysunek 2.3: Przebieg fragmentu mowy zawierający artefakt

Jak można zauważyć fragmenty poszukiwane jako artefakty wykazują też specyficzne właściwości na spektralne. Można je dostrzec na wykresie nie przeprowadzając żadnej głębszej analizy. Widać wyraźnie powtarzające się prążki częstotliwości na całej długości artefaktu. Wnioski te stanowiły podstawę pod dalsze rozważania nad algorytmem. Po wstępnej analizie należało dokonać pewnej modyfikacji założeń programu. Otóż okazuje się, że zawieszenie dźwięku jakim jest artefakt „yyy” czy „eee” w analizie czasowo-amplitudowej oraz czasowo-częstotliwościowej wyglada tak samo jak pozostałe dźwięki. Nie jest zatem możliwe wykrycie która z głosek dźwięcznych pojawiła się w wypowiedzi jako artefakt korzystając z podstawowej analizy częstotliwościowej. Równie dobrze może to być artefakt „ooo”, „uuu”

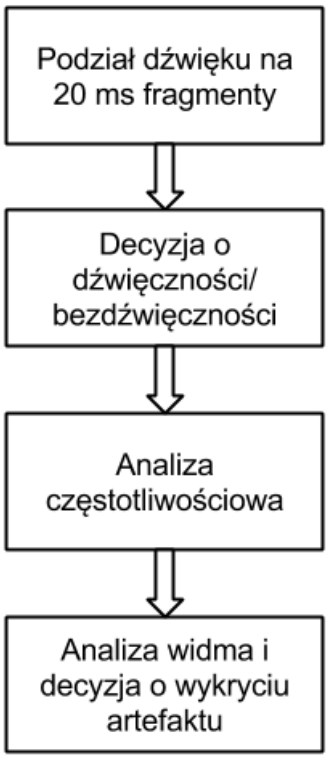


Rysunek 2.4: Spektrogram fragmentu wypowiedzi zawierający artefakt

czy „aaa". Aby rozróżnić takie dźwięki należałoby posłużyć się zaawansowanymi algorytmami przetwarzania mowy, co nie było celem niniejszej pracy.

Dzięki powyższym rozważaniom można stwierdzić, że w algorytmie przydatne będą wyłącznie części wypowiedzi zawierające głoski dźwięczne. Na rysunku 2.5 przedstawiony został schemat ideowy algorytmu detekcji artefaktów, zaimplementowany później za pomocą języka `Matlab` do przeprowadzenia testów na zebranych wcześniej próbkach. Składa się on z trzech głównych bloków. Na początku podejmowana jest decyzja o dźwięczności bądź bezdźwięczności zadanej próbki. Następnie, jeśli próbka jest dźwięczna, przeprowadza się jej analizę częstotliwościową. Dalsze operacje na otrzymanym widmie wynikają z własności dźwięku zawierającego artefakty i nie są oparte na żadnym powszechnie znanym algorytmie.

Do detekcji głosek dźwięcznych wybrany został początkowo algorytm LPC-10. Jest to klasyczny algorytm kompresji sygnału mowy z lat 70-XX wieku. Obecnie niektóre elementy tego algorytmu stosowane są w systemach kompresji mowy oraz sieciach telefonii komórkowej. W takich rozwiązaniach zakłada on, że użyteczna część sygnału mowy zawiera sie w paśmie $300 \div 3500$ Hz [6]. Dlatego też sygnał poddaje się filtracji dolnoprzepustowej filtrem o częstotliwości odcięcia ok. 3500 Hz, a częstotliwość próbkowania 8000 Hz jest często w zupełności wystarczająca. Najczęściej stosuje się próbkowanie od 7 do 16 bitów. Podczas kodowania dźwięk dzielony jest na krótkie fragmenty ok. 20 ms oraz zakłada się, że są one quasi-stacjonarne. Z tak przygotowanych próbek wyznacza się wartości parametrów modelu traktu mowy człowieka. Zaprojektowanie takiego modelu traktu głosowego zapewnia późniejsze zsyntezowanie fragmentów sygnału mowy podobnego do analizowanego. Na rysunku 2.6 został przedstawiony schemat blokowy kodera LPC-10. Jest on uproszczony dla lepszego przedstawienia idei wykorzystania go w algorytmie detekcji artefaktów.



Rysunek 2.5: Schemat ideowy algorytmu detekcji artefaktów

Najpierw sygnał poddawany jest preemfazie, w celu uwydatnienia składowych wyższych częstotliwości. W tym wypadku jest to nierekursywna filtracja FIR. Następnie przefiltrowany sygnał mnożony jest z oknem Hamminga o tej samej długości co fragment próbki dźwiękowej. Tak przygotowany sygnał jest następnie podawany na wejście filtru dolnoprzepustowego o częstotliwości odcięcia 900 Hz, z którego obliczana jest funkcja autokorelacji zgodnie z zależnością 1 [6].

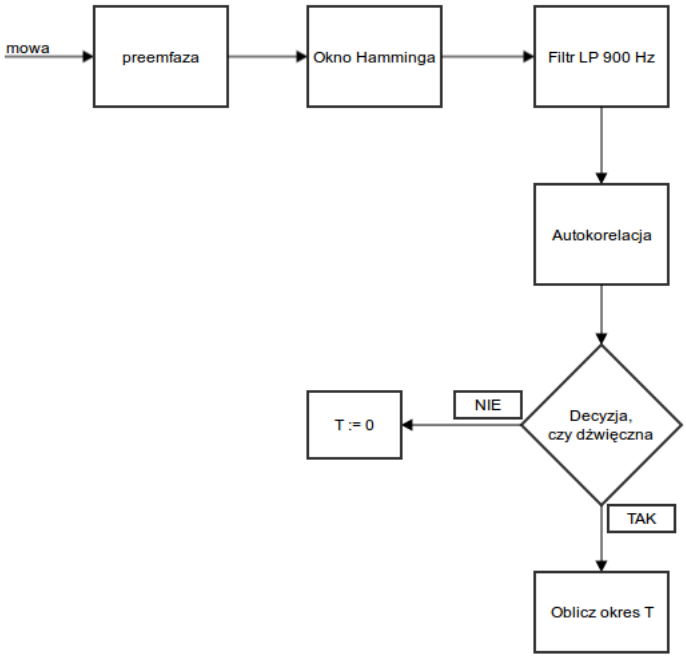
$$r_p(k) = \sum_{n=0}^{N-1-k} s_4(n)s_4(n+k), \quad k = 0, 1, 2, ..., N-1 \tag{1}$$

gdzie $s(n)$ to fragment sygnału powstały z oryginalnej mowy po wcześniejszych przekształceniach.

Następnym elementen na schemacie 2.6 znajduje się jednostka decyzyjna, rozstrzygająca o dźwięczności bądź bezdźwięczności próbki. Jest ona zrealizowana za pomocą zależności 2. Kiedy warunek jest spełniony, analizowany fragment sygnału uznany zostaje jako dźwięczny. Zakres zmienności k jest zależny od częstotliwości próbkowania. Powinien on zawierać się w zakresie tonu podstawowego, czyli około $50 \div 400$ Hz [6]. Nastepnie analizowany jest okres tonu podstawowego oraz tworzony bitowy strumień wyjściowy, jednak nie jest on używany w przypadku tworzonego w niniejszej pracy algorytmu.

$$max(r_p(k)) \geq 0,35 \cdot r(0), \quad k = 20, ..., 160 \tag{2}$$

W projektowanym algorytmie została wykorzystana jedynie część przedstawionego algorytmu. W dalszych pracach posługiwać się będziemy fragmentem pozwalającym na decyzję o dźwięczno-



Rysunek 2.6: Schemat blokowy kodera algorytmu LPC-10

ści bądź bezdźwięczności danego fragmentu. W tablicy 2.2 przedstawiono fragment kodu programu `Matlab` odpowiadający za detekcję głosek dźwięcznych i bezdźwięcznych. Został on wykorzystany do testowania zebranych fragmentów mowy jako element głównego algorytmu.

W późniejszej fazie projektowania aplikacji okazało się, że zaproponowany algorytm posiada zbyt dużą złożoność obliczeniową, aby mógł działać w czasie rzeczywistym. Powodem tego jest wykorzystywanie funkcji autokorelacji w postaci sumy, której obliczenie zajmowało czas około 10 razy dłuższy niż w przypadku innych algorytmów. Co prawda istnieją sposoby obliczania funkcji autokorelacji na podstawie widma sygnału, jednak wiązałoby się to z przeprowadzaniem transformaty Fouriera oraz późniejszego powrotu do dziedziny czasu. W tym przypadku użycie tego algorytmu nie zmniejszyłoby

```
1 for k = 0 : Nhamming-1
2         r(k+1) = sum( bx(1 : Nhamming-k) .* bx(1+k : Nhamming) );
3 end
4     offset = 20;
5     rmax = max(r(offset : Nhamming));
6     imax = find(r == rmax);
7     if ( rmax > 0.35*r(1) )
8     else
9     end
```

Tabela 2.1: Fragment kodu odpowiadający za detekcję głosek dźwięcznych

```
signalLength = length(bx);
currsum = 0;
prevsign = 0;

for i = 1:signalLength
  currsign = sign(bx(i));
  if (currsign > 0 && prevsign < 0) || (currsign < 0 && prevsign > 0)
    currsum = currsum + 1;
  end
  if currsign ~= 0
    prevsign = currsign;
  end
end

return ZCR = currsum;
```

Tabela 2.2: Fragment kodu obliczający wartość współczynnika ZCR

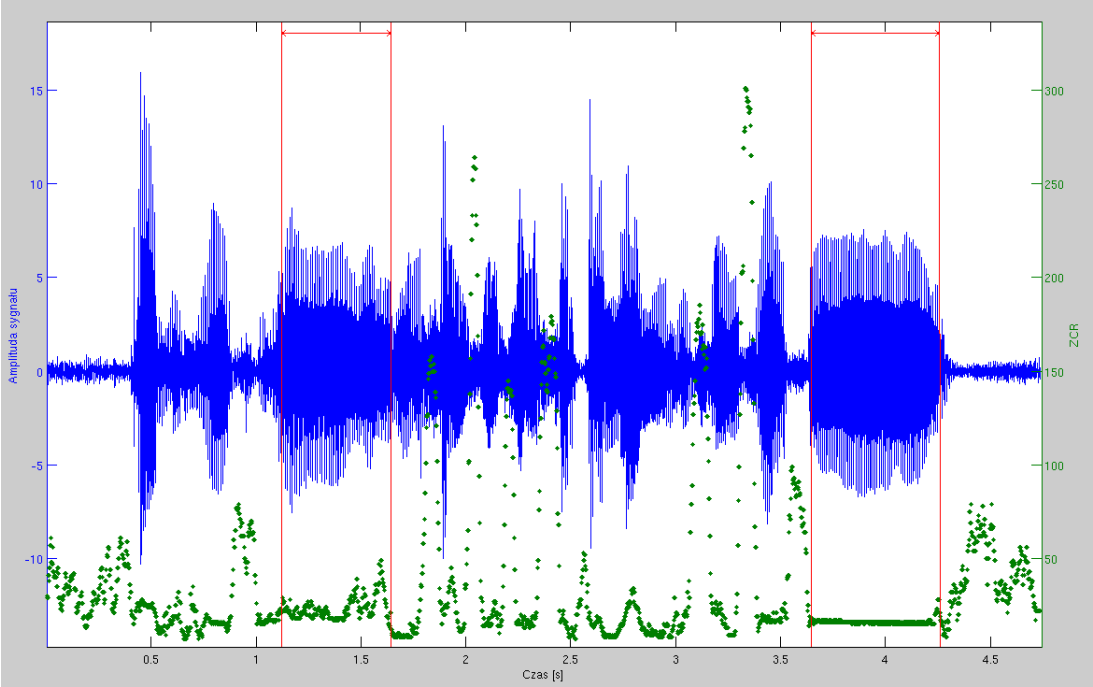
znacznie czasu obliczeń, gdyż należałoby wykonywać transformatę częstotliwościową dodatkowo w tej i kolejnej części algorytmu. Dlatego też wybrany został algorytm przejść przez zero (*zero-crossing algorithm*). Jest to metoda, polegająca na obliczaniu liczby zmian znaku funkcji przy przejściu przez zero. Alogrytm jest powszechnie stosowany do VAD (ang. *Voice Activity Detection*) umożliwiającym wykrywanie aktywności mówcy lub jej braku oraz do wykrywania dźwięczności głosu [1] i klasyfikacji dźwięków [3]. W związku z możliwościami, jakie oferuje algorytm oraz jego prostotą znalazł on zastosowanie m.in w telefonii VoIP oraz GSM i UMTS, pozwalając na oszczędzanie energii w nadajniku i odbiorniku oraz przepływności łączy podczas chwilowej nieaktywności użytkownika. W algorytmie wykorzystuje się parametr zwany *zero-crossing rate* (ZCR) dla określenia ilości przejść przez zero dla badanego sygnału. Definiuje się go zależnością 3. Wielkość ZCR jest ważna dla określenia dźwięczności bądź bezdźwięczności próbki.

$$ZCR = \sum_{k=-\infty}^{k=\infty} |sgn[x(k)] - sgn[x(k-1)]| \tag{3}$$

W 3 algorytm przejść przez zero został wykorzystany w aplikacji. Jest to metoda o małej złożoności obliczeniowej a przy tym bardzo wydajna i efektywna. W tablicy 2.2 przedstawiono fragment kodu programu `Matlab` z zaimplementowanym algorytmem *zero crossing*.

Parametr *zero-crossing rate* jest znacznie wyższy dla sygnałów bezdźwięcznych oraz bardziej zaszumionych. Wrażliwość algorytmu na szum jest niewątpliwie wadą tego algorytmu, gdyż determinuje pewne ograniczenia dla programu, które zostały omówione w rozdziale 4. Rysunek 2.7 prezentuje zależność zmienności wartości ZCR w czasie dla fragmentu wypowiedzi. Widać na nim wyraźny wzrost i duży rozrzut wartości parametru dla zaników sygnału oraz fragmentów bezdźwięcznych. Dla zaznaczonych

kolorem czerwonym artefaktów wartość ZCR jest wyraźnie mniejsza od pięćdziesięciu. Doświadczenia wykazały, że optymalna wartość tego parametru, pozwalająca na eliminację niechcianych dźwięków przy zadowalającym poziomie detekcji artefaktów wynosi 50.

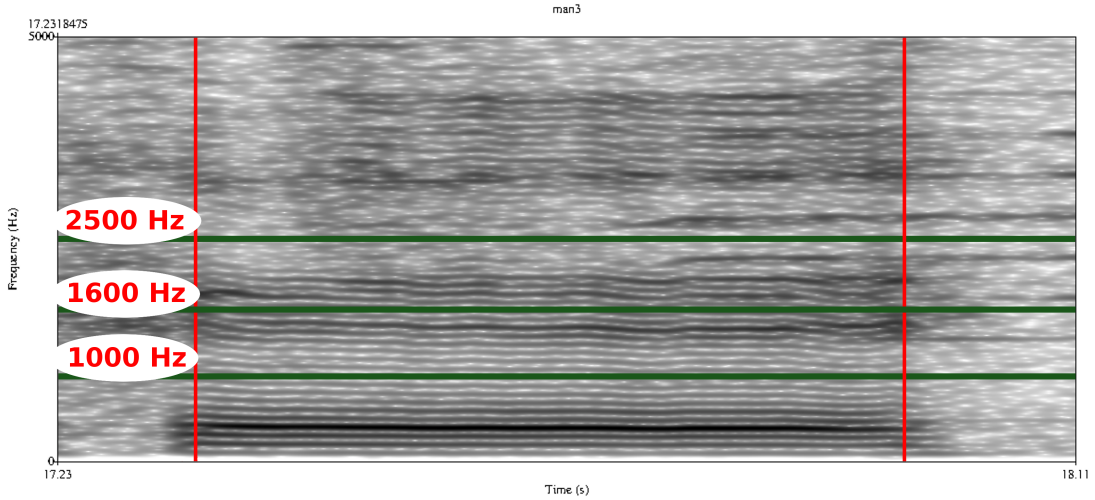


Rysunek 2.7: Zmienność wartości ZCR dla fragmentu dźwięku

Oba opisane do tej pory algorytmy są jedynymi, wykorzystywanymi w pracy, które operują na sygnałach w dziedzienie czasu. W dziedzinie częstotliwości wykorzystywana jest szybka transformata Fouriera. Jej złożoność obliczeniowa jest znacznie mniejsza, niż złożoność klasycznej dyskretnej transtormacji Fouriera i wynosi $Nlog_2(N)$ [6], gdzie N jest ilością próbek. FFT wymaga, aby analizowany sygnał składał się z $N = 2^p$ próbek (gdzie p jest liczbą naturalną).

Dzięki zastosowaniu powyższych algorytmów możliwe było przeprowadzenie testów zebranych wcześniej próbek. Badanie zależności między nimi opierało się o bardziej szczegółowy algorytm przedstawiony na rysunku 2.9. Powstał on w oparciu o analizowane widma fragmentów mowy. Jak wspomniano wcześniej, analiza niniejszym algorytmem opiera się na 20 ms fragmentach mowy. Są to jednak zbyt małe fragmenty, aby móc stwierdzić, czy próbka zawiera szukany artefakt czy też nie. Ponadto, przeprowadzając analizę częstotliwościową na próbkach 20 ms, otrzymujemy niezadowalającą rozdzielczość widma. W związku z tym w algorytmie poszukiwane są około 200 ms ciągi fragmentów dźwięcznych. Badania wykazały, że przy zastosowaniu zarówno algorytmu LPC-10, jak również algorytmu przejść przez zero próbki szukanych artefaktów nie zawierały żadnych fragmentów bezdźwięcznych. Daje to podstawę sądzić, że wyszukiwanie 200 ms dźwięcznych fragmentów dźwięku nie spowoduje błędów w detekcji artefaktów.

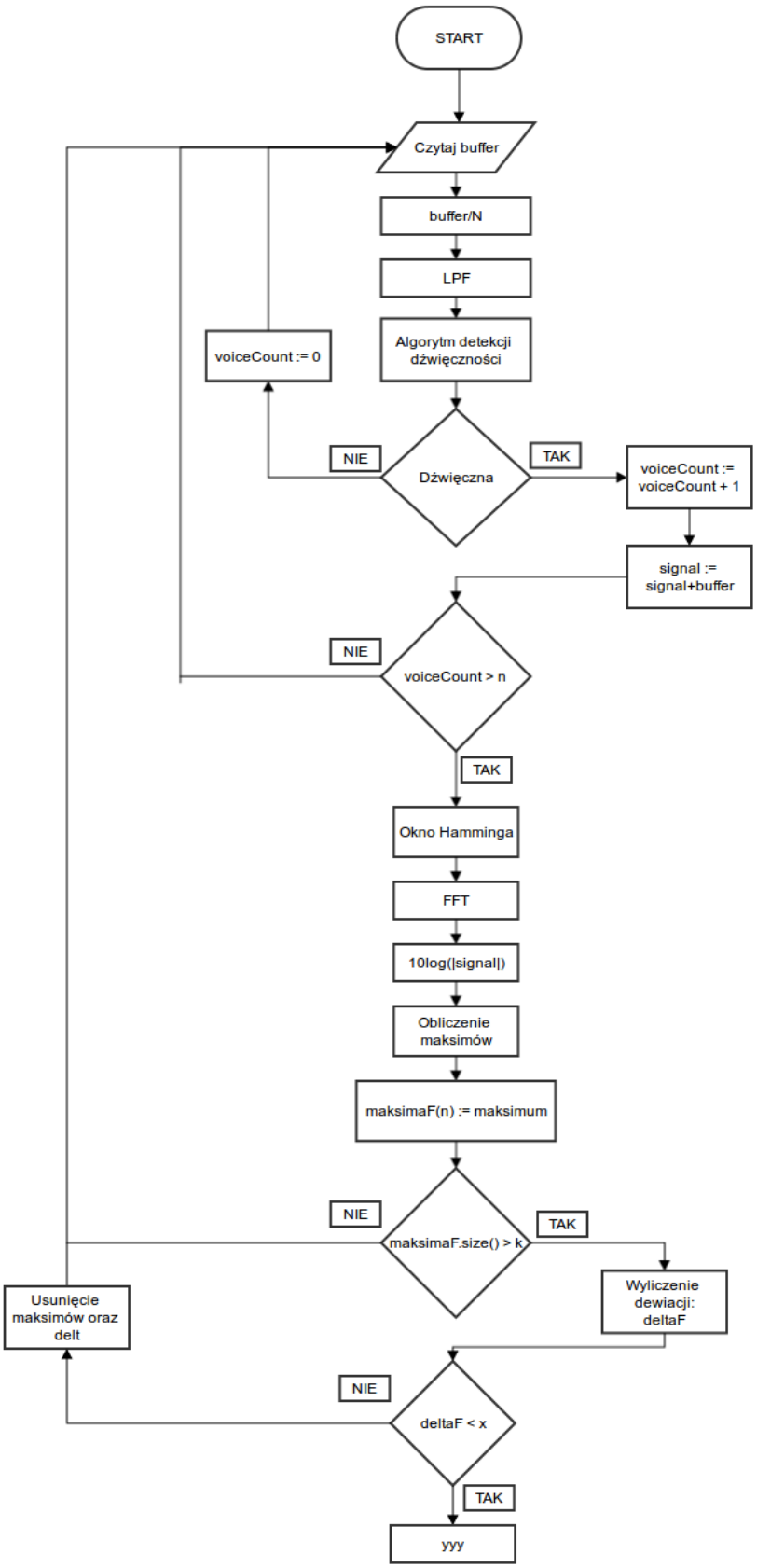
W trakcie zliczania próbki są przechowywane w pamięci, do momentu wystąpienia głoski bezdźwięcznej lub zliczenia do 200 ms i przeprowadzenia na niej FFT. Następnie na uzyskanym widmie do-



Rysunek 2.8: Zakresy detekcji maksimów

konywana jest najistotniejsza analiza. Na rysunku 2.8 przedstawiono fragment dźwięku zawierający artefakt „yyy" podzielony na podpasma. Po analizie zgromadzonych próbek stwierdzono, że zdecydowana większość z nich zawierała charakterystyczne maksima w trzech zaznaczonych na rysunku pasmach. Pierwszy analizowany zakres to $0 \div 1000$ Hz, drugi $1000 \div 1600$ Hz, oraz trzeci $1600 \div 2500$ Hz. W tych trzech zakresach poszukiwane są maksima widm znalezionych wcześniej 200 ms dźwięcznych fragmentów mowy. Każde z tych maksimów jest zapisywane i porównywane z kolejnym znalezionym fragmentem dźwięcznym. Ważne jest, aby między nimi nie było żadnych fragmentów bezdźwięcznych, gdyż wtedy algorytm ulegnie wyzerowaniu i rozpocznie obliczenia od nowa.

Jeżeli kolejna próbka będzie dźwięczna, oraz przeprowadzona będzie na niej analiza FFT i wyliczone zostaną maksima następuje porównanie z poprzednią próbką. W tym celu w każdym podanym wcześniej paśmie obliczana jest średnia maksimów, a następnie ich odchyłka od średniej wartości. Z przeprowadzonych testów wywnioskowano, że najlepszą efektywność algorytmu uzyskuje się przy odchyleniu około 50 Hz. Jeżeli tak obliczone maksima we wszystkich trzech zakresach znajdą się w zadanym przedziale odchylenia od wartości średniej, próbka zostaje uznana za zawierajacą szukany artefakt. Tak sformułowany algorytm pozwala na detekcję artefaktów o minimalnej długości około 400 ms. Modyfikacje ilości zebranych próbek pozwalają na zwiększenie tego okresu, jednakże nie jest możliwe jego skrócenie. Wszelkie próby zmniejszania wartości 400 ms skutkowały rozregulowaniem algorytmu, oraz uznawaniem fałszywych próbek jak szukane, tzw. *false positive*. Jest to zjawisko niepożądane, a jego wyeliminowanie pozwoliło na uzyskanie dobrej sprawności algorytmu.



Rysunek 2.9: Schemat algorytmu detekcji artefaktów

3. Implementacja

W niniejszym rozdziale przedstawiono opis platformy na jakiej zaimplementowana została aplikacja oraz rodzaj urządzenia, na którym przeprowadzono testy. Obejmuje on też przebieg procesu implementacji aplikacji.

3.1. Opis platformy

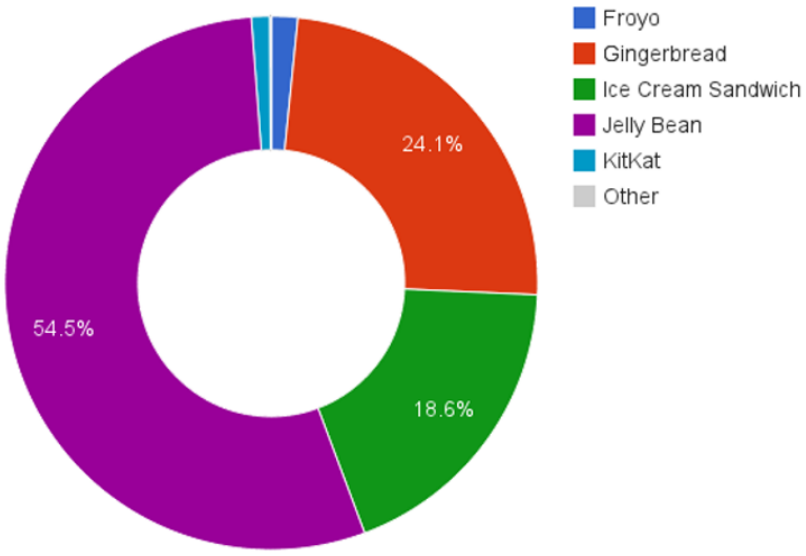
Rynek urządzeń mobilnych jest obecnie bardzo zróżnicowany. Użytkownicy mają do wyboru urządzenia w bardzo szerokim zakresie cenowym oraz oferujący szeroki wachlarz funkcjonalności. Telefon komórkowy juz dawno przestał pełnić wyłącznie funkcję porozumiewawczą. Dlatego ważne jest, aby współczesne urządzenia dostarczały możliwość wykonywania zdjęć, nagrywania filmów wideo, komunikacji internetowej oraz rozrywkę. Wśród firm produkujących urządzenia mobilne możemy wymienić takich gigantów, jak Apple, Samsung, Google, Microsoft, HTC czy Nokia. Wielu producentów zdecydowało się na stworzenie własnego systemu operacyjnego dla urządzeń przez siebie oferowanych. Jednak większa część korzysta z gotowych systemów. Według danych z trzeciego kwartału 2013 roku [9] na rynku dominuje platforma Android, z udziałem 81,9%. Na kolejnych miejscach plasują się kolejne dwa duże systemy: iOS firmy Apple - 12,1% oraz Microsoft - 3,6% udziałów w sprzedaży. Należy zaznaczyć, że w porównaniu do poprzednich lat Android oraz Microsoft zanotowały największy wzrost popularności.

Z wielu mobilnych systemów operacyjnych znajdujących się obecnie na rynku do zaimplementowania aplikacji został wybrany Android. Powodem decyzji była rosnąca popularność tej platformy oraz łatwy dostęp do sprzętu. Prostota implementacji aplikacji w tym systemie jest również jego mocnym atutem. Od 2005 roku Android Inc. jest rozwijany przez Google Inc. System wspiera takie platformy jak: ARM EABI, Intel x86 oraz MIPS. Jest on oparty na zmodyfikowanym, monolitycznym jądrze systemu Linux oraz udostępniany w oparciu o licencję Apache License 2.0. Android został dedykowany głównie dla urządzeń wyposażonych w ekrany dotykowe, takich jak smartfony oraz tablety. Urządzenia oparte na tym systemie pozwalają na komunikację nie tylko za pomocą gestów wykonywanych na ekranie ale też głosu czy ruchu. Większość smartfonów jest wyposażona kamerę wideo, GPS, czujniki położenia, akcelerometr, czy żyroskop. Dzięki tym czujnikom urządzenia mogą dostarczać coraz więcej funkcjonalności oraz upraszczać ich obsługę. Do najnowszych urządzeń producenci dodają również czujniki

ciśnienia, odległości, magnetometry czy termometry. Wszystkie te dodatki mają na celu zwiększenie możliwości oferowanych przez urządzenia, oraz poszerzyć zakres funkcjonalności systemów.

Od momentu wydania przez Google pierwszej wersji beta systemu w 2007 roku Android przeżył gwałtowny wzrost popularności. Ciągle wydawane są coraz nowsze wersje systemu. Obecnie na rynek wkracza *Android 4.4 Kit Kat*. Jednak tak szybki rozwój spowodował ogromne zróżnicowanie wśród użytkowników. Diagram 3.1 przedstawia rozkład popularności systemu z dnia 2.12.2013 roku [10]. Wynika z niego, że mimo pojawiających się coraz nowszych wersji Androida około 25% użytkowników Androida wciąż korzysta z wersji 2.3.3 wydanej w 2011 roku. Popularność pozostałych wersji platformy jest również nierównomierna. Ponad 18% użytkowników korzysta z wydania 4.0, około 37% z 4.1 a prawie 13% z 4.2. Tylko niewiele ponad 1% klientów używa najnowszej wersji systemu. Powodem takiego zróżnicowania jest liberalne podejście firmy Google do licencjonowania swojego systemu. Między innymi, nie nakłada ona na producentów urządzeń obowiązku aktualizacji oprogramowania do najnowszej wersji.

Wersja	Nazwa	API	Popularność
2.2	Froyo	8	1.6%
2.3.3 - 2.3.7	Gingerbread	10	24.1%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	18.6%
4.1.x	Jelly Bean	16	37.4%
4.2.x		17	12.9%
4.3		18	4.2%
4.4	KitKat	19	1.1%

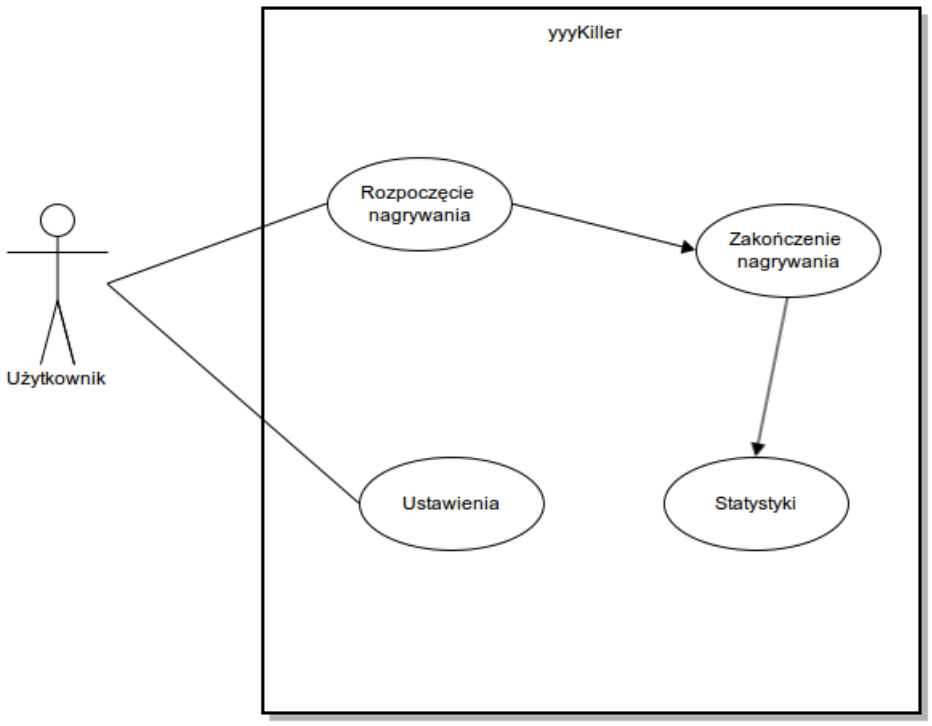


Rysunek 3.1: Rozkład popularności wersji systemu Android

Taki obraz popularności API Androida powoduje pewne problemy programistyczne, gdyż należy na początku prac nad projektem określić z jakiej minimalnej wersji systemu będziemy korzystać. Wciąż duża popularność API 10 powoduje, że pomimo nowych funkcjonalności dostarczanych przez system należy uwzględnić tych konsumentów, którzy pozostają przy starszych wersjach oprogramowania. Przed rozpoczęciem projektowania aplikacji warto więc przeanalizować wymagania systemowe dla aplikacji i na tej podstawie wybrać możliwie najniższą wersję SDK (Software Development Kit) Androida. W projekcie opisywanym w tej pracy wybrano SDK w wersji 2.3.3 (API 10). Było to spowodowane wykorzystaniem metod klasy `TimeUnit` konwertujących czas [8], które dostępne są dopiero od wersji API 10.

3.2. Implementacja aplikacji yyyKiller

Po wybraniu wersji API, które zostało użyte do wykonania programu rozpoczął się proces jego implementacji. Przebiegał on w oparciu o diagram klas (rysunek 3.4) oraz diagram przypadków użycia (rysunek 3.2). Pierwszym schematem, który został zdefiniowany był diagram przypadków użycia. Opisuje on system z punktu widzenia przyszłego użytkownika (aktora) oraz określa wymagania konieczne oraz opcjonalne systemu. Jak widać na schemacie, aktor w każdym momencie działania aplikacji ma możliwość zmiany ustawień dotyczących sposobu sygnalizacji czy czułości algorytmu. Użytkownik może rozpocząć prezentację, co spowoduje uruchomienie algorytmu oraz zegara liczącego czas. Po zakończeniu prezentacji możliwe są dwie akcje: przejście do ekranu startowego albo przejście do ekranu prezentacji statystyk. W przypadku przejścia do początku aplikacji następuje usunięcie wszystkich danych dotyczących statystyk użytkownika oraz algorytmu. Przejście do statystyk jest związane z wyświetleniem nowej aktywności. Z tego widoku można już tylko powrócić do ekranu startowego programu.



Rysunek 3.2: Diagram przypadków użycia aplikacji

Diagram klas jest jednym z najważniejszych składników dokumentacji projektu. Zawiera informacje o statycznych związkach pomiędzy elementami [2]. Był szczególnie pomocny w procesie implementacji, gdyż pozwolił na wyeliminowanie wielu błędów na etapie projektowania programu. Program składa się z pięciu klas, które reprezentują jego aktywności. Klasa `WelcomeScreen` reprezentuje powitalne okno aplikacji. Zawiera animację logotypu Akademii Górniczo-Hutniczej oraz logo aplikacji. Ekran pojawia

Rysunek 3.3: Widoki ekranu powitalnego aplikacji

się przy starcie aplikacji i wyświetla się przez 3 sekundy. W tym czasie nie są możliwe żadne akcje poza zminimalizowaniem programu. Na rysunku 3.3 przedstawione zostały widoki tej aktywności.

Klasa `Statistics` reprezentuje obiekt statystyk, które są gromadzone podczas prezentacji. Zawiera dwa pola: `presentationTime` oraz `artifactCount`. Każde uruchomienie algorytmu powoduje stworzenie obiektu `Statistics`, który nastepnie przekazywany jest do aktywności reprezentowanej przez klasę `StatisticsActivity`. Statystyki prezentują całkowity czas trwania prezentacji oraz ilość zliczonych w tym czasie artefaktów. Widok ekranu statystyk przedstawiono na rysunku 2.2.

Klasa `RecordActivity` stanowi najważniejszą część programu. Jest odpowiedzialna za prezentowanie głównej aktywności, w której użytkownik przeprowadza prelekcję. Klasa zawiera zmienne i metody odpowiedzialne za zmianę wyglądu ekranu oraz przede wszystkim wewnętrzną klasę `Looper`. W momencie startu prelekcji klasa `RecordActivity` tworzy wątek `Looper`, którego zadaniem jest nagrywanie oraz analiza otrzymanego dźwięku. Został w nim zaimplementowany algorytm opisany w rozdziale 2.2. Jeżeli algorytm uzna próbkę za artefakt, przesyła od klasy głównej wiadomość. Sygnał ten jest przekazywany za pomocą obiektu klasy `Handler`. Jest to natywna klasa pakietu `android.os` i poza przesyłaniem komunikatów miedzy wątkami umożliwia planowanie zadań w przyszłości. W konstruktorze tej klasy nadpisana została metoda `handlerMessage(Message msg)`, która steruje akcjami podejmowanymi po pozytywnej decyzji algorytmu. Taki zabieg pozwolił na odseparowanie algorytmu od zachowań aplikacji. W zależności od bieżących ustawień programu możliwa jest wibracja, sygnał dźwiękowy lub zmiana koloru ekranu. Metoda `handlerMessage` inkrementuje również obiekt `Statistics.artifactCount` zliczający ilość wykrytych artefaktów. W tablicy 3.2 znajduje się fragment kodu, w którym zaimplementowany został opisany obiekt `Handler`

3.3. Algorytm zaimplementowany w aplikacji

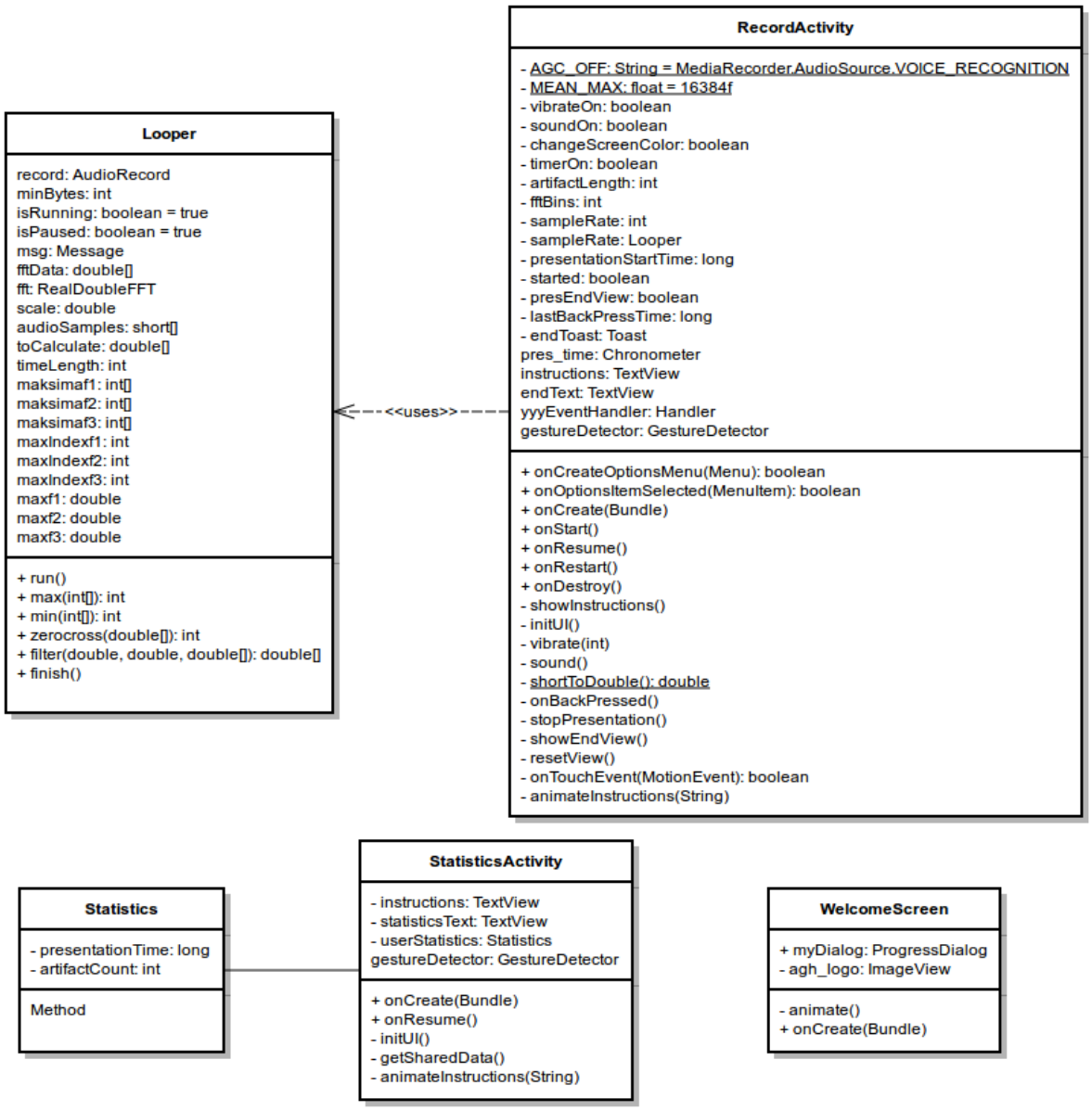
Do nagrywania dźwięku został wykorzystany obiekt klasy `android.media.AudioRecord`, który lepiej zachowuje się przy przetwarzaniu dźwięku w czasie rzeczywistym niż druga klasa wykorzystywana do nagrywanie dźwięku - `android.media.MediaRecorder`. Na potrzeby tworzenia

```
1  Handler yyyEventHandler = new Handler() {
2      @Override
3      public void handleMessage(Message msg) {
4
5          if (samplingThread.isAlive()) {
6              userStatistics.setArtifactCount(userStatistics.getArtifactCount()
7                                      + 1);
8              userStatistics.getArtifactTime().add(SystemClock.elapsedRealtime()
9                              - pres_time.getBase());
10
11             if (vibrateOn == true) {
12                 vibrate(300);
13                 Log.i(TAG_ARTIFACT_DETECT, "vibrate alarm");
14             }
15             if (soundOn == true) {
16                 sound();
17                 Log.i(TAG_ARTIFACT_DETECT, "sound alarm");
18             }
19             if (changeScreenColor == true) {
20                 getWindow().getDecorView().setBackgroundColor(Color.YELLOW);
21                 new CountDownTimer(200, 1) {
22                     public void onTick(long millisUntilFinished) {
23
24                     }
25                     public void onFinish() {
26                         if (samplingThread.isAlive()) {
27                             getWindow().getDecorView().setBackgroundColor(
28                                             Color.BLACK);
29                         }
30
31                     }
32                 }.start();
33                 Log.i(TAG_ARTIFACT_DETECT, "screen alarm");
34             }
35         }
36     }
37 };
```

Tabela 3.1: Implementacja obiektu `Handler`

obiektu `AudioRecord` należy określić minimalny bufor, który pozwoli pomyślnie stworzyć obiekt nagrywajacy. Rozmiar bufora jest określany na podstawie częstotliwości próbkowania, zastosowanego kodowania oraz ilości zadeklarowanych kanałów. W momencie uruchomienia wątku tworzony jest obiekt klasy `RealDoubleFFT`, który dostarcza metody umożliwiające przeprowadzenie analizy częstotliwościowej sygnału. W wątku rozpoczyna się nagrywanie oraz analiza dźwięku. Pobrany bufor jest analizowany pod kątem dźwięczności, a następnie poddawany jest analizie częstotliwościowej. Do wykonywania przekształceń czasowo-częstotliwościowych wykorzystano bibliotekę *jfftpack*, która jest wersją biblioteki *fftpack* w języku Java. Została ona zaimplementowana w ramach projektu SCUBA-2 przez pracowników UK Astronomy Technology Centre i jest dostępna w ramach licencji Apache.

Dalszy przebieg algorytmu jest taki sam jak opisany w rozdziale 2.2. W aplikacji został jeszcze zaimplementowany algorytm przejść przez zero (3).



Rysunek 3.4: Diagram klas aplikacji

4. Testy

Niniejszy rozdział zawiera opis i wyniki testów przeprowadzonych zarówno na próbkach dźwięku zebranych w fazie projektowej jak również testy wykonane za pomocą zaprojektowanej aplikacji.

4.1. Testy we współczesnych projektach informatycznych

Testy stanowią współcześnie podstawę wytwarzania oprogramowania. W inżynieria oprogramowania definiuje się kilka faz procesu powstawania projektu. Są to: określenie wymagań, kwalifikacja komponentów, projektowanie architektury, dostosowanie komponentów, integracja i testowanie, wdrożenie [4]. Oczywiście jest to schemat obowiązujący dla dużych projektów, składających się z wielu modułów i podprogramów. Warto jednak podkreślić, jak istotnym elementem tego procesu jest testowanie. Istnieje również takie pojęcie jak programowanie ekstremalne, które polega na odwróceniu procesu implementacji. Po zaprojektowaniu aplikacji tworzone są testy jej poprawności, natomiast później następuje implementacja samego programu w oparciu o stworzone wcześniej testy. Pozwala to na wykrycie wielu błędów już w fazie implementacyjnej co tym samym przyspiesza proces wytwarzania programu oraz czyni go tańszym i bardziej wydajnym.

W niniejszej pracy przedstawiono częściowe testy programu. Najwięcej ich zostało wykonanych w czasie projektowania algorytmu, gdyż każda zmiana musiała być zweryfikowana na zgromadzonych wcześniej próbkach. Częściowe testy zostały również przeprowadzone na zaimplementowanej już aplikacji. Ich przebieg oraz wynik został opisany w niniejszym rozdziale.

4.2. Wydajność algorytmu detekcji artefaktów

W tym podrozdziale zostały opisane testy przeprowadzone podczas projektowania algorytmu detekcji artefaktów. Opisano tu tylko część z nich, gdyż jak wcześniej wspomniano każda zmiana w algorytmie inicjowała przeprowadzenie tego procesu.

Ważnym momentem w czasie implementacji algorytmu w programie `Matlab` była zmiana sposobu wykrywania dźwięczności. Z powodu zbyt długiego czasu przetwarzania próbek w czasie rzeczywistym zamieniony został alborytm LPC-10 na algorytm przejść przez zero. Uzasadnienie tej zmiany uzyskano w teście przeprowadzonym na całych próbkach wypowiedzi. Przeprowadzone badanie wykazało, iż czas przetwarzania dźwięków w nowym algorytmie zmalał prawie 35-krotnie, co w aplikacjach przetwarzania

czasu rzeczywistego jest priorytetem. Program powinien działać w sposób ciągły, bez opóźnień oraz nie powodować gwałtownego rozładowania baterii urządzenia. Zmiana spowodowała dodatkową korzyść. W schemacie z zastosowaniem przetwarzania algorytmem LPC-10 udało się uzyskać skuteczność algorytmu na poziomie 72% przy bardzo dużym współczynniku *false positive*, czyli nieprawidłowych wykryć na poziomie 60%. Taki algorytm był nie do przyjęcia w aplikacji. Po zmianie współczynniki uległy znaczącej poprawie. Skuteczność algorytmu wzrosła do niemal 90%, natomiast współczynnik *false positive* zmalał do 15%. Taki poziom błędów oraz skuteczności jest do przyjęcia w projektowej fazie projektu.

4.3. Konfiguracja i testy użytkowe aplikacji

Sama aplikacja również podlegała testom. Po przeprowadzeniu testów wydajności oraz poprawności działania przyszedł czas na test z udziałem użytkowników. Program został zainstalowany na kilku telefonach oraz użyty podczas kilku wystąpień. Skonfigurowano go tak, aby wykrywał jak najkrótsze artefakty, czyli około 400 ms oraz uruchomiono z zestawem słuchawkowym oraz bez niego. Użytkownicy programu z zestawem słuchawkowym osiągali lepsze rezultaty, niż Ci bez niego. Okazało się, iż w związku z zastosowaniem algorytmu przejść przez zero do detekcji dźwięczności aplikacja stała się mniej odporna na szum. Problem ten był opisywany w rozdziale 2. Jednak mimo konieczności stosowania zewnętrznego mikrofonu funkcjonalność aplikacji nie jest w żaden sposób ograniczona a dodatkowo zyskujemy pewność, że wykrywane dźwięki na pewno pochodzą od prelegenta. Reakcja mówcy, na wykryte artefakty jest taka, jak przewidywano wcześniej, podczas założeń programu. Już po kilku wykrytych artefaktach osoba prowadząca prezentację stara się zwracać uwagę na popełniane błędy i od razu je koryguje.

Wykonano również test użytkowy z wykrywaniem artefaktów o długości 1,2 s. Okazało się, że aplikacja była mniej wrażliwa na dźwięki zakłócające oraz zdecydowanie rzadziej się myliła. Jednak nie wykrywała krótkich artefaktów. Taki tryb pracy jest wskazany dla osób z nawykiem długiego zawieszania głosu. W czasie testów okazało się również, że program wykrywa nie tylko głoski „yyy” oraz „eee” ale także pozostałe głoski dźwięczne, np. o, i, a, u. Jest to dodatkowy atut opracowanego algorytmu, który czyni aplikację możliwą do wykorzystywania również przez osoby mające problem z zawieszaniem głosu na innych głoskach dźwięcznych.

Wnioski końcowe

W przedstawionej pracy zaimplementowana została aplikacja mobilna ułatwiająca eliminację nawyku mówienia „yyy" oraz „eee" podczas prezentacji. W tym celu przeprowadzono badania, mające na celu określenie kierunku poszukiwań rozwiązania problemu automatycznego wykrywania szukanych artefaktów. Wykonane badania pozwoliły najpierw sformułować podstawowy zarys przebiegu procesu projektowania algorytmu, który stał się rdzeniem tworzonego programu. W kolejnych krokach udało się zaprojektować algorytm, który coraz lepiej spełniał swoją funkcję i pozwalał na osiągnięcie zadawalających wyników. Pierwotny algorytm zaimplementowany został w środowisku `Matlab`, tam również odbyły się testy, które pozwoliły na przeniesienie go później do właściwej aplikacji. Proces projektowania algorytmu był najdłuższą i najbardziej pracochłonną częścią niniejszej pracy. Wymagał analizy ponad 80 próbek 10 niezależnych od siebie mówców.

Pogram, który jest rezultatem niniejszej pracy został zaimplementowany w środowisku Android, z zastosowaniem tzw. dobrych praktyk programowania ułatwiających uzyskanie jak najwyższej jakości kodu źródłowego. W języku Java zaimplementowana została logiczna część aplikacji, natomiast język XML posłużył do implementacji interfejsu użytkownika aplikacji. Mimo wcześniejszych obaw o niewystarczającą szybkość języka Java w zastosowaniach przetwarzania sygnałów okazał sie on wystarczająco wydajny w tego typu aplikacji, zwłaszcza po zastosowaniu zoptymalizowanych bibliotek do transformacji Fouriera.

Należy też zaznaczyć że wybrany sposób detekcji artefaktów był jednym z wielu możliwych. Jest to tylko przykładowy algorytm, który w przyszłości można rozbudować lub zmodyfikować. Użyteczne mogą stać się metody cepstralne przetwarzania sygnałów, możliwość korzystania z formantów, które powszechnie wykorzystuje się w syntezie mowy, oraz wiele innych metod i algorytmów wykorzystywanych w cyfrowym przetwarzaniu sygnałów. Również implementacja samej aplikacji spowodowała, iż pojawiło się wiele pomysłów nie będących ściśle związanych z problemem niniejszej pracy ale związanych z wspieraniem prelegentów podczas wygłaszania swoich przemówień. I tak np. jednym z pomysłów jest implementacja zegara liczącego w tył, tak aby użytkownik miał możliwość lepszego kontrolowania swojej prezentacji.

Implementacja aplikacji, a w szczególności algorytmu detekcji artefaktów w tak uniwersalnym i przenośnym języku jakim jest Java powoduje, że może stać się podstawą do stworzenia podobnych aplikacji działających na różnych platformach, które zapewniają wsparcie dla tego języka. Umożliwia to dalszy rozwój programu i powoduje, że w przyszłości może się stać dostępny dostępny dla szerszego

grona użytkowników platform mobilnych oraz aplikacji uruchamianych na tradycyjnych komputerach klasy PC.

Bibliografia

[1] Aye Yin Yin „Speech Recognition Using Zero-Crossing Features”, 2009

[2] Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John „Design Patterns: Elements of Reusable Object-Oriented Software”

[3] „On the use of zero-crossing rate for an application of classification of percusive sounds” Gouyon Fabien, Pachet François, Olivier Delerue Verona, Italy 7-9.12.2000

[4] Sacha Krzysztof „Inżynieria oprogramowania”, Wydawnictwo Naukowe PWN, 2010

[5] Szczeniowski Szczepan „Fizyka doświadczalna. Mechanika i akustyka” PWN, Warszawa 1980

[6] Zieliński Tomasz P „Cyfrowe przetwarzanie sygnałów” Wydawnictwa Komunikacji i Łączności WKŁ, 2012

[7] http://www.pgi.gov.pl/pl/oddzial-geologii-morza-home/gdansk/aktualnosci-geologia-morza/398-trudne-badania-metody-geologii-morza.html

[8] http://developer.android.com/reference/java/util/concurrent/TimeUnit.html

[9] http://www.gartner.com/newsroom/id/2623415

[10] http://developer.android.com/about/dashboards/index.html

[11] http://www.doz.pl/newsy/a1505-Ucho_pod_kontrola