

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Elektronika (EKA)
SPECJALNOŚĆ: Inżynieria akustyczna (EIA)

**PRACA DYPLOMOWA
INŻYNIERSKA**

Narzędzie do detekcji i ekstrakcji zdarzeń
akustycznych w materiale audio

A tool for detecting and extracting acoustic
events in an audio material

AUTOR:
Konrad Kamil Janowski

PROWADZĄCY PRACĘ:
dr Maciej Walczyński, Katedra Akustyki i Mul-
timediów (W4/K5)

OCENA PRACY:

*Dla Rodziców, którzy zawsze mnie
wspierali i pozwolili mi rozwinąć
się w kierunku, który pragnąłem.*

Spis treści

1	Wstęp	3
1.1	Cel i zakres pracy	3
1.2	Motywacja	3
2	Wprowadzenie	5
2.1	Zdarzenie akustyczne	5
2.2	Przyjęta definicja zdarzenia akustycznego	5
2.3	Metoda wykrywania zakłóceń	7
2.4	Istniejące rozwiązania do wykrywania zdarzeń akustycznych	8
3	Metodologia	9
3.1	System kontroli wersji - Git, GitHub, GitKraken	9
3.1.1	System kontroli wersji	9
3.1.2	Git	10
3.1.3	GitKraken	10
3.2	Język programowania - Python	11
3.3	Środowisko programistyczne - Pycharm	12
3.4	Plik z informacjami o zdarzeniach: JSON	12
4	Struktura oraz działanie programu	13
4.1	Sposób działania programu - aplikacja konsolowa	13
4.2	Struktura programu	13
5	Szczegółowe omówienie funkcji w programie	15
5.1	Odnajdywanie plików wave	16
5.2	Odczytywanie pliku Wave	17
5.3	Konwersja pliku	17
5.4	Detekcja i organizacja zdarzeń	21
5.5	Zapis pliku .json oraz plików wave	23
5.5.1	JsonEventsWriter	23
5.5.2	Ekstrakcja z plików wav	23
6	Sposób użycia	25
6.1	Zachowanie podczas normalnego działania	27
7	Testy	29
	Bibliografia	29

Rozdział 1

Wstęp

1.1 Cel i zakres pracy

Przedmiotem pracy jest oprogramowanie do detekcji i ekstrakcji zdarzeń akustycznych w materiale audio. Oprogramowanie zostało stworzone do obsługi plików w formacie wave o rozdzielczości bitowej szesnaście(16) i dwadzieścia cztery(24) bity oraz częstotliwości próbkowania czterdzieści cztery i dziesięć setnych (44,1) kilo Herca i czterdzieści osiem (48) kilo Herca. Oprogramowanie zostało przetestowane używając plików o tych parametrach. Oprogramowanie zostało stworzone modularnie aby możliwy był jego dalszy rozwój. Z powodu tego założenia, program ma możliwość obsługi dowolnej częstotliwości próbkowania oraz dowolnej rozdzielczości bitowej choć nie jest to zalecane. Praca w swojej logicznej strukturze zawiera cztery moduły:

1. Moduł odczytywania pliku,
2. Moduł przetwarzania sygnału,
3. Moduł detekcji zdarzeń,
4. Moduł zapisu wyników do pliku.

Autor ma nadzieję, że zaprojektowany w ten sposób system pozwoli na późniejszy rozwój i ułatwi powtórne wykorzystywanie tego narzędzia.

1.2 Motywacja

Analiza nagrań odgrywa bardzo ważną rolę we współczesnych zastosowaniach akustyki. Jednym z przykładów może być ochrona ludzi przed hałasem. Aby zbadać hałas na stanowisku pracy lub ocenić narażenie na hałas przestrzeni publicznej niejednokrotnie wykonuje się bardzo długie nagrania, trwające nawet kilkanaście godzin. Ze względu na metody analizy nieraz niezbędne jest wyselekcjonowanie z nagrania konkretnych zdarzeń akustycznych i analiza ich w izolacji, nie biorąc pod uwagę poziomu tła w pozostałym czasie.

Z drugiej strony, możemy wyobrazić sobie potrzebę wyizolowania z nagrania bardzo wielu zdarzeń akustycznych, które wydarzają się w krótkim odstępie czasu. Jako przykład może tutaj posłużyć strzelnica, gdzie chcąc policzyć liczbę strzałów w ciągu dnia możemy zrobić nagranie krótkie i na podstawie tego krótkiego wycinka czasu oszacować hałas podczas całego dnia. Innym razem możemy mieć potrzebę wykryć te zdarzenia do celów

z pozoru nie związanych z akustyką. Gdyby ktoś chciał policzyć ile razy została odbita piłka do koszykówki podczas badania jej wytrzymałości, jednym z możliwych sposobów na zliczenie liczby odbić mogłoby być policzenie zdarzeń akustycznych jakimi są odbicia piłki od ziemi. Być może ktoś ze względów medycznych chciałby sprawdzić, ile czasu w ciągu jego snu zajmuje chrapanie i w ten sposób ocenić jak duży problem ma z tą dolegliwością. Również w tym przypadku, przy pewnej kontroli hałasu środowiska mógłby zrobić to za pomocą stworzonego narzędzia. Jak widać, wyszukiwanie zdarzeń akustycznych wraz z czasem ich trwania może być niezwykle użyteczne w wielu z pozoru nie związanych z tym dziedzinach.

Biorąc pod uwagę, że przytoczone wyżej przykłady są jedynie wąskim wycinkiem zastosowań, które mogłyby przyjść do głowy komuś, kto ma wiedzę i zainteresowanie w innej dziedzinie niż autor pracy nie ulega wątpliwości, że warto posiadać takie oprogramowanie. Oczywiście wszystkie przytoczone sytuacje można analizować ręcznie, odsłuchując nagrane próbki. Analiza takich nagrań jednak pochłania dużo czasu i może być problematyczna. Z pomocą stworzonego narzędzia można proces zupełnie lub częściowo zautomatyzować co w obu przypadkach skutkuje znaczną oszczędnością czasu.

Praca powstała w połowie z chęci stworzenia użytecznego narzędzia a w połowie z chęci Autora do poznania współczesnych metod analizy cyfrowych sygnałów fonicznych. Rozwinięcie wiedzy w zakresie tworzenia oprogramowania w języku Python, pracy z systemem kontroli wersji oraz implementacja algorytmów znanych z książek do realnie działającego programu jest procesem, który autor chciał zgłębić. Przedstawienie wyników w formie zrozumiałej, czytelnej i praktycznej jest nieraz jeszcze większym wyzwaniem i nie można nabyć w tym wprawy inaczej, niż pracując z tymi wynikami i samemu przekonać się na ile są one użyteczne.

Powyższe przesłanki zadecydowały o stworzeniu narzędzia.

Rozdział 2

Wprowadzenie

W tym rozdziale zostaną wprowadzone podstawowe pojęcia, którymi autor posługiwał się podczas tworzenia pracy. Omówione zostaną teoretyczne podstawy problemu oraz ogólna struktura logiczna programu.

2.1 Zdarzenie akustyczne

Jak wspomniano w powyższym rozdziale, nie ma jednej ogólnej definicji zdarzenia akustycznego, które odnosi się do wszystkich dziedzin akustyki. Jedną z możliwych definicji podana jest w normie [PN-ISO-1996-1:2006, 2006] dotyczącej akustyki środowiskowej. Zapis normatywny mówi:

"Należy podawać czas trwania zdarzenia w odniesieniu do pewnej cechy dźwięku, jak np, liczba przekroczeń pewnego ustalonego poziomu.

Przykład: czas trwania zdarzenia można zdefiniować jako całkowity czas, w którym poziom ciśnienia akustycznego mieści się w zakresie 10 dB maksymalnego poziomu ciśnienia akustycznego podczas zdarzenia [PN-ISO-1996-1:2006, 2006]."

Biorąc pod uwagę przytoczone wcześniej możliwe zastosowania stworzonego oprogramowania powyższa definicja dobrze opisuje te sytuacje, które mają być detekowane. Ponadto, wykonanie oprogramowania, które rozpoznaje zdarzenia opisane w normie daje perspektywy na możliwości jego praktycznego zastosowania.

2.2 Przyjęta definicja zdarzenia akustycznego

W poprzedniej sekcji przytoczony został zapis normatywny, który opisuje zdarzenie akustyczne na potrzeby akustyki środowiskowej. Sugeruje on, żeby czas trwania zdarzenia akustycznego definiować jako całkowity czas, w którym poziom ciśnienia akustycznego mieści się w zakresie 10 dB maksymalnego poziomu ciśnienia akustycznego podczas zdarzenia. Nie można zapomnieć, że pomiary w akustyce środowiskowej często wykonywane są przez długi czas. Dla przykładu, pomiar hałasu na stanowisku pracy może być wykonywany przez osiem godzin bez przerwy [PN-ISO-9612:2011, 2011]. Plik z ośmiogodzinnymi pomiarami może być monofoniczny, zapisany w formacie wave o częstotliwości próbkowania czterdziestu ośmiu kilo Herców (48 kHz) oraz rozdzielczości dwudziestu czterech bitów.

Każda próbka takiego pliku zajmuje zatem dwadzieścia cztery bity w pamięci a każda sekunda zawiera czterdzieści osiem tysięcy próbek [Pohlmann, 2000]. Przy pomocy prostego wzoru możemy obliczyć jego rozmiar w pamięci komputera liczonej w bitach:

$$\text{rozdzielczość bitowa} * \text{częstotliwość próbkowania} * \text{długość pliku} = \text{rozmiar pliku}. \quad (2.1)$$

$$24 [b] * 48000 \left[\frac{1}{s}\right] * 8 * 60 * 60 [s] = 24 [b] * 48000 \left[\frac{1}{s}\right] * 28800 [s] = 3.31776 * 10^{10} [b]. \quad (2.2)$$

Po przeliczeniu tego na jednostki bardziej sprzyjające interpretacji wyniku, przybliżając że

$$1GB \approx 8 * 10^9 [b]$$

, otrzymamy:

$$\frac{3.31776 * 10^{10} [b]}{8 * 10^9 \left[\frac{b}{GB}\right]} = 4,1472[GB]. \quad (2.3)$$

Po dokonaniu takich obliczeń, można zauważyć, że przetwarzania takiego pliku nie jest zadaniem trywialnym. Jeżeli program wczytywałby cały plik do pamięci RAM, potrzebowalby on około czterech gigabajtów tej pamięci na samo obsłużenie pliku. Gdy dodamy do tego potrzebę pamięci operacyjnej dla samego programu, potrzeby systemu operacyjnego oraz innych aplikacji działających równolegle z programem zauważamy problem w postaci braku tych zasobów. Nowoczesne stacje robocze poradziłyby sobie z takim obciążeniem, jednak należy pamiętać o kilku rzeczach:

- Nie wszystkie firmy i osoby fizyczne dysponują stacjami roboczymi, posiadającymi duże zasoby pamięci operacyjnej RAM,
- plik może być dłuższy,
- plik może zostać nagrany z wyższą częstotliwością próbkowania,
- może nastąpić potrzeba współdzielenia pamięci z innymi programami bez możliwości ich wyłączenia.

Powyższe czynniki w głównej mierze skłoniły autora do tego aby swój program oprzeć o przetwarzanie sygnału fragmentarycznie. Dokładny sposób działania programu zostanie omówiony w dalszej części pracy. Co istotne w tym punkcie, wracając do definicji przytoczonej w normie [PN-ISO-1996-1:2006, 2006], zadaniem nietrywialnym jest osiągnąć jednocześnie obie te funkcjonalności:

1. Odczytywanie programu fragmentarycznie.
2. Zapamiętanie wszystkich poprzednich wartości próbek aby w razie potrzeby cofnąć się do poprzedniego fragmentu celem odnalezienia spadku poziomu o 10 dB.

Mając na uwadze powyższe przesłanki oraz możliwość późniejszego rozszerzania funkcjonalności programu, autor zdecydował się aby ograniczyć definicję zdarzenia akustycznego do definicji:

"Należy podawać czas trwania zdarzenia w odniesieniu do pewnej cechy dźwięku, jak np, liczba przekroczeń pewnego ustalonego poziomu [PN-ISO-1996-1:2006, 2006]."

Powyższa definicja jest mniej praktyczna niż jej rozszerzona wersja. Wymaga ona wiedzy o pewnych danych środowiska i zdarzenia, jak np. poziom tła akustycznego oraz przewidywany poziom ciśnienia akustycznego generowany podczas zdarzenia. Pomimo świadomości tych ograniczeń autor postanowił zastosować tę uproszczoną definicję, aby program realizował fragmentaryczne przetwarzanie danych i tym samym spełniał w całości założenia pracy jednocześnie dając lepsze możliwości na jego rozwój w przyszłości. Autor ma nadzieję, że dodanie innego algorytmu do detekcji bazującego na przetwarzaniu sygnału partiami będzie prostsze niż przyszła zmiana samego centrum oprogramowania, czyli odczytywania i zapisywania wyników.

2.3 Metoda wykrywania zakłóceń

Na podstawie informacji, o których mowa powyżej autor podjął decyzję aby wykrywać przekroczenia pewnego ustalonego poziomu ciśnienia akustycznego. Ciśnienie akustyczne to jednak określenie zbyt ubogie aby w pełni precyzyjnie opisać funkcjonalność narzędzia. Autor zdecydował się na zaczerpnięcie dodatkowych informacji z rozporządzeń normatywnych [PN-EN 61672-1:2014-03, 2015]. W związku z tym, program wykonuje następujące operacje:

1. Odnalezienie wszystkich plików wave i pliku referencyjnego w zadanym folderze,
2. odczytanie referencyjnego pliku wave o znanym poziomie dB SPL Z,
3. korekcja częstotliwościowa i uśrednianie w czasie zgodnie z zaleceniami normy [PN-EN 61672-1:2014-03, 2015],
4. przeliczenie otrzymanych próbek na wartość dB FS,
5. odczytanie fragmentu pliku wave zawierające badany sygnał
6. korekcja częstotliwościowa i uśrednianie w czasie zgodnie z zaleceniami normy [PN-EN 61672-1:2014-03, 2015],
7. przeliczenie otrzymanych próbek na wartości dB FS,
8. przeliczenie otrzymanych wartości na wartości dB SPL poprzez porównanie ich z wartością referencyjną,
9. wyszukanie przekroczeń ustalonego poziomu i zapisanie do pamięci stanu programu (ilości znalezionych zdarzeń, zdarzeniu które miało początek na końcu przetwarzanego bloku),
10. repetycja punktów 4–8 aż do końca pliku,
11. segregacja odnalezionych zdarzeń,
12. ekstrakcja zdarzeń do osobnych plików wave w podfolderze folderu, w którym znajdowały się nagrania,
13. zapisanie wyniku do pliku w formacie .json,
14. repetycja kroków 4–12 dla wszystkich znalezionych plików wave,
15. wyświetlenie komunikatu o poprawnym zakończeniu działania programu.

2.4 Istniejące rozwiązania do wykrywania zdarzeń akustycznych

Rozdział 3

Metodologia

W tej części zostanie omówiona metodologia przeprowadzonej pracy. Przedstawione zostaną narzędzia użyte do stworzenia programu, oprogramowanie, środowisko programistyczne oraz język programowania.

3.1 System kontroli wersji - Git, GitHub, GitKraken

3.1.1 System kontroli wersji

We współczesnym świecie istnieje bardzo wiele języków programowania. Każdy z nich oferuje nieco inne możliwości i funkcjonalności. Niezależnie jednak od wybranego języka, niezwykle istotnym jest system kontroli wersji.

Jednym z najpopularniejszych współcześnie systemów kontroli wersji jest Git. System kontroli wersji oferuje możliwość ciągłego śledzenia zmian w kodzie programu bez potrzeby ręcznego zapisywania wielu wersji pliku [Software Freedom Conservancy, 2018]. Problem wersjonowania oprogramowania jest znany od dawna i jest jednym z kluczowych zagadnień pracy nad programem, szczególnie kiedy pracuje się w zespole. Gdy kilka osób pracuje nad jednym fragmentem kodu, wydaje się niemożliwe aby współpracować bez systemu kontroli wersji.

Pomimo iż praca dyplomowa inżynierska jest projektem jednoosobowym, system kontroli wersji spełnia swoją rolę i w takim przypadku. Praca z kodem jest nierozłącznie związana z wielokrotnymi zmianami wcześniej wprowadzonych rozwiązań. Czasami wynika to z faktu odkrycia nowych, lepszych rozwiązań. Bywają też przypadki, kiedy w zaawansowanym stadium pracy okaże się, że już na samym początku został popełniony błąd w logice działania i trzeba coś zmienić w jednej funkcji. Często te zmiany wymagają powrotu do momentu pracy sprzed paru dni a nawet tygodni, ponieważ kolejne fragmenty kodu opierały swoje funkcjonowanie na działaniu tych wadliwych elementów. Nie jest możliwym pamiętanie wszystkich tych zmian i płynny powrót do nich poprzez przepisanie kodu. W takiej sytuacji, system kontroli wersji jest niezawodnym narzędziem, które pomaga zorganizować rozwój oprogramowania.

Dodatkowo nie można nie docenić możliwości porównywania wersji roboczej pliku z dowolną wersją tego pliku przechowywaną na serwerze. Daje to możliwość ocenienia wprowadzonych zmian, zastanowienia się czy wszystkie były konieczne oraz odpowiednie przypomnienia sobie od czego zaczynaliśmy. Wszystkie te elementy pozwalają pracować w sposób efektywny i uporządkowany.

Ostatnim — ale na pewno nie najmniej ważnym — elementem jest możliwość przechowywania całego programu na zdalnym serwerze. W dobie komputeryzacji bardzo czę-

sto zdarza się, że pracujemy na kilku różnych urządzeniach nad tym samym projektem. Korzystając z tego udogodnienia nie występuje problem przenoszenia danych pomiędzy urządzeniami. Dodatkowo w razie utraty sprzętu postępy pracy nie zostają stracone.

Powyżej omówione aspekty jednoznacznie wskazują, że pracując nad oprogramowaniem, chcąc robić to w sposób odpowiedzialny i umożliwiający przyszłą współpracę korzystanie z systemu kontroli wersji jest nieodzownym elementem pracy. Dbanie o czystość i przejrzystość wprowadzanych zmian również jest elementem, którym powinna cechować się dobrze wykonana praca.

3.1.2 Git

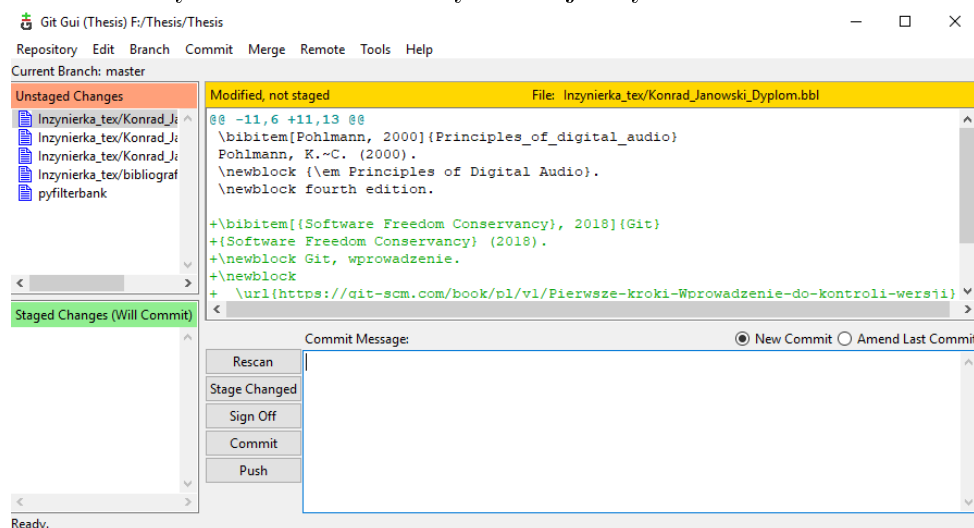
Jak zostało powiedziane, autor zdecydował się na skorzystanie z Git'a jako systemu do kontroli wersji. Jest to bardzo popularny system. Oferuje prostotę obsługi, przejrzystość, dobrą możliwość współpracy w małych grupach. Dodatkowo twórcy Git'a oferują darmowe prywatne miejsca na serwerze do przechowywania danych. Dzięki temu gestowi oraz wymienionych wyżej zaletach autor zdecydował się wybrać właśnie ten system.

3.1.3 GitKraken

Pomimo iż autor zdecydował się tworzyć oprogramowanie w ramach pracy dyplomowej inżynierskiej warto pamiętać o tym, że praca w przeznaczeniu jest skierowana do akustyków. W związku z tym poza efektywnością działania programu ważne jest też jego prostota i przejrzystość aby podczas pracy nad nim oraz przyszłej możliwej rozbudowy można skupiać się na istocie działania a nie odkrywaniu zawitych sposobów obsługi oprogramowania towarzyszącego.

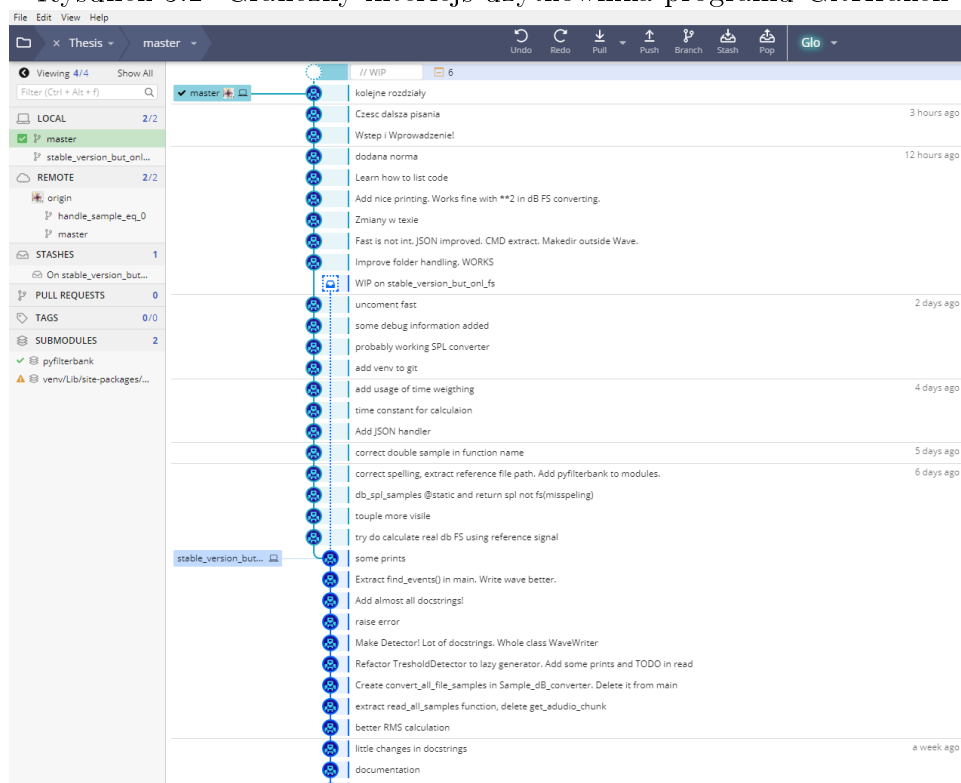
Git w wersji podstawowej jest dostarczany z bardzo ubogim graficznym interfejsem użytkownika. Przykładowe okno zostało zaprezentowane na obrazie (3.1)

Rysunek 3.1 Graficzny interfejs użytkownika GIT



Aby lepiej móc skupić się na logicznej strukturze programu autor zdecydował skorzystać z programu GitKraken, który umożliwia pracę w oparciu o system kontroli wersji Git w bardziej przejrzystej i czytelnej formie. Przykładowe drzewo kolejnych zmian zaprezentowano na obrazie (3.1.3)

Rysunek 3.2 Graficzny interfejs użytkownika programu GitKraken



W opinii autora dobranie odpowiednich narzędzi jest bardzo istotne w procesie tworzenia aplikacji. Wielu twórców oprogramowania, którzy nie są ukierunkowani na programowanie a jedynie używają go jako narzędzia do wykonania innych czynności zaniedbują wyżej wymienione aspekty dbałości o czystość kodu. Dostęp i odpowiednie użycie graficznych narzędzi, które są znacznie wygodniejsze i bardziej intuicyjne niż ich konsolowe lub ubogie odpowiedniki z pewnością może zachęcić twórców do większej dbałości o ten aspekt tworzenia programów

3.2 Język programowania - Python

Mając pomysł na działanie programu oraz niezbędne narzędzie do kontrolowania postępów prac należy wybrać język programowania, w którym zostanie napisany program. Zdecydowano, że tym językiem będzie Python w wersji 3.7.1. Jest to najnowsza stabilna wersja tego języka [Python Software Foundation, 2018d] w czasie tworzenia oprogramowania.

Wybrano Pythona z kilku powodów. Python jest obecnie jednym z najpopularniejszych języków programowania [Rassmussen College, 2017]. Rozwijanie programu i przyszła współpraca z innymi twórcami może być dzięki temu ułatwiona. Łatwiej znaleźć innych ludzi posługujących się Pythonem niż innymi, mniej popularnymi językami.

Dodatkowo Python jest prosty w składni, intuicyjny i ma nieduży próg wejścia. Nawet osoba, która nie jest doświadczona w programowaniu może stworzyć podstawowe aplikacje. Oczywiście bardziej zaawansowane struktury wymagają wiedzy, umiejętności i doświadczenia ale ta początkowa intuicyjność daje szansę zapoznania się z nim niezawodowym programistom.

Ponadto Python ma rozbudowaną bazę bibliotek i modułów, które można używać ponownie do swoich zastosowań [Python Software Foundation, 2018e]. Daje to ogromne

możliwości tworzenia programów dopasowanych do potrzeb, korzystając z gotowych modułów. Gdyby za każdym razem trzeba było zapisywać podstawowe operacje od nowa, proces kodowania byłby zdecydowanie dłuższy i mógłby zajmować więcej czasu niż osoba zajmująca się inną dziedziną może przeznaczyć na opracowanie narzędzia.

Do tego Python ma doskonale opracowaną dokumentację. Większość modułów jest opisana czytelnie i zrozumiale. Zdecydowanie ułatwia to pracę z tym językiem.

Co ważne, Python jest językiem interpretowanym a nie kompilowanym [Reitz and schulsser, 2018]. Oznacza to, że do działania potrzebuje interpretera, który działa niezależnie od systemu operacyjnego i jego bibliotek. Co prawda spowalnia to nieco jego działanie w porównaniu to języków kompilowanych takich jak C++ ale daje lepszą możliwość dzielenia się oprogramowaniem. Można skonstruować interpreter tak, by mógł być wysłany razem z kodem programu i osoba odbierająca bez problemu będzie mogła ten kod interpretować. W przypadku języków kompilowanych występują często problemy z udostępnianiem kodu źródłowego właśnie ze względu na ich silną interakcję z systemem operacyjnym. W kontekście przyszłego rozwoju programu dla akustyków jest to ważny aspekt wyboru języka.

Z podanych powodów, Python został wybrany do napisania pracy.

Z wyborem Pythona wiąże się jeszcze jeden wybór, jego wersji. Obecnie najnowszą wersją jest Python 3.7.1 ale równocześnie wspierany jest Python w wersji 2.7.x [Python Software Foundation, 2018d]. Niestety, zmiana z Pythona wersji drugiej na wersję trzecią wiąże się ze znaczącymi zmianami w funkcjonalności języka i wersje te nie są ze sobą kompatybilne. Wiele modułów aktualnie dostępnych do użytku dalej funkcjonuje w wersji drugiej języka. Można by więc pokusić się o napisanie programu w starszej wersji. Python wersji drugiej przestaje jednak być wspierany wraz z końcem 2020 roku [Python Software Foundation, 2016]. Oznacza to, że najprawdopodobniej ze względów bezpieczeństwa i braku wsparcia większość nowych modułów będzie pisana w Pythonie w wersji trzeciej a dodatkowo w razie nieprawidłowości w działaniu wersji drugiej, nie będzie można liczyć na żadną pomoc.

3.3 Środowisko programistyczne - Pycharm

Pycharm jest środowiskiem dedykowanym do obsługi Pythona. Mając niewielkie doświadczenie z tym środowiskiem autor wybrał je aby je poznać i sprawdzić jego działanie. Każde środowisko oferuje odmienny zestaw funkcjonalności i ułatwień użytkownika.

3.4 Plik z informacjami o zdarzeniach: JSON

Do reprezentacji wyników został wybrany format JSON opisany przez standard Ecma [Ecma International, 2017]. JSON jest popularnym sposobem wyświetlania i przekazywania danych. Po raz kolejny o jego wyborze przesądziło jego rosnąca popularność i prostota użycia [Strassner,]. Dodatkowo, jest on łatwo odczytywany zarówno przez oprogramowanie jak i przez człowieka co dodatkowo poprawia prostotę użytkownika. Do przekazania wyników i ich obejrzenia nie potrzebujemy żadnego specjalnego oprogramowania, wystarczą standardowe aplikacje do odczytu plików tekstowych dostępne na większości systemów operacyjnych.

Rozdział 4

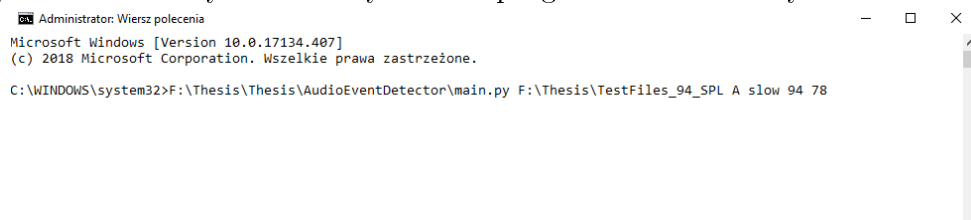
Struktura oraz działanie programu

4.1 Sposób działania programu - aplikacja konsolowa

Decydując się na program, należy wybrać w jaki sposób będzie on uruchamiany. Może być jedynie biblioteką, czyli zestawem funkcjonalności, które mogą być wykorzystywane przez inny program. Jest również możliwość skonstruowania rozbudowanego graficznego interfejsu użytkownika. Na ten moment, autor zdecydował aby oprogramowanie działało jako program konsolowy.

Oznacza to, że wywołuje się go z konsoli systemu z odpowiednimi parametrami. Przykładowe wywołanie widzimy na obrazie(4.1). Parametry wywołania i struktura działania programu zostaną omówione w dalszej części pracy.

Rysunek 4.1 Przykładowe wywołanie programu w konsoli systemu Windows



Tego typu konstrukcja daje możliwość używania programu jako samodzielnego narzędzia. Jednocześnie, ze względu na ograniczony czas pracy nad programem stworzenie graficznego interfejsu mogłoby negatywnie wpłynąć na zadbanie o poprawną funkcjonalność. W ramach rozwoju programu Autor nie wyklucza dodania graficznego interfejsu użytkownika do programu.

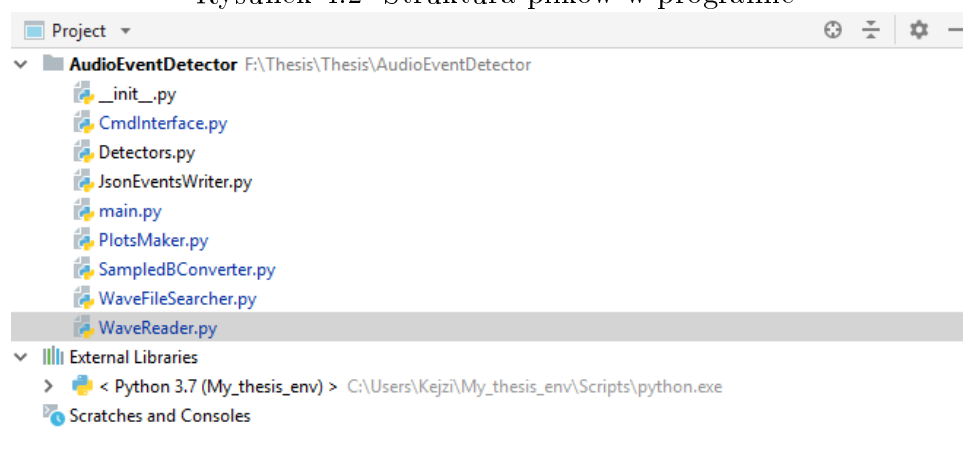
4.2 Struktura programu

Struktura plików w programie przedstawiona została na rysunku (4.2)

Katalog "AudioEventDetector" zawiera wszystkie pliki źródłowe używane w programie. Na początku widzimy plik "__init__.py", który informuje interpreter Pythona, że katalog zawiera pythonowe pakiety [Drakos and Moore, 2003].

Plik "CmdInterface.py" odpowiada za komunikację z wierszem poleceń i odczytywaniem odpowiednich parametrów podanych przez użytkownika. "Detectors.py" to dwie klasy odpowiedzialne za detekowanie i uporządkowanie zdarzeń akustycznych. "JsonEventWriter.py" zawiera funkcjonalność odpowiedzialną za zapisywanie wyniku to pliku

Rysunek 4.2 Struktura plików w programie



.json. "main.py" to główna część programu, która wywołuje pozostałe moduły w celu skonstruowania funkcjonalnej całości. "PlotsMaker.py" zawiera jedną klasę odpowiedzialną za zrobienie prostego wykresu z danych. Nie jest ona używana w programie ale podczas rozwoju często przydaje się do wizualnej obserwacji danych. "SampleDbConverter.py" zawiera klasy odpowiedzialne za konwersję pliku z danych reprezentujących dynamiczny przebieg ciśnienia akustycznego na skorygowany częstotliwościowo uśredniony w czasie poziom ciśnienia akustycznego. "WaveFileSearcher.py" jest odpowiedzialny za odnalezienie plików wave do analizy oraz pliku referencyjnego. "WaveRader.py" odpowiada za obsługę plików wave. Odczytuje fragmentami pliki na początku działania programu aby była możliwa ich analiza, na końcu programu zapisuje wybrane fragmenty do podfolderu.

Uważny czytelnik zauważy, że wszystkie nazwy modułów zachowane są w jednej konwencji, nazwanej "CamelCase", co w tłumaczeniu na polski można określić jako "Styl Wielbłąda". Autor zdecydował się na ten sposób notacji uwzględniając rekomendacje twórców języka [van Rossum et al., 2001]. Używanie spójnego nazewnictwa jest o tyle istotne, że pozwala się w prosty sposób zorientować w strukturze programu. Jeżeli twórcy oprogramowania przestrzegają ustalonych, umownych zasad kod staje się z czasem coraz bardziej przejrzysty i wymiana informacji jest dużo prostsza.

Rozdział 5

Szczegółowe omówienie funkcji w programie

W tym rozdziale na przykładzie kodu szczegółowo zostaną omówione główne funkcje programu. Autor postara się, żeby funkcje były omówione w kolejności jak najbardziej zbliżonej do logiki programu. Należy mieć jednak na uwadze, że niektóre funkcje wykorzystywane są kilkakrotnie co wyklucza zupełnie liniowe rozumienie kodu.

Warto w tym miejscu nadmienić, że istotnym elementem programu poza jego modularnością jako całości jest też zachowanie czystości kodu wewnątrz klas. Zasada pojedynczej odpowiedzialności, znaczących nazw jak i inne dobre zasady pisania estetycznego kodu zostały zaczerpnięte z literatury [Martin, 2014]. Autor wierzy, że każdy kod powinien być możliwie łatwy do przeczytania przez człowieka, ponieważ to on jest głównym podmiotem pracującym nad nim. Być może zastosowanie tych zasad ułatwi a może nawet i uprzyjemni czytanie i zrozumienie stworzonego kodu.

Należy mieć na uwadze, że w umieszczonych fragmentach kodu może się zdarzyć ominięcie niektórych wcięć. Spowodowane jest to szerokością kodu, który do odczytu w środowiskach powinien mieć 150 [van Rossum et al., 2001]. Taka szerokość jednak nie pozwala na komfortowe umieszczenie listingu na stronie formatu A4. Dołożono wszelkich starań, aby wszystkie wcięcia istotne dla zrozumienia logiki kodu zostały zachowane

5.1 Odnajdywanie plików wave

```

1 class WaveFileSearcher:
2     def find_wave_files_paths(self, ):
3         """
4         Find wave files under path given in cmd. Path must be a directory containing wav files
5         and REFERENCE.wav file.
6         Returns
7         -----
8         wave_files_and_reference_paths: [str]
9         tuple of two list. First is wave files paths to analyze,
10        second is reference file path.
11        """
12        path = CmdInterface.get_path_from_cmd()
13        wave_files_paths = []
14        if os.path.isfile(path):
15            raise NotADirectoryError('path is a file not a directory')
16        elif os.path.isdir(path):
17            path_content = os.listdir(path)
18            wave_files_paths = ["{}/{}/".format(path, file_path) for file_path
19                                in path_content if (".wav" in file_path
20                                                    or ".WAV" in file_path) and ("REFERENCE" not in file_path)]
21            reference_file_path = ["{}/{}/".format(path, file_path) for file_path in path_content
22                                   if "REFERENCE" in file_path]
23        else:
24            logging.error('{}/ is no valid path'.format(path))
25            raise FileNotFoundError('{}/ is not a valid path'.format(path))
26
27        assert wave_files_paths, 'there is no any wave file under given path'
28        assert reference_file_path, 'there is no REFERENCE.wav file'
29        wave_files_names = [ntpath.basename(path) for path in wave_files_paths]
30
31        print('I found {} files. Files names are: {}'.format(len(wave_files_names), wave_files_names))
32        print('Found reference file under path: {}'.format(ntpath.abspath(reference_file_path[0])))
33        print('')
34
35        wave_files_and_reference_paths = (wave_files_paths, reference_file_path)
36        return wave_files_and_reference_paths

```

Listing 5.1 fragment kodu źródłowego pliku WaveFileSearcher.py

Pierwsze co zostało zapewnione w tej metodzie to odporność na podanie niewłaściwej ścieżki do folderu. Program działa jedynie gdy zostanie podana ścieżka do folderu, który musi zawierać pliki wave oraz plik REFERENCE.wav. Jeżeli któregoś z tych elementów zabraknie, dostaniemy odpowiednie informacje o błędach odpowiednio w liniach 13, 24 i 25. Dodatkowo widać w liniach 28–30 dodatkowe instrukcje print. Zostały wprowadzone w tym jak i w innych modułach w celu informowania użytkownika o tym, w którym miejscu programu aktualnie się znajduje podczas pracy. Są to informacje pomocnicze, nie ostrzegające o błędach a jedynie informujące o przebiegu. Pomagają jednak zorientować się, gdy przypadkiem ustawimy nie takie opcje jak byśmy chcieli. Główną funkcjonalność programu widzimy w liniach 14–19 w którym to tworzone są dwa obiekty, "wave_files_paths" oraz "reference_file_path". Są to listy, które zawierają odpowiednio ścieżki to plików wave, które zostaną poddane analizie oraz ścieżkę do pliku referencyjnego. Na koniec, w lini 33. zostaje zwrócony obiekt zawierający obie te wartości.

Dodatkowo można zauważyć tutaj kolejną konwencję nazewnictwa, klas, funkcji oraz zmiennych. Autor podczas całego programu stosował nazewnictwo zgodne ze standardem [van Rossum et al., 2001]. Co ułatwia odnalezienie się w programie.

5.2 Odczytywanie pliku Wave

```

1 def read_audio_data_chunk(self, seconds_to_read=30):
2     """ Read audio data in chunks.
3     Parameters
4     -----
5         seconds_to_read: int
6             define how many seconds will be read from file.
7     Returns
8     -----
9         samples: [float]
10             list of value of audio samples. """
11     chunk_size = seconds_to_read * self.frame_rate
12     total_length = round(self.audio_file.getnframes() / self.audio_file.getframerate(), 2)
13     print('Read file {}'.format(ntpath.abspath(self.file_path)))
14     print('{} frame rate is {}, chunk size is {}'.format(self.frame_rate, chunk_size))
15     print('total length is {} s'.format(total_length))
16
17     while True:
18         start = self.audio_file.tell()
19
20         print('Read samples from {} to {}'.format(start, start + chunk_size))
21
22         samples = self.audio_file.readframes(chunk_size)
23
24         print('Samples from {} to {} has been read'.format(start, self.audio_file.tell()))
25
26         if not samples:
27             print('End reading {}. Read {} frames '.format(ntpath.basename(self.file_path),
28                 self.audio_file.tell()))
29             print('')
30             self.audio_file.close()
31             return
32         print('')
33         samples = self.decode_audio_chunk(samples)
34         yield samples

```

Listing 5.2 fragment kodu źródłowego pliku WaveFileReader.py

Na listingu 5.2 widzimy moduł odpowiedzialny za odczytywanie danych z pliku wave. Jak zostało wspomniane wcześniej, funkcja jest przystosowana to działania wielokrotnie i odczytywania pliku w kawałkach. Aby to umożliwić, funkcja została zaprojektowana aby była używana jako generator [Python Software Foundation, 2018c]. Pozwala to na cykliczne zwracanie próbek, co dzieje się w linii 33. bez utraty informacji z poprzedniego wywołania. Istotnym elementem tej funkcji jest moment końca iteracji. Kiedy plik zostanie przeczytany do końca, zostaje wywołane słowo kluczowe "return"(linia 30) zamiast "yield"(linia 33). Jest to zgodne z rekomendacją do Pythona wersji trzeciej, [Ewing, 2009] która zmieniła działanie generatorów. W poprzednich wersjach przerwanie iteracji wykonywało się zazwyczaj poprzez instrukcję "raise(StopIteration)".

Ilość czytanych danych jest z góry zdefiniowana w linii 1. Jest ona przeliczana w linii 11 na rozmiar czytanego fragmentu w bitach.

5.3 Konwersja pliku

Konwersja pliku odbywa się wewnątrz piku SampledBConverter.

wewnątrz znajdują się dwie klasy:

```

1
2 class SamplesDbFsConverter:
3     """Convert samples from given wave file to frequency and time weighted signal
4     according to IEC 61672-1:2013"""
5
6     def __init__(self, file_path):
7         self.wave_reader_object = WaveReader(file_path)
8         self.audio_samples_generator = self.wave_reader_object.read_audio_data_chunk()
9         self.frequency_weighting = CmdInterface.get_frequency_weighting_from_cmd()
10        self.time_weighting = CmdInterface.get_time_weighting_from_cmd()
11        self.reference_db_spl_value = CmdInterface.get_reference_db_spl()

```

Listing 5.3 fragment kodu źródłowego pliku SampledBConverter.py,
klasa SamplesDbFsConverter

```

1
2 class SamplesDbSPLConverter(SamplesDbFsConverter):
3     """Subclasses of SamplesDbFsConverter which add conversion to dB SPL"""
4     def __init__(self, file_path, reference_db_fs_value):
5         super().__init__(file_path)
6         self.reference_db_fs_value = reference_db_fs_value

```

Listing 5.4 fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDbSPLConverter

Zaprezentowano listing konstruktorów klas SamplesDbFsConverter (5.3) oraz SamplesDbSPLConverter (5.4). Jak widać, klasa SamplesDbSPLConverter dziedziczy po klasie SamplesDbFsConverter. Pokazuje to linia 2 listingu 5.4 gdzie w linii 2. widzimy odziedziczenie wszystkich metod i w linii 5. wywołanie konstruktora klasy nadrzędnej. Rozwiązanie to zastosowano, ponieważ klasa SamplesDbFsConverter zostaje użyta to konwersji pliku referencyjnego. Gdyby od razu zaimplementować pełną funkcjonalności klasy SamplesDbSPLConverter, z oczywistej przyczyny referencyjna wartość "reference_db_fs_value" nie mogłaby być jeszcze znana.

Przyjrzyjmy się zatem bliżej metodom klasy SamplesDbFsConverter.

```

1
2 def _convert_samples_to_db_fs(self, energy_samples):
3     """Convert samples in energy unit (preferably p^2) to dB FS according to AES17-2015.
4     FS value is calculated from sample_width of read object.
5     Parameters
6         energy_samples: [float]
7             list of samples in energy unit (e.x p^2).
8     Returns
9         list(samples_db_fs): [float]
10            list of samples in dB FS format.
11    """
12    max_value = 2*(self.wave_reader_object.sample_width*8-1)
13    samples_db_fs = [10 * self._log10_dealing_with_0((sample/max_value**2)*np.sqrt(2))
14                    for sample in energy_samples]
15    print('{} energy unit samples has been converted
16          to {} samples dB FS'.format(len(energy_samples), len(samples_db_fs)))
17    print('Full scale level is {}'.format(max_value))
18    print('')
19    return list(samples_db_fs)
20
21

```

Listing 5.5 fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDbFsConverter, metoda _convert_samples_to_db_fs

Metoda _convert_samples_to_db_fs pokazana na listingu 5.5 odpowiada za konwersję sampli do wartości dB FS. W linii 14. na podstawie rozdzielczości bitowej zostaje obliczona maksymalna wartość cyfrowa, która mogła znaleźć się w odczytywanym pliku wave [Zieliński, 2005]. Obliczenie wartości dB FS zostaje wykonane zgodnie z normą AES [Audio Engineering Society, Inc, 2018]. Ciekawym momentem jest linia 15, która wykorzystuje listę składaną do konstrukcji listy zawierającej przekonwertowane próbki. Listy składane pojawiały się już wcześniej w programie. Do tej pory jednak działanie ich wynikało głównie z dbania o czytelność kodu, listy składane bowiem często są bardziej zwarte i czytelne niż ich klasyczne odpowiedniki [Python Software Foundation, 2018b]. Tutaj jednak szczególnie istotne jest to, o czym wspomniano w rozdziale 3.2. Język Python jest językiem interpretowanym, jednak część jego funkcji zostaje na poziomie interpretacji sprowadzona do kodu w języku C i skompilowane, przez co działają z szybkością równą językom kompilowanym [Lutz, 2011]. Jedną z takich struktur jest lista składana. W widocznym przykładzie jest to o tyle istotne, że wykonywanych jest wiele obliczeń i czas ich wykonania jest istotnym elementem całego czasu działania programu.

```

1  def _filter_db_samples_with_time_constant(self, samples):
2      """Take dynamic pressure samples and integrate it with time
3      constant defined in IEC-61672-2013.
4      Interact which command line do take time constant to use.
5      Allowed constants are "slow" or "fast".
6
7      Parameters
8      -----
9          samples: [float]
10             list of samples with dynamic pressure level.
11
12      Returns
13      -----
14          time_weighted_samples: [float]
15             list of samples weighted which defined time constant.
16
17      """
18      if self.time_weighting == 'slow':
19          time_weighted_samples = standards.iec_61672_1_2013.slow(np.array(samples),
20                                                                self.wave_reader_object.frame_rate)
21          print("Slow constant is applied")
22      elif self.time_weighting == 'fast':
23          time_weighted_samples = standards.iec_61672_1_2013.fast(np.array(samples),
24                                                                self.wave_reader_object.frame_rate)
25          print("Fast constant is applied")
26      else:
27          raise ValueError('time weighting must be "slow" or "fast"
28                          , not {}'.format(self.time_weighting))
29
30      print('{} samples has been converted to {} samples with {} time constant'.format(len(samples),
31                                                                                          len(time_weighted_samples),
32                                                                                          self.time_weighting))
33      print('')
34      time_weighted_samples = list(time_weighted_samples)
35      return time_weighted_samples

```

Listing 5.6 fragment kodu źródłowego pliku SampledBConverter.py,
klasa SamplesDbFSConverter, metoda _filter_db_samples_with_time_constant

Na listingu 5.6 widzimy funkcję odpowiedzialną za uśrednienie sygnału w czasie zgodnie z normą [IEC-61672-2013, 2013]. Wartość stałej czasowej jest pobierana z atrybutów klasy. W tej funkcji zostały wykorzystane metody z biblioteki `acoustics.standards` [Rietdijk, 2013].

```

1  def _filter_samples_with_weighting_filter(self, samples):
2      """Filter samples with weighting filter. Use one of the weighting defined in IEC-61672.
3      Weighting is defined in class variable.
4      Parameters
5      -----
6          samples: [float]
7             list of samples representing dynamic pressure level.
8      Returns
9      -----
10         weighted_samples: [float]
11            list of samples representing weighted samples of dynamic pressure level.
12
13      """
14
15      weighted_samples = splweighting.weight_signal(samples,
16                                                    self.wave_reader_object.frame_rate,
17                                                    self.frequency_weighting)
18      return weighted_samples

```

Listing 5.7 fragment kodu źródłowego pliku SampledBConverter.py,
klasa SamplesDbFSConverter, metoda _filter_samples_with_weighting_filter

Listing 5.7 przedstawia metodę odpowiedzialną za filtrowanie próbek zgodnie z krzywą częstotliwościową A, B lub C zdefiniowane w normie [IEC-61672-2013, 2013]. Wykorzystano tutaj funkcję `weight_signal()` z biblioteki `splweighting` [Gündert, 2014].

```

1  def convert_samples(self,):
2      """
3      Make full conversion from dynamic representation to frequency and time
4      weighted samples according to IEC-61672.
5      Returns
6      -----
7      db_fs_samples: [float]
8          frequency and time weighted full scale level.
9      """
10     while True:
11         try:
12             samples = next(self.audio_samples_generator)
13         except StopIteration:
14             return
15         frequency_weighted_samples = self._filter_samples_with_weighting_filter(samples)
16         time_weighted_samples = self._filter_db_samples_with_time_constant(
17             frequency_weighted_samples ** 2)
18         db_fs_samples = self._convert_samples_to_db_fs(time_weighted_samples)
19         yield db_fs_samples

```

Listing 5.8 fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDbSPLConverter, metoda convert_samples

Listing 5.8 pokazuje funkcję, która jest główną funkcjonalnością klasy. Wykorzystuje ona poprzednio omówione metody aby wykonać pełną konwersję pakietu danych. Widzimy więc w liniach 11–14 obsługę generatora próbek. Linie 15–17 wywołują metody opisane poprzednio, aby w linii 18. zwrócić pakiet próbek po korekcji czasowej i częstotliwościowej. Wszystko opisane jest również jako generator, którego działanie kończy się gdy skończą się próbki, w liniach 13–14. Nie jest tutaj wykonywana konwersja próbek na wartości dB SPL. Ta funkcjonalność zostaje rozszerzona w klasie SamplesDbSPLConverter.

```

1  def convert_samples(self,):
2      """
3      Make full conversion to dB SPL from dynamic representation to
4      frequency and time weighted samples according
5      to IEC-61672.
6      Returns
7      -----
8      db_fs_samples: [float]
9          frequency and time weighted full scale level.
10     """
11     while True:
12         try:
13             samples = next(self.audio_samples_generator)
14         except StopIteration:
15             return
16         frequency_weighted_samples = self._filter_samples_with_weighting_filter(samples)
17         time_weighted_samples = self._filter_db_samples_with_time_constant(
18             frequency_weighted_samples ** 2)
19         db_fs_samples = self._convert_samples_to_db_fs(time_weighted_samples)
20         db_spl_samples = self._convert_samples_to_db_spl(list(db_fs_samples))
21         yield db_spl_samples

```

Listing 5.9 fragment kodu źródłowego pliku SampleSPLConverter.py, klasa SamplesDbSPLConverter, metoda convert_samples

Na listingu 5.9 widzimy nadpisanie funkcji z listingu 5.8 rozszerzając ją o linię 18, odpowiedzialną za konwersję na dB SPL. Aby prześledzić jej działanie należy spojrzeć na listing 5.10.


```

1
2 def convert_samples_to_db_spl(self, db_fs_samples):
3     """Convert samples in dB FS to dB SPL using reference value given in command line
4     Parameters
5     -----
6     db_fs_samples: [float]
7     Returns
8     -----
9     db_spl_samples [float]
10    """
11    reference_db_spl_value = CmdInterface.get_reference_db_spl()
12    db_spl_samples = [reference_db_spl_value + (sample - self.reference_db_fs_value)
13                      for sample in db_fs_samples]
14    print('{0} dB FS {1} {2} samples has been converted to
15    {3} dB SPL {1} {2} samples'.format(len(db_fs_samples),
16                                       self.frequency_weighting,
17                                       self.time_weighting,
18                                       len(db_spl_samples)))
19    print('Reference value was {0} dB FS {1} {2} '
20          'with was equivalent to {3} dB SPL {1} {2}'.format(round(self.reference_db_fs_value, 2),
21                                                                self.frequency_weighting,
22                                                                self.time_weighting,
23                                                                self.reference_db_spl_value, ))
24    print('')
25
26    return db_spl_samples

```

Listing 5.10 fragment kodu źródłowego pliku SampleSPLConverter.py, klasa SamplesDbSPLConverter, metoda convert_samples_to_db_spl

Za konwersję odpowiedzialna jest linia 11. Tam następuje porównanie obliczonej wartości dB FS z wartością referencyjną i odpowiednia korekcja wskazania, zależna od zadeklarowanej wartości odniesienia w dB SPL oraz obliczonej wcześniej wartości odniesienia w dB FS. W ten sposób program konwertuje próbki z reprezentacji dynamicznego ciśnienia akustycznego na skorygowany częstotliwościowo, uśredniony w czasie poziom ciśnienia akustycznego. W następnej części przyjrzymy się części odpowiedzialną za odnajdywanie i organizowanie zdarzeń.

5.4 Detekcja i organizacja zdarzeń

Za detekcję i organizowanie zdarzeń odpowiedzialny jest plik "Detectors.py". Zawiera on dwie klasy, "count_occurrence" oraz "organise_events". Razem pozwalają na zdetekowanie i przygotowanie w formie możliwej do dalszej analizy zdarzeń akustycznych. Moduł ten na razie zawiera tylko jedną klasę detekującą. W dalszym rozwoju możliwe jest dodanie kolejnych. Wystarczy aby każda z klas zwracała wyniki w odpowiedniej formie a wtedy dodając tylko jej wywołanie do struktury programu "main.py" będzie można dodać dodatkową detekcję zdarzeń.

```

1  def count_occurrence(threshold, _last_previous_index=0, _found=0):
2  """Detect starts and ends acoustic events. Events are all samples above threshold.
3  Parameters
4  -----
5      threshold: float
6          Variable which defined starts and ends events. Expected value depends on yielded data.
7      _last_previous_index: int
8          Internal variable use for keeping memory of previous data.
9      _found: int
10         Internal variable use for keeping memory of founded events.
11 Returns
12 -----
13     events: [(float, float)]
14         List of tuples of two parameters. First defined detected value,
15         second number of sampel in whole file
16         If use properly it keep memory of previous data.
17 """
18 while True:
19     data = yield
20     first_index_of_chunk = _last_previous_index
21     _last_previous_index = len(data) + first_index_of_chunk
22     events = []
23     for value in data:
24         if _found % 2 == 0 and value >= threshold:
25             events += [(value, data.index(value)+first_index_of_chunk)]
26             _found += 1
27         if _found % 2 != 0 and value <= threshold:
28             events += [(value, data.index(value)+first_index_of_chunk)]
29             _found += 1
30     yield events

```

Listing 5.11 fragment kodu źródłowego pliku Detectors.py, klasa ThresholdCrossDetector, metoda count_occurrence

Listing 5.11 przedstawia funkcję znajdującą zdarzenia. Ponieważ cały program przetwarza plik we fragmentach, funkcja musi posiadać pamięć poprzednich zdarzeń i zachować ciągłość zliczania odebranych próbek. Zapewniają to inicjalizacja wartości na liście inicjalizacyjnej w linii 1. oraz następna ich aktualizacja z każdym przebiegiem funkcji. Lista inicjalizacyjna została użyta aby z każdym wywołaniem generatora nie inicjalizować na nowo wartości, które są jedynie aktualizowane w liniach 19–20. Funkcja ta działa jako generator, pobierając dane w linii 18. oraz zwracając je w linii 29. przy pomocy słowa kluczowego yield. Zliczanie zdarzeń odbywa się w pętli w liniach 22–28.

```

1  def organise_events(events, time_constant):
2  """Search in events list and prepare easy to use list of tuples.
3  Parameters
4  -----
5      time_constant: str
6          Time constant applied to a signal. Can be slow or fast.
7      events: [(float, float)]
8          List of two element tuples. Must contain list of all events
9          to organise where first value of every
10         tuple is value of sample, second is number of it position in file. Must contain every starts
11         and endings of events(except first and last).
12 Returns
13 -----
14     events_starts_ends_lengths: [(float, float, float)]
15         List of three element tuple which contains starts, endings and lengths of all founded events.
16 """
17 time_constant_ms = {'slow': 1,
18                     'fast': 0.125}
19 events_starts = [time*time_constant_ms[time_constant] for _, time in events[::2]]
20 events_ends = [time*time_constant_ms[time_constant] for _, time in events[1::2]]
21 events_length = []
22 events_starts_ends_lengths = []
23 for event_number in range(len(events_starts)):
24     try:
25         events_length.append(events_ends[event_number] - events_starts[event_number])
26     except IndexError:
27         print('there is an event which has only start')
28         break
29     events_starts_ends_lengths.append((events_starts[event_number],
30                                       events_ends[event_number],
31                                       events_length[event_number]))
32
33 if events_starts_ends_lengths:
34     print('Events found are: {}'.format(events_starts_ends_lengths))
35     print('')
36 return events_starts_ends_lengths

```

Listing 5.12 fragment kodu źródłowego pliku Detectors.py, klasa EventsOrganiser, metoda organise_events

Kod pokazany na listingu 5.12 odpowiada za zestawienie zdarzeń w formie umożliwiającej ich dalszą obróbkę. W liniach 18–20 zapisuje istotne cechy zdarzenia, biorąc pod

uwagę stałą czasową z którą zostało dokonane przetwarzanie próbek. Linie 25–27 zabezpieczają działanie programu w przypadku gdy na nagraniu pojawi się zdarzenie, które nie kończy się w czasie nagrania. Ponieważ funkcja przyjmuje jako parametr wejściowy listę znalezionych zdarzeń nie musi być przygotowana do operowania na dużych ilościach danych. Przetwarzanie odbywa się więc jednorazowo dla wszystkich próbek.

5.5 Zapis pliku .json oraz plików wave

Po przejściu poprzednich etapów, otrzymujemy gotowe dane ze zdarzeniami akustycznymi. Jedyne co pozostaje do zrobienia to zapisać je do pliku .json oraz wyekstrahować je z pliku w którym zostały znalezione.

5.5.1 JsonEventsWriter

```

1 class JsonEventsWriter:
2     def __init__(self, organised_events, file_name, destination_directory):
3         self.organised_events = organised_events
4         self.file_name = file_name
5         self.destination_directory = destination_directory
6
7     def create_events_in_json(self, ):
8         events_for_json = {'All_Events': len(self.organised_events)}
9         count = 1
10        for event in self.organised_events:
11            start, end, length = event
12            events_for_json['Event_{}'.format(count)] = {'start': start,
13                                                         'end': end,
14                                                         'length': length}
15            count += 1
16        json_events = json.dumps(events_for_json)
17        return json_events
18
19    def save_json_to_file(self, ):
20        json_events = self.create_events_in_json()
21        json_file_name = '{}_events.json'.format(self.file_name.replace('.wav',
22        ''').replace('.WAV', '''))
23        json_file_path = '{}/{}/{}'.format(self.destination_directory,
24                                           json_file_name)
25        with open(json_file_path, 'w') as json_file:
26            json_file.write(json_events)
27        print('JSON file with events has been saved to: {}'.format(json_file_path))
28        print('')

```

Listing 5.13 fragment kodu źródłowego pliku JsonEventWriter.py, klasa JsonEventsWriter

Klasa przedstawiona na listingu 5.13 posiada dwie metody, `create_events_in_json` i `save_json_to_file`. Pierwsza z nich odpowiada za przygotowanie obiektu JSON w Pythonie z listy zawierającej zdarzenia akustyczne. Następnie korzystając z tak przygotowanego pliku metoda `save_json_to_file` zapisze je w pliku z rozszerzeniem .json, w podkatalogu katalogu w którym znajdowały się pliki audio. Przykładowy plik zapisany przy pomocy tej klasy widzimy na listingu 5.14.

```

1 {"All_Events": 2, "Event_1": {"start": 11, "end": 20, "length": 9},
2  "Event_2": {"start": 30, "end": 43, "length": 13}}

```

Listing 5.14 Przykładowa zawartość pliku .json

5.5.2 Ekstrakcja z plików wav

Ostatnią czynnością jest ekstrakcja plików ze zdarzeniami akustycznymi z analizowanych plików. Zajmuje się tym klasa `WaveWriter` umieszczona w pliku `WaveReader.py`. Posiada ona dwie metody. Pierwszą z nich widzimy na listingu 5.15.

```

1
2  def read_defined_frames(self, file_path, event):
3      """Read events from audio file
4      Params
5      -----
6          file_path: str
7              path to audio file which will be read
8          event: (float, float, float)
9              tuple containing start, end and length of event
10
11      Returns
12      -----
13          frames_and_params: (b, ())
14              tuple containing frames which event and params of file.
15
16      start, end, length = event
17      audio_file = wave.open(file_path, 'rb')
18      audio_file.rewind()
19      frame_rate = audio_file.getframerate()
20      start_position = audio_file.tell() + int(start * frame_rate)
21      audio_file.setpos(start_position)
22      frames_to_read = int(length * frame_rate)
23
24      print('Read frames from {} to {}'.format(start_position, start_position + frames_to_read))
25
26      frames = audio_file.readframes(frames_to_read)
27
28      print('Samples from {} to {} has been read.'.format(start_position, audio_file.tell()))
29      print('')
30
31      params = audio_file.getparams()
32      frames_and_params = (frames, params)
33      return frames_and_params

```

Listing 5.15 fragment kodu źródłowego pliku WaveReader.py,
klasa WaveWriter, metoda read_defined_frames

Odczytuje ona tylko te bity pliku wave, które zawierają znalezione zdarzenie akustyczne. Linie 15–21 odpowiadają za przygotowanie wskaźników na obsługiwanym pliku aby odczytać odpowiednią ilość bitów z odpowiedniego miejsca. Linia 25 odczytuje zadane wcześniej bity. Zwracana jest krotka zawierająca bity oraz parametry które powinny zostać nagrane do pliku wave. Metoda ta jest wywoływana przez drugą metodę, widoczną na listingu 5.16.

```

1  def write_defined_frames(self, file_dir_path, frames_and_params, count):
2      """Write frames to file under defined path.
3      Params
4      -----
5          file_dir_path: str
6              path to dir where file should be save.
7          frames_and_params: (b, ())
8              frames to write and params of audio file
9          count: int
10             value to different each event"""
11      frames, params = frames_and_params
12      file_name = '{} / event_{}.wav'.format(file_dir_path, count)
13      audio_file = wave.open(file_name, 'wb')
14      audio_file.setparams(params)
15      audio_file.writeframes(frames)
16      print('Wrote file {}'.format(file_name))
17      audio_file.close()

```

Listing 5.16 fragment kodu źródłowego pliku WaveReader.py,
klasa WaveWriter, metoda write_defined_frames

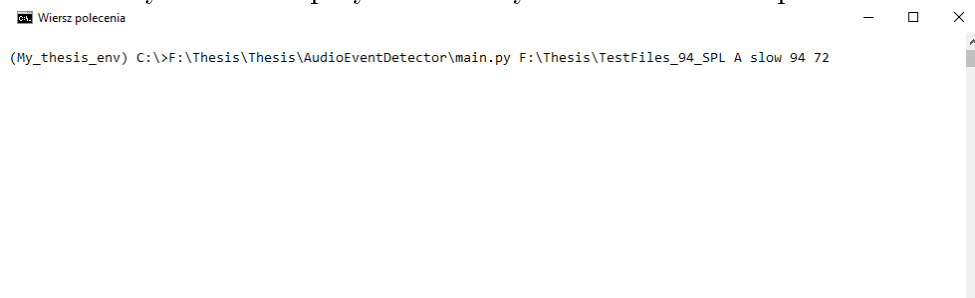
Funkcja ta ma za zadanie zapisać odczytane wcześniej bity do pliku o nazwie stworzonej w linii 11. używając parametrów zgodnych z plikiem wejściowym. W ten sposób zamyka ona proces detekcji i ekstrakcji zdarzenia z zadanego pliku.

Rozdział 6

Sposób użycia

Ta część dotyczyć będzie sposobu użycia programu z punktu widzenia użytkownika. Jak wspomniano w rozdziale 4.1 program działa jako aplikacja konsolowa. Oznacza to, że wywołanie następuje poprzez wiersz poleceń systemu operacyjnego. Rozpatrzmy dokładniej wywołanie programu pokazane na rysunku 6.1.

Rysunek 6.1 przykładowe wywołanie z wiersza poleceń
















Pierwsze co widzimy to napis `(My_thesis_env)` przed samym wywołaniem. Oznacza to, że wiersz poleceń działa w wirtualnym środowisku Python [Python Software Foundation, 2018a]. Nie jest to konieczne ale niezbędny jest interpreter Pythona w wersji 3.7.1. Następne jest 6 argumentów podanych do linii komend. Znaczenie każdego z nich to:

1. To ścieżka prowadząca do pliku main.py, który uruchamia cały program,
2. To ścieżka do folderu, w którym znajdują się pliki wave oraz plik referencyjny,
3. Definiuje krzywą korekcji częstotliwościowej, która zostanie użyta do filtracji sygnału. Możliwe opcje to "A", "B" lub "C".
4. Definiuje stałą czasową, według której zostaną uśrednione próbki. Dopuszczalne wartości to "slow" oraz "fast".
5. To poziom w dB SPL pliku referencyjnego. Jeżeli sygnał referencyjny był o częstotliwości innej niż 1kHz, lub użyty został mikrofon do którego wymagane jest wprowadzenie poprawki do odczytu z pistofonu poprawka ta musi zostać wprowadzona w tym miejscu. Dozwolona jest dowolna liczba zmiennoprzecinkowa mieszcząca się w zakresie typu float.
6. To próg wykrycia zdarzenia. Program odnajdzie wszystkie fragmenty pliku, który poziom mierzony w dB SPL przekroczy zadaną wartość. Dozwolona jest dowolna liczba zmiennoprzecinkowa mieszcząca się w zakresie typu float.

Należy zwrócić uwagę na strukturę folderu w którym trzymane są pliki do analizy.

Przykładowa zawartość katalogu przedstawiona została na rysunku 6.2

Rysunek 6.2 Przykładowa struktura katalogu z plikami wave

Nazwa	Nr	Tytuł
 REFERENCE.wav	2	sin_1k_114dB_SPL
 sin_1k_94dB_SPL.W...	1	SVAN 959 SN:14725
 sin_1k_94dB_SPL_1....	1	REFERENCE
 SVAN0005.WAV	1	SVAN 959 SN:14725
 SVAN0006.WAV	1	SVAN 959 SN:14725
 SVAN0007.WAV	1	SVAN 959 SN:14725
 SVAN0008.WAV	1	SVAN 959 SN:14725
 SVAN0009.WAV	1	SVAN 959 SN:14725
 SVAN0010.WAV	1	SVAN 959 SN:14725
 SVAN0011.WAV	1	SVAN 959 SN:14725
 SVAN0012.WAV	1	SVAN 959 SN:14725
 SVAN0013.WAV	1	SVAN 959 SN:14725
 SVAN0014.WAV	1	SVAN 959 SN:14725

Na szczególną uwagę zasługuje plik REFERENCE.wav. Jest on niezbędny do zadziałania programu. Powinien znajdować się w nim sygnał referencyjny o znanym poziomie ciśnienia akustycznego. Nie może zawierać w sobie ciszy. Reszta plików może posiadać dowolne nazwy i być dowolnej długości. Jeżeli detekcja ma odbyć się prawidłowo niezbędne

jest aby wszystkie pliki były nagrane tym samym zestawem pomiarowym bez zmian nastaw w czasie nagrań. Jeżeli wystąpi konieczność zmiany wzmocnienia podczas nagrania, należy nagrać nowy plik referencyjny i rozdzielić nagrania na dwa katalogi dzieląc je według parametrów nagrania. Po uruchomieniu programu możemy spodziewać się powstania katalogów ze znalezionymi zdarzeniami podobnych do tych pokazanych na rysunku 6.3

Rysunek 6.3 Przykładowy katalog zawierający podkatalogi ze znalezionymi zdarzeniami





Nazwa	Nr	Tytuł
SVAN0005_events		
SVAN0006_events		
SVAN0007_events		
SVAN0009_events		
SVAN0011_events		
SVAN0012_events		
SVAN0013_events		
SVAN0014_events		
REFERENCE.WAV	1	SVAN 959 SN:14725
sin_1k_94dB_SPL_1.wav	1	REFERENCE
sin_1k_114dB_SPL.wav	2	sin_1k_114dB_SPL
SVAN0005.WAV	1	SVAN 959 SN:14725
SVAN0006.WAV	1	SVAN 959 SN:14725
SVAN0007.WAV	1	SVAN 959 SN:14725
SVAN0008.WAV	1	SVAN 959 SN:14725
SVAN0009.WAV	1	SVAN 959 SN:14725
SVAN0010.WAV	1	SVAN 959 SN:14725
SVAN0011.WAV	1	SVAN 959 SN:14725
SVAN0012.WAV	1	SVAN 959 SN:14725
SVAN0013.WAV	1	SVAN 959 SN:14725
SVAN0014.WAV	1	SVAN 959 SN:14725

A zawartość każdego z nich powinna zawierać pliki .wav zawierające zdarzenia akustyczne oraz plik .json podsumowujący wyszukiwanie. Przykładowy wygląd podkatalogu prezentuje rysunek 6.4

6.1 Zachowanie podczas normalnego działania

Podczas normalnego działania program wypisuje do wiersza poleceń informacje o stanie jego działania. W razie błędu pomaga to zorientować się co go powoduje. Na rysunku 6.5 możemy zobaczyć przykład poprawnego działania programu a na rysunku 6.6 przykład wypisania informacji o błędzie spowodowanego nieprawidłowym wywołaniem programu.

Rysunek 6.4 Przykładowa struktura podkatalogu zawierającego zdarzenia

Nazwa	Nr	Tytuł
 event_1.wav		
 event_2.wav		
 event_3.wav		
 SVAN0013_events.js...		

Rysunek 6.5 Przykład wiersza poleceń podczas poprawnego działania programu

```

Wiersz polecenia - F:\Thesis\Thesis\AudioEventDetector\main.py F:\Thesis\TestFiles_94_SPL A slow 94 72
Slow constant is applied
2204 samples has been converted to 0 samples with slow time constant

0 energy unit samples has been converted to 0 samples dB FS
Full scale level is 32768

0 dB FS A slow samples has been converted to 0 dB SPL A slowsamples
Reference value was -49.81 dB FS A slow with was equivalent to 94.0 dB SPL A slow

Read samples from 2882204 to 4322204
Samples from 2882204 to 2882204 has been read
End reading SVAN0006.WAV. Read 2882204 frames

Events found are: [(30, 58, 28)]

Found 1 events

Read frames from 1440000 to 2784000
Samples from 1440000 to 2784000 has been read.

Wrote file F:\Thesis\TestFiles_94_SPL\SVAN0006_events\event_1.wav
JSON file with events has been saved to: F:\Thesis\TestFiles_94_SPL\SVAN0006_events\SVAN0006_events.json

Read file F:\Thesis\TestFiles_94_SPL\SVAN0007.WAV
frame rate is 48000, chunk size is 1440000
total length is 60.05 s
Read samples from 0 to 1440000
Samples from 0 to 1440000 has been read

```

Rysunek 6.6 Przykład wiersza poleceń podczas wykrycia błędu

```

Wiersz polecenia
Read samples from 1440000 to 2880000
Samples from 1440000 to 1549638 has been read

Slow constant is applied
109638 samples has been converted to 2 samples with slow time constant

2 energy unit samples has been converted to 2 samples dB FS
Full scale level is 32768

Read samples from 1549638 to 2989638
Samples from 1549638 to 1549638 has been read
End reading REFERENCE.wav. Read 1549638 frames

-1594.0339608656197
32
Convert reference file return: -49.81356127705062

ERROR:root:Threshlod must be a number, not AA
Traceback (most recent call last):
  File "F:\Thesis\Thesis\AudioEventDetector\main.py", line 51, in <module>
    events, time_constant = find_events(file_path, reference_db_fs_value)
  File "F:\Thesis\Thesis\AudioEventDetector\main.py", line 27, in find_events
    threshold = CmdInterface.get_threshold()
  File "F:\Thesis\Thesis\AudioEventDetector\CmdInterface.py", line 83, in get_threshold
    threshold = float(threshold)
ValueError: could not convert string to float: 'AA'

```


Rozdział 7

Testy

W celu weryfikacji działania programu przeprowadzono testy. Próbki dźwiękowe nagrano przy pomocy miernika poziomu dźwięku Svantek 959 kalibrowanego kalibratorem Larson Davis CAL200. W celu kalibracji nagrano 3 pliki dźwiękowe, 2 z poziomem 94 dB SPL oraz jeden z poziomem 114dB SPL. Następnie nagrano różne zdarzenia akustyczne, następujące po sobie w zmieniającej się kolejności. Zdarzenia te to:

- działanie odkurzacza,
- działanie wiatraka,
- uderzanie młotem w kowadło,
- odtwarzanie muzyki z głośników laptopa,
- wiatrak od komputera.

Nagrano łącznie 9 nagrań o różnej długości. Dodatkowo aby sprawdzić działanie programu podczas przetwarzania długich plików, za pomocą programu Audacity stworzono jeden ponad pięciogodzinny plik składający się z kombinacji poprzednich, krótkich nagrań.

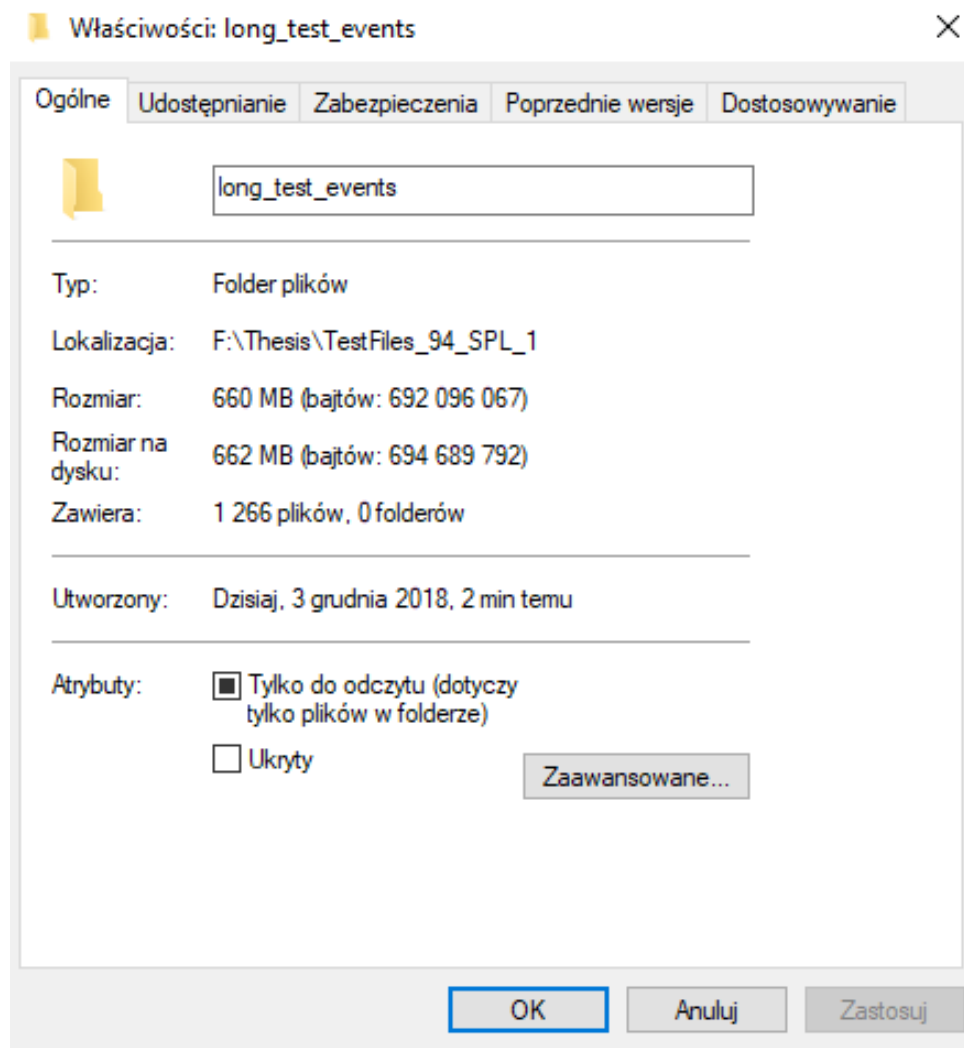
Poza czasie pracy program nie wykazywał różnic w zachowaniu przy odczycie i zapisie plików krótkich jak i pliku długiego. Pomimo bardzo długiego plików zawierających wiele zdarzeń program nie obciążał urządzeń mocniej niż podczas normalnej pracy. Folder utworzony podczas pracy prezentuje rysunek 7.1

W celu weryfikacji czy program odnalazł tylko fragmenty zawierające zdarzenia akustyczne, posłużono się wizualizacją dostępną w programie Audacity. Przykłady znalezionych zdarzeń prezentuje rysunek 7.2. Zdarzenia zostały znalezione w pliku

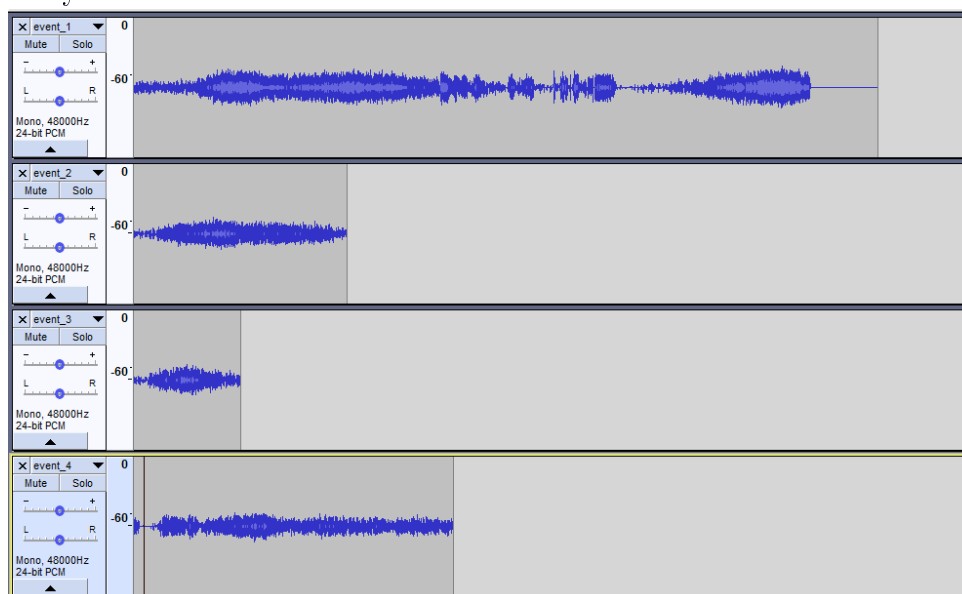
"SVAN0007.wav" którego pełny przebieg wraz z zaznaczonymi obszarami gdzie zostały odnalezione zdarzenia oznaczono kolorem czerwonym pokazuje 7.3. Podany przykład został przetestowany z parametrami "A slow 94 85"

Testy zostały przeprowadzone dla tych samych plików używając referencji z pliku 94 dB SPL oraz 114 dB SPL. W obu przypadkach program wyszukiwał tyle samo zdarzeń akustycznych we wszystkich plikach przy tym samym progu zadziałania. Testowano również stałe czasowe i różne krzywe ważenia. Dla wszystkich kombinacji program działał zgodnie z oczekiwaniami.

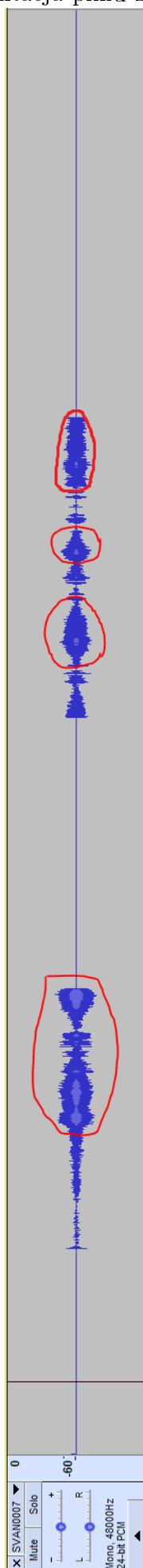
Rysunek 7.1 Folder utworzony po przeszukaniu pliku o długości ponad 5 godzin



Rysunek 7.2 Zdarzenia akustyczne prezentowane w formie graficznej przy użyciu programu Audacity



Rysunek 7.3 Wizualna reprezentacja pliku z którego ekstrahowano zdarzenia



Bibliografia

- [Audio Engineering Society, Inc, 2018] Audio Engineering Society, Inc (2018). *AES17-2015, AES standard method for digital audio engineering — Measurement of digital audio equipment*.
- [Drakos and Moore, 2003] Drakos, N. and Moore, R. (2003). Przewodnik po języku python, 6.moduły. <https://pl.python.org/docs/tut/node8.html>. Accessed: 2010-09-30.
- [Ecma International, 2017] Ecma International (2017). *ECMA-404 2 nd Edition / December 2017*.
- [Ewing, 2009] Ewing, G. (2009). Syntax for delegating to a subgenerator. <https://www.python.org/dev/peps/pep-0380/>. Accessed: 2010-09-30.
- [Gündert, 2014] Gündert, S. (2014). Spectral weighting filter. <http://sigfigue.github.io/pyfilterbank/splweighting.html>. Accessed: 2010-09-30.
- [IEC-61672-2013, 2013] IEC-61672-2013 (2013). *Electroacoustics - Sound level meters - Part 1: Specifications*.
- [Lutz, 2011] Lutz, M. (2011). *Wprowadzenie Python*. Helion.
- [Martin, 2014] Martin, R. C. (2014). *Czysty Kod Podręcznik dobrego programisty*. Helion.
- [PN-EN 61672-1:2014-03, 2015] PN-EN 61672-1:2014-03 (2015). *Elektroakustyka – Mierzenie poziomu dźwięku – Część 1: Wymagania*.
- [PN-ISO-1996-1:2006, 2006] PN-ISO-1996-1:2006 (2006). *Akustyka – Opis, pomiary i ocena hałasu środowiskowego – Część 1: Wielkości podstawowe i procedury oceny*.
- [PN-ISO-9612:2011, 2011] PN-ISO-9612:2011 (2011). *Akustyka – Wyznaczanie zawodowej ekspozycji na hałas – Metoda techniczna*.
- [Pohlmann, 2000] Pohlmann, K. C. (2000). *Principles of Digital Audio*. Mc Graww Hill, fourth edition.
- [Python Software Foundation, 2016] Python Software Foundation (2016). Pep 373 – python 2.7 release schedule. <https://legacy.python.org/dev/peps/pep-0373/>. Accessed: 2010-09-30.
- [Python Software Foundation, 2018a] Python Software Foundation (2018a). Creation of virtual environments. <https://docs.python.org/3/library/venv.html>. Accessed: 2010-09-30.

- [Python Software Foundation, 2018b] Python Software Foundation (2018b). Python 3.7.1 documentation, data structures. <http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.html>. Accessed: 2010-09-30.
- [Python Software Foundation, 2018c] Python Software Foundation (2018c). Python 3.7.1 generator objects. <https://docs.python.org/3/c-api/gen.html>. Accessed: 2010-09-30.
- [Python Software Foundation, 2018d] Python Software Foundation (2018d). Python 3.7.1 latest releases. <https://www.python.org/downloads/>. Accessed: 2010-09-30.
- [Python Software Foundation, 2018e] Python Software Foundation (2018e). The python package index. <https://pypi.org/>. Accessed: 2010-09-30.
- [Rasmussen College, 2017] Rasmussen College (2017). 10 best programming languages based on earnings & opportunities. <https://www.rasmussen.edu/degrees/technology/blog/best-programming-languages-based-on-earnings-and-opportunities/>. Accessed: 2010-09-30.
- [Reitz and schulsser, 2018] Reitz, K. and schulsser, T. (2018). *Przewodnik po Pythonie*. Helion.
- [Rietdijk, 2013] Rietdijk, F. (2013). python acoustics. <http://python-acoustics.github.io/python-acoustics/>. Accessed: 2010-09-30.
- [Software Freedom Conservancy, 2018] Software Freedom Conservancy (2018). Git, wprowadzenie. <https://git-scm.com/book/pl/v1/Pierwsze-kroki-Wprowadzenie-do-kontroli-wersji>. Accessed: 2010-09-30.
- [Strassner, | Strassner, T. Xml vs json. https://www.cs.tufts.edu/comp/150IDS/final_papers/tstras01.1/FinalReport/FinalReport.html. Accessed: 2010-09-30.
- [van Rossum et al., 2001] van Rossum, G., Warsaw, B., and Coghlan, N. (2001). Style guide for python code. <https://www.python.org/dev/peps/pep-0008/#names-to-avoid>. Accessed: 2010-09-30.
- [Zieliński, 2005] Zieliński, T. P. (2005). *Cyfrowe przetwarzanie sygnałów Od teorii do zastosowań*. Wydawnictwa Komunikacji i Łączności Warszawa.

Spis rysunków

3.1	Graficzny interfejs użytkownika GIT	10
3.2	Graficzny interfejs użytkownika programu GitKraken	11
4.1	Przykładowe wywołanie programu w konsoli systemu Windows	13
4.2	Struktura plików w programie	14
6.1	przykładowe wywołanie z wiersza poleceń	25
6.2	Przykładowa struktura katalogu z plikami wave	26
6.3	Przykładowy katalog zawierający podkatalogi ze znalezionymi zdarzeniami	27
6.4	Przykładowa struktura podkatalogu zawierającego zdarzenia	28
6.5	Przykład wiersza poleceń podczas poprawnego działania programu	28
6.6	Przykład wiersza poleceń podczas wykrycia błędu	28
7.1	Folder utworzony po przeszukaniu pliku o długości ponad 5 godzin	30
7.2	Zdarzenia akustyczne prezentowane w formie graficznej przy użyciu programu Audacity	30
7.3	Wizualna reprezentacja pliku z którego ekstrahowano zdarzenia	31

Listings

5.1	fragment kodu źródłowego pliku WaveFileSearcher.py	16
5.2	fragment kodu źródłowego pliku WaveFileReader.py	17
5.3	fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDbFSConverter	17
5.4	fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDb- SPLConverter	18
5.5	fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDbFSConverter, metoda _convert_samples_to_db_fs . . .	18
5.6	fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDbFSConverter, metoda _filter_db_samples_with_time_constant	19
5.7	fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDbFSConverter, metoda _filter_samples_with_weighting_filter	19
5.8	fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDb- SPLConverter, metoda convert_samples	20
5.9	fragment kodu źródłowego pliku SampleSPLConverter.py, klasa SamplesDb- SPLConverter, metoda convert_samples	20
5.10	fragment kodu źródłowego pliku SampleSPLConverter.py, klasa SamplesDb- SPLConverter, metoda convert_samples_to_db_spl	21
5.11	fragment kodu źródłowego pliku Detectors.py, klasa ThresholdCrossDetector, metoda count_occurrence	22
5.12	fragment kodu źródłowego pliku Detectors.py, klasa EventsOrganiser, metoda organise_events	22
5.13	fragment kodu źródłowego pliku JsonEventWriter.py, klasa JsonEventsWriter	23
5.14	Przykładowa zawartość pliku .json	23
5.15	fragment kodu źródłowego pliku WaveReader.py, klasa WaveWriter, metoda read_defined_frames	24
5.16	fragment kodu źródłowego pliku WaveReader.py, klasa WaveWriter, metoda write_defined_frames	24