

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

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	7
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	Sposób działania oraz struktura 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	22
5.5.1	JsonEventsWriter	23
5.5.2	Ekstrakcja z plików wave	24
6	Użytkowanie programu	25
6.1	Sposób użycia	25
6.2	Zachowanie podczas normalnego działania	27
7	Testy	29
8	Podsumowanie	33
	Bibliografia	34

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 Herców i czterdzieści osiem (48) kilo Herców. Oprogramowanie zostało przetestowane używając plików o tych parametrach. Oprogramowanie zostało stworzone modułarnie 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 języki 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 perspektywę 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 $1GB \approx 8 * 10^9 [b]$, 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 czytywałby cały plik do pamięci RAM, potrzebowałby 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 równoległe 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, zdarzenia 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

Autor nie spotkał się z rozwiązaniami programowymi, realizującymi podobną funkcjonalność. Niektóre mierniki poziomu dźwięku w podstawowym stopniu oferują tego typu rozwiązania, jednak autor nie znalazł informacji o żadnym, który umożliwia takie przetwarzanie na zapisanych uprzednio plikach.

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 odpowiedniego 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 często 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

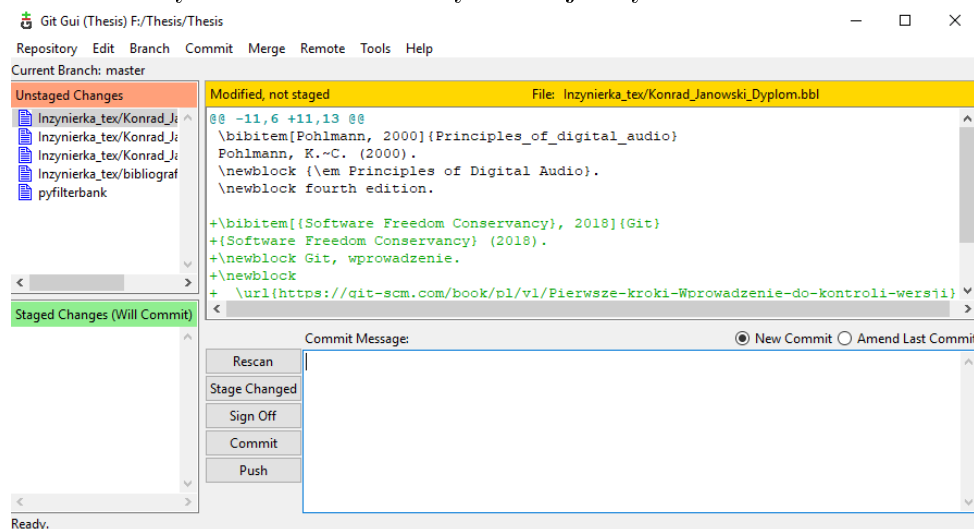
Jak zostało powiedziane, autor zdecydował się na skorzystanie z Git'a jako systemu do kontroli wersji. Jest to bardzo popularny system. Oferuje prostotę obsługi, przejrzystość, dobrą możliwość współpracy w małych grupach. Dodatkowo twórcy Git'a oferują darmowe prywatne miejsca na serwerze do przechowywania 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 zawiłych sposobów obsługi oprogramowania towarzyszącego.

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

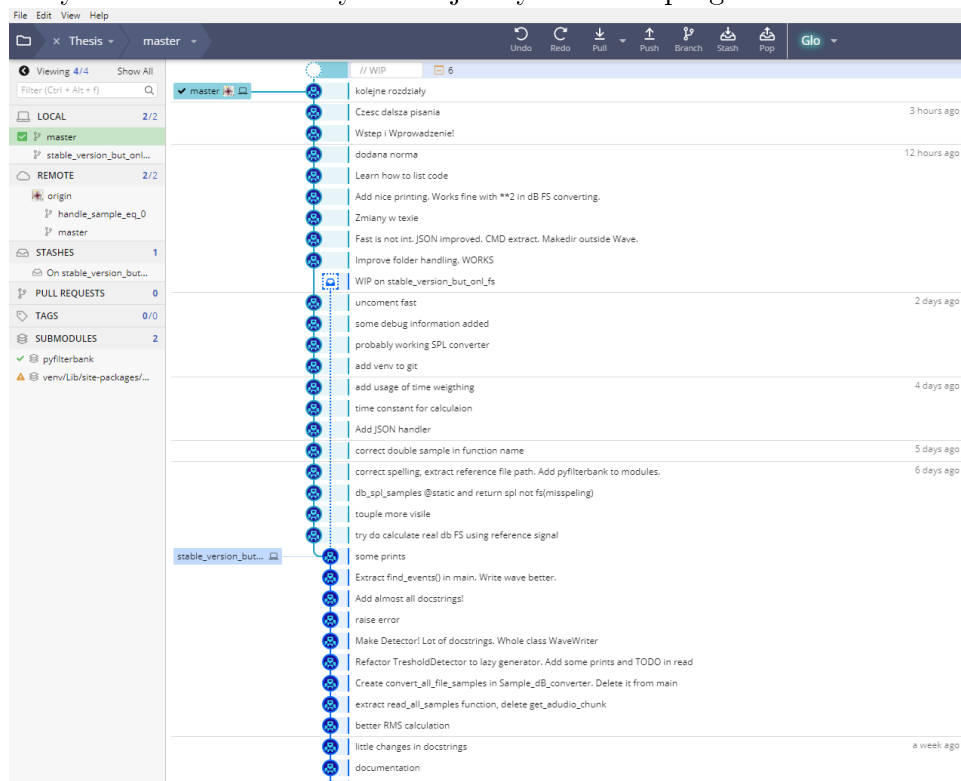
Rysunek 3.1 Graficzny interfejs użytkownika GIT



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 Rysunku 3.1.3.

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 zaniedbuje 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

Rysunek 3.2 Graficzny interfejs użytkownika programu GitKraken



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 intuicja 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, choć jego pełna nazwa brzmi "JavaScript Object Notation" jest uniwersalnym, popularnym sposobem wyświetlania i przekazywania danych. O jego wyborze przesądziła jego rosnąca popularność i prostota użycia [Strassner, bdw]. 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

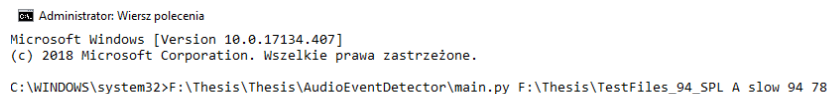
Sposób działania oraz struktura programu

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 Rysunku 4.1. Parametry wywołania i struktura działania programu zostaną omówione w dalszej części pracy.

Rysunek 4.1 Przykładowe wywołanie programu w konsoli systemu Windows



```
Administrator: Wiersz polecenia
Microsoft Windows [Version 10.0.17134.407]
(c) 2018 Microsoft Corporation. Wszelkie prawa zastrzeżone.
C:\WINDOWS\system32>F:\Thesis\Thesis\AudioEventDetector\main.py F:\Thesis\TestFiles_94_SPL A slow 94 78
```

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.

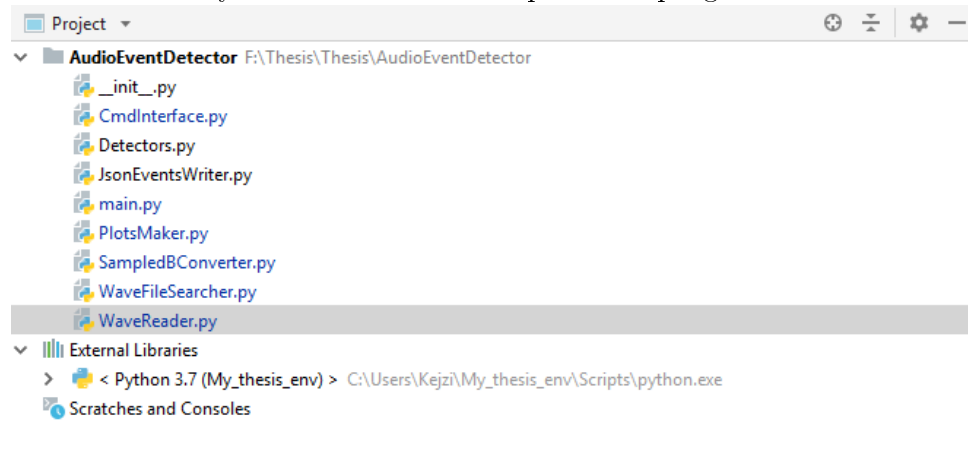
4.2 Struktura programu

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 do pliku .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.

Rysunek 4.2 Struktura plików w programie



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 "StylWielbłąda". 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. Dołożono wszelkich starań, żeby funkcje były omówione w kolejności jak najbardziej zbliżonej do logiki programu. Należy mieć jednak na uwadze, że niektóre funkcje wykorzystywane są kilkukrotnie 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

Listing 5.1 przedstawia metodę odnajdującą pliki wave w zadanym folderze.

```

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 linii 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

Na Listingu 5.2. został przedstawiony fragment kodu realizujący funkcję odczytywania odnalezionego 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 {}'.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.

Znajdują się tam 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 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 funkcjonalność 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     -----
7         energy_samples: [float]
8             list of samples in energy unit (e.x p^2).
9     Returns
10    -----
11         list(samples_db_fs): [float]
12         list of samples in dB FS format.
13     """
14     max_value = 2**((self.wave_reader_object.sample_width*8-1)
15     samples_db_fs = [10 * self._log10_dealing_with_0((sample/max_value**2)*np.sqrt(2))
16                     for sample in energy_samples]
17     print('{} energy unit samples has been converted
18     to {} samples dB FS'.format(len(energy_samples), len(samples_db_fs)))
19     print('Full scale level is {}'.format(max_value))
20     print('')
21     return list(samples_db_fs)

```

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 [AES17-2015, 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(
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
26
```

```

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 metodę, 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 metodę 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.
9          Must contain list of all events to organise where first value of every
10         tuple is value of sample, second is number of it position in file.
11         Must contain every starts 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,
16         endings and lengths of all founded events.
17 """
18 time_constant_ms = {'slow': 1,
19                     'fast': 0.125}
20 events_starts = [time*time_constant_ms[time_constant] for _, time in events[::2]]
21 events_ends = [time*time_constant_ms[time_constant] for _, time in events[1::2]]
22 events_length = []
23 events_starts_ends_lengths = []
24 for event_number in range(len(events_starts)):
25     try:
26         events_length.append(events_ends[event_number] - events_starts[event_number])
27     except IndexError:
28         print('there is an event which has only start')
29         break
30     events_starts_ends_lengths.append((events_starts[event_number],
31                                       events_ends[event_number],
32                                       events_length[event_number]))
33
34 if events_starts_ends_lengths:
35     print('Events found are: {}'.format(events_starts_ends_lengths))
36     print('')
37 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 umożliwiającą 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

Przeanalizujmy działanie klasy 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                                                '.json'))
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 wave

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

```

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

Odczytuje ona tylko te bity plik 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

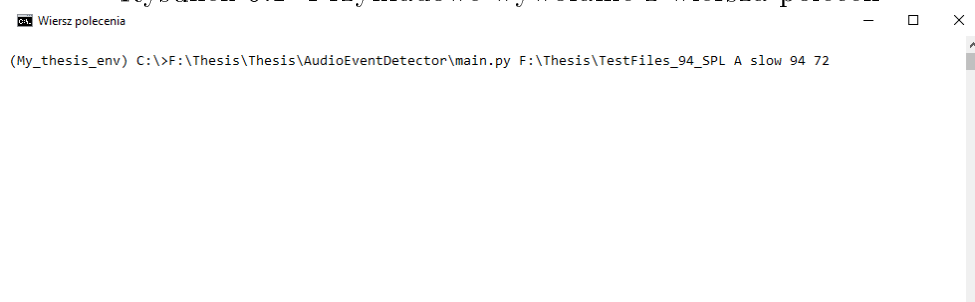
Użytkowanie programu

Ta część dotyczy sposobu użycia programu z punktu widzenia użytkownika.

6.1 Sposób użycia

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 1 kHz, 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.

6.2 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:

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

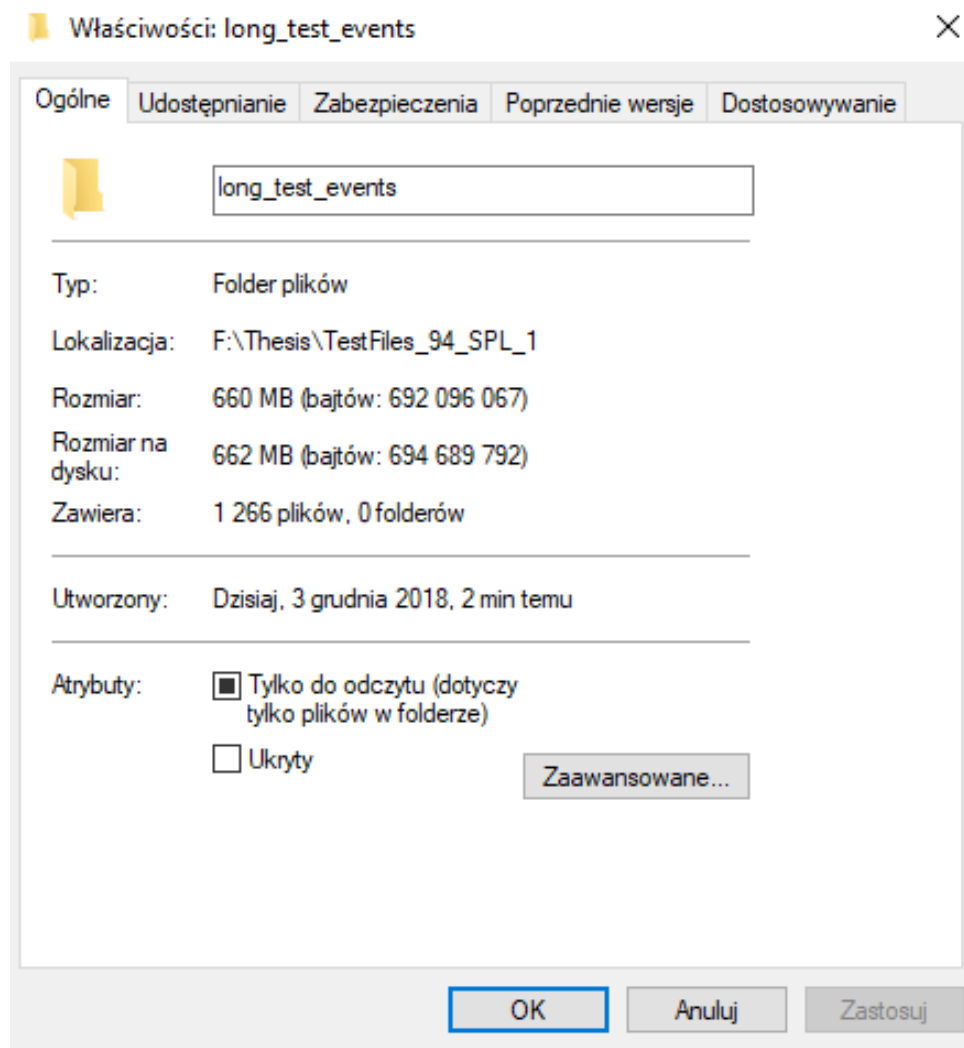
Nagrano łącznie 9 nagrań o różnej długości. Dodatkowo aby sprawdzić działanie programu podczas przetwarzania długich plików, za pomocą programu Audacity stworzono jeden ponad pięciogodzinny plik składający się z kombinacji poprzednich, krótkich nagrań.

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.

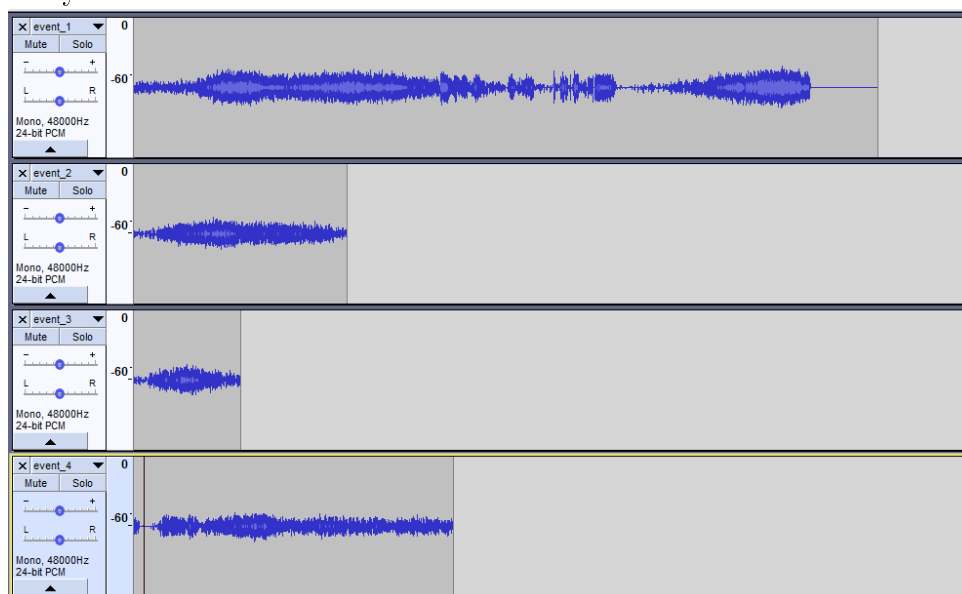
W celu weryfikacji czy program odnalazł tylko fragmenty zawierające zdarzenia akustyczne, posłużono się wizualizacją dostępną w programie Audacity. Przykłady znalezionych zdarzeń prezentuje Rysunek 7.2. Zdarzenia zostały znalezione w pliku "SVAN0007.wav" którego pełny przebieg wraz z zaznaczonymi obszarami gdzie zostały odnalezione zdarzenia oznaczono kolorem czerwonym pokazuje 7.3. Podany przykład został przetworzony 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.

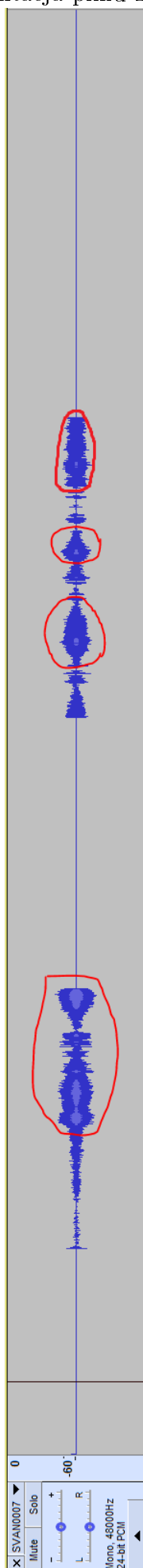
Rysunek 7.1 Folder utworzony po przeszukaniu pliku o długości ponad 5 godzin



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



Rysunek 7.3 Wizualna reprezentacja pliku z którego ekstrahowano zdarzenia



Rozdział 8

Podsumowanie

Celem pracy było stworzenie oprogramowania, które umożliwia detekcję i ekstrakcję zdarzeń akustycznych z materiału audio. Na początku zdecydowano o zawężeniu definicji zdarzenia akustycznego na potrzeby pracy. Zdarzenie akustyczne zdefiniowano jako przekroczenie pewnego poziomu ciśnienia akustycznego uśrednionego w czasie zgodnie zadaną stałą czasową i skorygowanego według zadanej krzywej korekcyjnej. Zagadnienie to zostało szerzej omówione w rozdziale 2.

Następnym krokiem było dobranie odpowiednich narzędzi do pracy. W rozdziale 3. zostały przedstawione narzędzie użyte do stworzenia pracy. Zdecydowano się na korzystanie z języka programowania Python w wersji 3.7.1, będącą najnowszą wersją języka podczas tworzenia oprogramowania. Dodatkowo autor zdecydował się na pracę z systemem kontroli wersji Git przy pomocy oprogramowania GitKraken. Do pisania programu użyto środowiska programistycznego Pycharm. Aby w pełni zrealizować zamierzone cele, wykorzystano również standard zapisu danych JSON do przechowywania informacji o odnalezionych zdarzeniach.

W rozdziale 4 przedstawiono sposób działania programu. Zdecydowano się na stworzenie aplikacji konsolowej. Omówiono również w podstawowym stopniu strukturę program.

Rozdział 5 w sposób szczegółowy omawia funkcjonalność programu. Przedstawione zostały listingi wybranych, najbardziej istotnych do zrozumienia logiki programu klas i metod. Autor tłumaczy jak przebiega konwersja od nieskompresowanego sygnału wave, prezentującego dynamiczny przebieg ciśnienia akustycznego w czasie na skorygowany częstotliwościowo uśredniony w czasie poziom ciśnienia akustycznego. Następnie objaśnia moduły odpowiedzialne za detekcję zdarzeń, ich organizację oraz sposób zapisu danych.

Następnie rozdział 6. opisuje sposób użytkowania programu z punktu widzenia użytkownika. Zostają objaśnione parametry wejściowe oraz przygotowanie katalogów niezbędne do prawidłowego funkcjonowania. Zaprezentowany został również przykład zachowania programu podczas normalnego działania, zarówno poprawnego jak i podczas wykrycia błędów użytkownika.

Pracę zakończyły testy oprogramowania omówione w rozdziale 7. Program został użyty do detekcji zdarzeń w plikach o różnym czasie trwania, różnej zawartości oraz przy użyciu różnych plików referencyjnych. Oprogramowanie podczas przeprowadzonych testów zachowywało się w sposób stabilny i działało zgodnie z oczekiwaniami.

Efektom pracy jest oprogramowanie, działające jako aplikacja konsolowa. Może być używane jako samodzielne narzędzie lub użyte jako element większego systemu. Oprogramowanie potrafi rozpoznać w podanym pliku wave zdarzenia akustyczne, przekraczające zadany poziom dźwięku. Po detekcji następuje zapisanie wyników z ilością zdarzeń, oraz informacji gdzie się zaczynają, kończą oraz jaki jest ich czas trwania. Dodatkowo zdetekowane zdarzenia zostają zapisane jako pliki wave. Oprogramowanie przeszło podstawowe testy i działa zgodnie z oczekiwaniami. Cel pracy został zatem zrealizowany.

Oprogramowanie poprzez swoją modułarną budowę oraz dbałość autora o czytelność kodu daje możliwości dalszego rozwoju. W przyszłości można rozbudować funkcjonalności progra-

mu o dodatkowe moduły, biorące pod uwagę bardziej szczegółowe cechy zdarzeń dźwiękowych. Najprostszym i najbardziej pożądanym, zdaniem autora, usprawnieniem byłoby dodanie odnajdywania zdarzeń akustycznych na podstawie ich relatywnego poziomu względem tła, nie zaś bezwzględnego odgórnie ustalanego jak to ma miejsce w stworzonym oprogramowaniu. Dodatkowo można rozbudowywać je o moduły odnajdywania hałasów tonalnych, o maksimach w wybranych pasmach częstotliwościowych, jedynie o zadanym czasie trwania itd. Dodatkowym usprawnieniem mogłaby być również funkcjonalność odnajdywania wydarzeń nagrywanych w czasie rzeczywistym. Przetwarzanie zdarzeń małymi fragmentami daje dobre fundamenty do implementacji dodatkowych modułów działających w czasie rzeczywistym. Te i inne usprawnienia z pewnością mogą sprawić, że oprogramowanie ma szansę stać się narzędziem użytecznym w praktyce inżynierskiej.

Bibliografia

- [AES17-2015, 2018] AES17-2015 (2018). *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://siggigue.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 – Mierniki 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, bdw] Strassner, T. (bdw). 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

Listingi

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 SamplesDbFSCConverter	17
5.4	Fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDbSPLConverter	18
5.5	Fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDbFSCConverter, metoda _convert_samples_to_db_fs	18
5.6	Fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDbFSCConverter, metoda _filter_db_samples_with_time_constant	19
5.7	Fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDbFSCConverter, metoda _filter_samples_with_weighting_filter	19
5.8	Fragment kodu źródłowego pliku SampledBConverter.py, klasa SamplesDbSPLConverter, metoda convert_samples	20
5.9	Fragment kodu źródłowego pliku SampleSPLConverter.py, klasa SamplesDbSPLConverter, metoda convert_samples	20
5.10	Fragment kodu źródłowego pliku SampleSPLConverter.py, klasa SamplesDbSPLConverter, metoda convert_samples_to_db_spl	21
5.11	Fragment kodu źródłowego pliku Detectors.py, klasa ThresholdCrossDetector, metoda count_occurrence	21
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