Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki
Inżynieria Akustyczna (studia międzykierunkowe)

KATEDRA ELEKTRONIKI

PRACA INŻYNIERSKA

JAKUB PISZCZEK

ADAPTACYJNY SYSTEM REDUKCJI SZUMU AKUSTYCZNEGO

PROMOTOR:
dr inż. Jakub Gałka

Kraków 2012

OŚWIADCZENIE AUTORA PRACY

OŚWIADCZAM, ŚWIADOMY ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ WYKONAŁEM OSOBIŚCIE I SAMODZIELNIE, I NIE KORZYSTAŁEM ZE ŹRÓDEŁ INNYCH NIŻ WYMIENIONE W PRACY.

. . . . . .

PODPIS

AGH

University of Science and Technology in Krakow

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

DEPARTMENT OF ELECTRONICS

B.SC. THESIS

JAKUB PISZCZEK

ADAPTIVE ACOUSTIC NOISE REDUCTION SYSTEM

SUPERVISOR:
Jakub Gałka Ph.D

Krakow 2012

Składam serdeczne podziękowania Panu dr inż. Jakubowi Gałce za udostępnienie ciekawego tematu pracy oraz cenne uwagi przekazane w trakcie jej realizacji.

Spis treści

1. Wprowadzenie

W komunikacji głosowej mowie zawsze towarzyszą zakłócenia, których główną składową są zazwyczaj odgłosy pochodzące z otoczenia, w którym znajduje się osoba mówiąca. Naturalnym skutkiem występowania tła akustycznego jest degradacja jakości sygnału mowy powodująca utrudnienia w jej zrozumieniu przez odbiorcę. W celu poprawy jakości (zrozumiałości) rejestrowanej mowy wykorzystuje się algorytmy redukcji szumu. Ich zastosowanie może być również motywowane przygotowaniem sygnału do dalszego przetwarzania np. kompresji. Bardzo popularną metodą redukcji szumu w systemach jednomikrofonowych jest odejmowanie widmowe (ang. *spectral subtraction*). Algorytm ten wyznacza estymację sygnału mowy poprzez obliczenie różnicy widm sygnału wejściowego oraz estymacji widma tła akustycznego. W niniejszej pracy został wykorzystany zmodyfikowany algorytm odejmowania widmowego zwiększający skuteczność redukcji szumu oraz minimalizujący powstawanie zniekształceń sygnału przetwarzanego.

2. Zastosowany algorytm

Zaszumiony sygnał $y(t)$ możemy przedstawić jako sumę sygnałów mowy $s(t)$ oraz szumu tła:

$$y(t) = s(t) + d(t) \tag{2.1}$$

Przeprowadzając transformacje Fouriera sygnał można zapisać jako sumę widm odpowiednich składowych:

$$Y(f) = S(f) + D(f) \tag{2.2}$$

Ze względu na niestacjonarną charakterystykę sygnału analiza częstotliwościowa jest przeprowadzana po poddaniu sygnału operacji ramkowania.

$$Y_w(f) = S_w(f) + D_w(f) \tag{2.3}$$

Aby móc przeprowadzić odejmowanie widm musimy wyznaczyć ich wartoci energii dla poszczególnych częstotliwości. Ostatecznie krótkoczasowe widmo energii sygnału zapiszemy następująco:

$$|Y_w(f)|^2 = |S_w(f)|^2 + |D_w(f)|^2 + S_w(f) \cdot D_w^*(f) + S_w^*(f) \cdot D_w(f) \tag{2.4}$$

gdzie $S_w^*(f)$ oraz $D_w^*(f)$ są wartościami sprzężonymi odpowiednio do $S_w(f)$ oraz $D_w(f)$. Przyjmując założenie, że szum jest nieskorelowany z sygnałem mowy, wyrażenie

$$S_w(f) \cdot D_w^*(f) + S_w^*(f) \cdot D_w(f) \tag{2.5}$$

jest równe zeru.

Jedynie sygnał zaszumiony jest mierzony bezpośrednio, więc zarówno widmo mowy jak i szumu zastępujemy ich estymacjami.

$$E[|S_w(f)|^2] = |Y_w(f)|^2 E[|D_w(f)|^2] \tag{2.6}$$

By uzyskać odszumiony sygnał w dziedzinie czasu wykorzystuje się widmo amplitudowe estymacji sygnału mowy oraz widmo fazowe sygnału zaszumionego.

Poprzez wykonanie odejmowania widmowego wprost można uzyskać ujemne wartości widma sygnału. Aby zminimalizować zniekształcenia wywołane nad estymacją wartości widma szumu wprowadzono parametr progu widma $\beta = 0.01$ [1].

$$E[|S_w(f)|^2] = \begin{cases} E[|S_w(f)|^2] & \text{gdy } E[|S_w(f)|^2] \geq \beta \cdot E[|D_w(f)|^2] \\ \beta \cdot E[|D_w(f)|^2] & \text{gdy } E[|S_w(f)|^2] < \beta \cdot E[|D_w(f)|^2] \end{cases} \tag{2.7}$$

Zwiększenie skuteczności usuwania szumu można uzyskać poprzez wprowadzenie mnożnika widma szumu [2].

$$E[|S_w(f)|^2] = |Y_w(f)|^2 - \alpha \cdot E[|D_w(f)|^2] \tag{2.8}$$

Jeśli α będzie miało zbyt dużą wartość to zniekształcenia spowodowane odejmowaniem znacznie pogorszą jakość sygnału mowy, natomiast gdy wartość α będzie zbyt mała to algorytm pozostawi w sygnale znaczne wartości niepożądanego szumu. Optymalne wykorzystanie współczynnika α można uzyskać poprzez uzależnienie jego wartości od stosunku sygnału do szumu (SNR). Wartość SNR jest definiowana następująco:

$$SNR = 10 \cdot log_{10}(\frac{E[|S_w(f)|^2]}{E[|D_w(f)|^2]}) \tag{2.9}$$

Wartość α jest aktualizowana z ramki na ramkę na podstawie wartości SNR dla poprzedniej ramki przetwarzanego sygnału. Zakładamy maksymalna wartość $\alpha_{max} = 5$ dla $SNR = -5[dB]$, a minimalną $\alpha_{min} = 1$ dla $SNR = 20$[dB]. [2].

$$\alpha_i = \begin{cases} 5 & \text{gdy } SNR_{i-1} \leq -5 \\ 4.2 - 0.16 \cdot SNR_{i-1} & \text{gdy } -5 < SNR_{i-1}\ 20 \\ 1 & \text{gdy } SNR_{i-1} \geq 20 \end{cases} \tag{2.10}$$

Biorąc pod uwagę fakt, że mowa jest nierównomiernie zakłócana przez szum tła w dziedzinie częstotliwości wprowadzono analizę sygnału w N pasmach częstotliwościowych [3]. Algorytm odejmowania jest przeprowadzany dla każdego pasma niezależnie z uwzględnieniem dodatkowego parametru δ zależnego od częstotliwości.

$$E[|S_w(f)|^2] = |Y_w(f)|^2 - \alpha_i \cdot \delta_i \cdot E[|D_w(f)|^2] \tag{2.11}$$

Do podziału widma sygnału na pasma częstotliwościowe niezależnie od częstotliwości próbkowania wykorzystano skalę melową [4] zakładając szerokość pasma 250[mel]. Wartości współczynników $\delta(m)$ dla poszczególnych pasm zostały wybrane empirycznie.

$$\delta(m) = \begin{cases} 1 & \text{gdy } m < 500 \\ 1.5 & \text{gdy } 500 \leq m < 750 \\ 2.0 & \text{gdy } 750 \leq m < 1000 \\ 2.5 & \text{gdy } 1000 \leq m < 1250 \\ 2.0 & \text{gdy } 1250 \leq m < 1500 \\ 1.5 & \text{gdy } 1500 \leq m < 2000 \\ 1.0 & \text{gdy } m \geq 2000 \end{cases} \tag{2.12}$$

3. Implementacja

3.1. FFTW

Podstawa zaimplementowanego algorytmu jest analiza częstotliwościowa sygnałów. Do obliczenia widm sygnału została wykorzystana biblioteka FFTW (`http://www.fftw.org`) udostępniana na warunkach GNU General Public License [5].

FFTW jest biblioteką generującą optymalna procedurę wyliczania transformacji Fouriera na podstawie pomiaru możliwości sprzętowych komputera, na którym jest uruchamiania [6]. Adaptacja procedury obliczania FFT może być przeprowadzona z różną dokładnością zależną od ustawionej flagi będącej argumentem funkcji wyznaczającej plan algorytmu.

```
fftw_plan fftw_plan_dft_1d(int n, fftw_complex *in, fftw_complex *out,
                                  int sign, unsigned flags)
```

Do obliczania szybkiej transformacji Fouriera wykorzystałem plan wygenerowany z ustawioną flagą `FFTW_PATIENT` pozwalającą na uzyskanie maksymalnej szybkości wykonywania obliczeń kosztem dłuższego czasu inicjalizacji.

3.2. VST

Algorytm odejmowania widmowego został zaimplementowany w standardzie VST ® (ang. *Virtual Studio Technology*), który został opracowany przez firmę Steinberg Media Technology ® . Standard ten określa interfejs komunikacji pomiędzy wtyczka przetwarzająca sygnały audio a oprogramowaniem kontrolującym tzw. hostem, jest to najczęściej oprogramowaniem typu DAW (ang. *Digital Audio Workstation*, np. Cubase, Samplitude itp) powszechnie używanym do rejestracji nagrań oraz realizacji produkcji dźwiękowych. Licencja na wykorzystanie technologii VST jest udzielana przez firmę Steinberg nieodpłatnie. Podstawowe wymagania stawiane przez standard VST[7] to:

- Możliwość przetwarzanie sygnałów stereofonicznych
- Przetwarzanie otrzymanych bloków próbek w czasie krótszym niż ich czas trwania (w przypadku gdy przetwarzanie zajmuje zbyt wiele czasu, wykonywana procedura zostanie przerwana i sterowanie zostanie przekazane z powrotem do miejsca obsługi odbierania nowych sampli)
- Wspierania przetwarzania 'w miejscu' - próbki wyjściowe są wstawiane w miejsce otrzymanych próbek wejściowych

– Poinformowanie hosta o opóźnieniu generowanym przez algorytm zaimplementowanym we wtyczce w postaci ilości sampli, które musi ona otrzymać przed rozpoczęciem działania - jest informacja umożliwiająca synchronizacje ścieżek przez host (pozwala to uniknąć występowania np. zjawiska filtru grzebieniowego)

– posiadanie przez wtyczkę unikatowego identyfikatora otrzymywanego po zarejestrowaniu wtyczki w serwisie internetowym firmy Steinberg

3.2.1. Inicjalizacja wtyczki

Inicjalizacja wtyczki odbywa się dwuetapowo: w czasie skanowania przez host ścieżek w poszukiwaniu wtyczek wywoływany jest konstruktor klasy, następnie w momencie uaktywnienia wtyczki wywoływana jest metoda `resume()`. Gdy host jest uruchamiany wywołuje on konstruktory wszystkich dostępnych wtyczek, więc jest bardzo ważne aby nie wykonywały one żadnych czasochłonnych operacji a jedynie te, które są niezbędne do określenia trybu oraz środowiska pracy. Alokacja pamięci przeznaczonej na bufory wewnętrzne oraz inicjalizaca pozostałych komponentów jest wykonywana w momencie wywołania funkcji `resume()`.

Kod konstruktora wtyczki oraz metody `resume()`:

```
Denoise::Denoise (audioMasterCallback audioMaster)
 : AudioEffectX (audioMaster, kNumPrograms, kNumParams),
    (.. inicjalizacja zmiennych ..)
    {
    // ustawienie bieżącego programu
    this->curProgram = 0;
    // przepisanie nazwy programu domylnego
    vst_strncpy (programName, "Default", kVstMaxProgNameLen);

    // ustawienie ilości wejść i wyjść wtyczki
    setNumInputs (kMaxNumOfChannels);
    setNumOutputs (kMaxNumOfChannels);
    // identyfikator wtyczki
    setUniqueID ('325k');
    // wspiera przetwarzanie 'w miejscu'
    canProcessReplacing (true);
    // wspiera podwójną precyzję
    canDoubleReplacing (true);
    // programy sš klasami określającymi stan wtyczki
    programsAreChunks(true);

    // alokacja ustawienia kanałów
    allocateArrangement(&plugInput,kMaxNumOfChannels);
```

```
    allocateArrangement(&plugOutput,kMaxNumOfChannels);

    }
//------------------------------------------------------------------
void Denoise::resume ()
    {
    // pobranie wartości częstotliwości próbkowania
    float sampleRate = this->getSampleRate();
    // próba wymuszenia przetwarzania z podwójna precyzja
    this->setProcessPrecision(kVstProcessPrecision64);
    // obliczenie ilości pasm melowych
    double mel = 2595.0*log10(sampleRate/1400.0 + 1);
    numOfBands = static_cast<int>(mel/250.0);
    // ustawienie długości buforów
    this->frameSize = static_cast<int>(sampleRate * 0.025);
    this->hopSize = static_cast<int>(frameSize * (1 - 0.4));
    this->leftoversSize = frameSize - hopSize;
    // poinformowanie hosta o opóźnieniu wprowadzanym przez wtyczkę
    this->setInitialDelay(hopSize);
    // inicjalizacja komponentów
    this->initializePlugin()

    return;
    }
```

3.2.2. Obsługa parametrów

Host przekazuje informacje do wtyczki poprzez interfejs klasy `AudioEffectX`. Za obsługę parametrów ustawianych przez użytkownika odpowiadają metody:

- `void setParameter (VstInt32 index, float value)`
- `float getParameter (VstInt32 index)`
- `void getParameterName (VstInt32 index, char* label)`
- `void getParameterDisplay (VstInt32 index, char* text)`
- `void getParameterLabel (VstInt32 index, char* label)`

Przesyłane parametrów są liczbami zmiennoprzecinkowymi przyjmującymi wartości od 0.0 do 1.0.

Kod zaimplementowanych funkcji obsługi parametrów:

```
void Denoise::setParameter (VstInt32 index, float value)
    {
    // pobierz aktualny program
    DenoiseProgram * p = &(bank.getProgram(curProgram));
    // wpisz przekazana wartość do wtyczki oraz bieżącego programu
    switch (index)
        {
        case kNoiseMargin:
            noiseMargin = static_cast<double>(value*value);
            p->noiseMargin = noiseMargin;
            break;
        case kGain:
            gain = static_cast<double>(value);
            p->gain = gain;
            break;
        }
    return;
    }
//------------------------------------------------
float Denoise::getParameter (VstInt32 index)
    {
    // zwróć wartość odpowiedniego parametru
    switch (index)
        {
        case kNoiseMargin:
            return static_cast<float>(sqrt(noiseMargin));
        case kGain:
            return static_cast<float>(gain);
        default:
            return 0.0f;
        }
    }
//------------------------------------------------
void Denoise::getParameterName (VstInt32 index, char* label)
    {
    // przepisz odpowiednia nazwę do przesłanej tablicy
    switch (index)
        {
        case kNoiseMargin:
            vst_strncpy (label, "Margin", kVstMaxParamStrLen);
```

```
            break;
        case kGain:
            vst_strncpy (label, "Gain", kVstMaxParamStrLen);
            break;
        }
    return;
    }
//-----------------------------------------------------
void Denoise::getParameterDisplay (VstInt32 index, char* text)
    {
    // przekaż wartość wyświetlaną przez host
    switch (index)
        {
        case kNoiseMargin:
            dB2string (static_cast<float>(noiseMargin),
                       text, kVstMaxParamStrLen);
            break;
        case kGain:
            dB2string (static_cast<float>(gain), text, kVstMaxParamStrLen);
            break;
        }
    return;
    }
//-----------------------------------------------------
void Denoise::getParameterLabel (VstInt32 index, char* label)
    {
    // przepisz jednostki odpowiedniego parametru
    switch (index)
        {
        case kNoiseMargin:
            vst_strncpy (label, "dB", kVstMaxParamStrLen);
            break;
        case kGain:
            vst_strncpy (label, "dB", kVstMaxParamStrLen);
            break;
        }
    return;
    }
```

3.2.3. Zapisywanie stanu wtyczki

Standard VST pozwala na zapisywanie ustawień wtyczki w pliku projektu, w którym jest wykorzystywana na dwa sposoby:

– poprzez zapisanie wartości parametrów dostępnych kontrolowanych przez użytkownika i ich przywrócenie poprzez wywołanie dla każdego z nich metody `setParameter(..)`

– poprzez zapisanie bloku bajtów określających stan wtyczki mogącego zawierać dowolne informacje zdefiniowane programistę

Pierwszy sposób zapisywania stanu wtyczki jest domyślny i wykonywany przez host automatycznie jeśli wtyczka nie zgłosiła, że jej stan jest określany przez inna strukturę danych (przez tzw. 'chunk'). Jeśli wtyczka wywołała w czasie uruchamiania wywołała funkcję `programsAreChunks(true)` to zapisanie oraz przywróceni jej stanu będzie się odbywało poprzez metody `setChunk(..)` i `getChunk(..)`. W wykonanej przeze mnie programie będącym tematem niniejszej pracy wykorzystałem drugą z wymienionych metod obsługi przechowywania stanu wtyczki. Zapisywanie oraz odczyt stanu mogą odbywać się w dwóch trybach zależnych od wartości atrybutu logicznego `isPreset` - gdy przyjmuje on wartość prawdziwą to zapisywany/odczytywany jest jeden program (chunk), a gdy fałszywą to zapisywany/odczytywany jest cały bank programów. Przesyłanym blokiem danych jest obiekt klasy `DenoiseProgram` badz `DenoiseProgramBank`.

Kod źródłowy zaimplementowanych metod obsługi zapisywania stanu wtyczki:

```
VstInt32 Denoise::setChunk(void* data,
    VstInt32 byteSize, bool isPreset)
    {

    if (isPreset)
        {
        // czy rozmiar bloku się zgadza
        if (byteSize != sizeof(DenoiseProgram))
            return 0;
        // próba zapisania programu do banku
        if(!bank.setProgram(static_cast<DenoiseProgram*>(data),
                                curProgram))
            return 0;
        }
    else
        {
        // czy rozmiar bloku się zgadza
        if (byteSize != sizeof(DenoiseProgramBank))
            return 0;
        // interpretujemy przesłany blok jako bank programów
```

```
        bank = *static_cast<DenoiseProgramBank*>(data);
        }
    setProgram(curProgram);
    return 1;
    }
//------------------------------------------------------------------------
VstInt32 Denoise::getChunk(void **data, bool isPreset)
    {
    // zapisanie obiektu programu do zadanego miejsca w pamięci
    if (isPreset)
        {
        *data = &(bank.getProgram(curProgram));
        return sizeof(DenoiseProgram);
        }
    else
        {
        *data = &bank;
        return sizeof(DenoiseProgramBank);
        }
    }
```

3.2.4. Obsługa strumienia

Host przekazuje próbki do wtyczki poprzez wywołanie metody `processReplacing(..)` bądź `processDoubleReplacing(..)`. Przekazując jako argumenty wskaźniki do miejsca w pamięci gdzie znajdują się próbki wejściowe oraz gdzie maja się znaleźć wyjściowe oraz ilość próbek, które wtyczka ma przetworzyć w tym wywołaniu. Ilość próbek przekazywanych do przetworzenia może być w każdym wywołaniu inna (jest to następstwo konieczności synchronizacji przekazywania parametrów między hostem a wtyczką z dokładnością do jednego sampla) ale nigdy nie jest większa niż wartość przechowywana w zmiennej `blockSize`.

Kod zaimplementowanej funkcji `processDoubleReplacing(..)`:

```
void Denoise::processDoubleReplacing (double** inputs,
    double** outputs, VstInt32 sampleFrames)
    {
    double* in;
    double* out;

    // dla każdego kanału
    for (int c = 0; c < plugInput->numChannels; c++)
        {
```

```
        in = inputs[c];
        out = outputs[c];

        // jeśli dane istnieją
        if ((!(in == 0)) && (!(out == 0)))
            {
            // dla wszystkich przekazanych próbek
            for (int i = 0; i < sampleFrames; i++)
                {
                // jeśli bufor wejściowy jest...
                // ... niepełny
                if (sampleIndex[c] < hopSize)
                    {
                    // wpisz próbkę do bufora wejściowego
                    inputBuffer[sampleIndex[c]] = in[i];
                    // wyślij próbkę z bufora wyjściowego
                    out[i] = outputBuffer[c][sampleIndex[c]++];
                    }
                // ... pełny
                else
                    {
                    sampleIndex[c] = 0;
                    // wykonaj algorytm
                    doProcess(c);
                    }
                }
            }
        }
    return;
    }
```

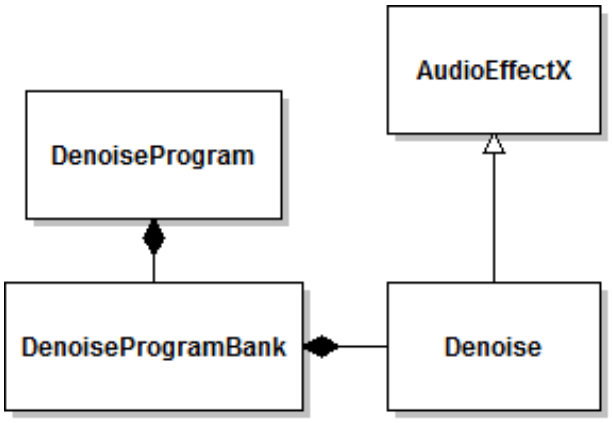
3.3. Struktura programu

Wykonany przeze mnie program składa się z trzech klas, o powiązaniach pokazanych na rys 1:

– `Denoise` - główna klasa dziedzicząca po klasie AudioEffectX będącej częścią VST SDK

– `DenoiseProgram` - klasa określająca stan (program) wtyczki zapisywany wraz z projektem, w którym została użyta

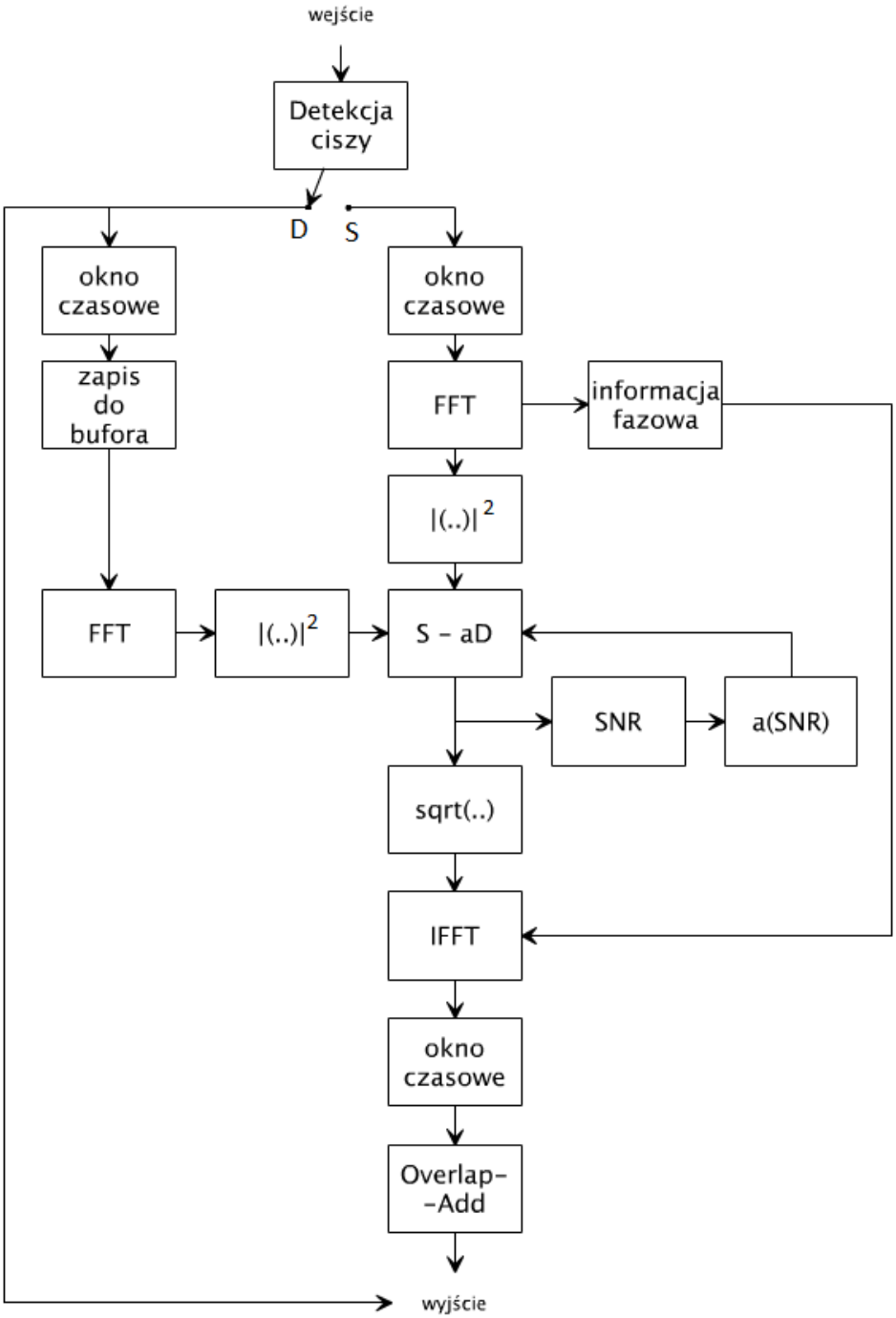
– `DenoiseProgramBank` - zbiór zapisanych programów wtyczki



Rysunek 3.1: Struktura programu

3.4. Implementacja algorytmu

Sygnał wejściowy jest dzielony na ramki o długości $25ms$ zakładkowane w stosunku 0.4 oraz mnożony przez okno czasowe będące, którego wartości są pierwiastkami z wartości okna cosinus-owego. Następnie na podstawie wartości energii sygnału zawartego w ramce jest podejmowana decyzja czy sygnał jest szumem bądź mowa. Jeśli jest szumem to próbki są zapisywane do bufora cyklicznego, w którym znajduje się ostatnie 10 ramek zawierających szum. Natomiast jeśli ramka zawiera sygnał mowy to aplikowany jest opisany w rozdziale 2 algorytm, przy czym widmo szumu jest średnia z widmem ramek znajdujących się w buforze cyklicznym. Korzystając z liniowości transformacji Fouriera [8] uśredniamy ramki zapisane w buforze cyklicznym w dziedzinie czasu i wykonujemy tylko jedną operację obliczania FFT co pozwala zaoszczędzić czas procesora. Sygnał jest przetwarzany z powrotem do dziedziny czasu z wykorzystaniem okna czasowego identycznego jak na wejściu oraz operacji overlap-add [9].



Rysunek 3.2: Schemat blokowy algorytmu

Kod źródłowy metody implementującej algorytm odejmowania widmowego:

```
void Denoise::doProcess (int channelNo)
    {
    bool signalType;
    // wpisujemy odpowiednie próbki do ramki
    fillFrame(channelNo);
    //-------------------------
    // sprawdzamy czy sygnał w ramce jest szumem
    signalType = determineSignalType(channelNo);
    //-------------------------
    // w zależności od typu sygnału:
    // ... gdy mamy do czynienia z szumem
    if (signalType == noise)
        {
        // uaktualniamy zawartość bufora cyklicznego
        updateCircularBuffer(channelNo);
        // wypełniamy bufor wyjciowy
        fillOutputBuffer(channelNo);
        }
    // ... gdy mamy do czynienia z mową
    else if (signalType == speech)
        {
        // czy obliczone widmo szumu jest aktualne...
        if (!noisePSisSet)
            {
            // obliczamy widmo mocy szumu na podstawie danych w buforze cyklic
            computeNoisePowerSpectrum(channelNo);
            }
        //--------------------
        // obliczamy widma sygnału mowy
        computeSpeechSpectrums();
        // odejmujemy widma
        subtractSpectrums(channelNo);
        // syntezujemy sygnał z przetworzonych ramek
        reconstructSignal(channelNo);
        // obliczamy wsp. odejmowania na podstawie aktualnej ramki
        computeAlpha(channelNo);
        // uaktualniamy bufor próbek do przetworzenia w następnym wywołaniu
        updateLeftovers(channelNo);
```

```
        }
    return;
    }
```

3.5. Interfejs użytkownika

Standard VST 2.4 pozwala twórcy wtyczki na niezdefiniowanie interfejsu użytkownika. W takim przypadku interfejs zostanie utworzony przez program kontrolujący. Na rysunku 2 przedstawiony jest przykładowy interfejs użytkownika wtyczki wywołanej przy użyciu programu Cubase 4 LE.

Rysunek 3.3: GUI w programie Cubase 4 LE

Użytkownik ma do dyspozycji dwa parametry:

– **Margin** - próg szumu [dBFs] - określa poniżej jakiego poziomu sygnał będzie klasyfikowany jako szum

– **Gain** - wzmocnienie szumu [dBFs] - określa z jakim tłumieniem sygnał sklasyfikowany jako szum będzie przekazywany do wyjścia

4. Dokumentacja

4.1. Klasa Denoise

```
#include "fftw3.h"
#include "denoiseProgramBank.h
```

4.1.1. Atrybuty prywatne

double noiseMargin

Poziom dźwięku powyżej, którego ramki będą klasyfikowane jako sygnał mowy.

---

double gain

Współczynnik wzmocnienia dla ramek zakwalifikowanych jako szumowe.

---

double *beta

Wskaźnik do tablicy współczynników progu widma.

---

double **alpha

Wskaźnik do tablicy współczynników odejmowania aktualizowana po każdej przetworzonej ramce mowy.

---

double *delta

Wskaźnik do tablicy współczynników wzmocnienia pasm widma szumu.

---

double *previousPower;

Wskaźnik do tablicy poziomów mocy poprzedzających ramek.

---

double *snr

Wskaźnik do tablicy stosunków poziomu sygnału du szum.

---

double `*window`

Wskaźnik do tablicy zawierającej wejściowe okno czasowe.

---

double `*synthesisWindow`

Wskaźnik do tablicy zawierającej wyjściowe okno czasowe.

---

double `*speechPowerSpectrum`

Wskaźnik do tablicy zawierającej widmo mocy sygnału mowy.

---

double `*speechPhaseSpectrum`

Wskaźnik do tablicy zawierającej widmo fazowe sygnału mowy.

---

double `**noisePowerSpectrum`

Wskaźnik do tablicy zawierającej widma mocy sygnałów szumowych.

---

double `**inputLeftovers`

Wskaźnik do tablicy zawierającej wejściowe próbki pozostałe do przetworzenia w kolejnym cyklu.

---

double `**outputLeftovers`

Wskaźnik do tablicy zawierającej wyjściowe próbki pozostałe do przetworzenia w kolejnym cyklu.

---

double `***circularBuffer`

Wskaźnik do tablicy będącej buforem cyklicznym przechowującym ramki szumu.

---

double `*inputBuffer`

Wskaźnik do tablicy będącej buforem próbek wejściowych.

---

double `**outputBuffer`

Wskaźnik do tablicy będącej buforem próbek wyjściowych.

---

double `*frame`

Wskaźnik do tablicy będącej ramką sygnału.

---

int numOfMeans

Ilość uśrednień ramek szumu.

---

int frameSize

Długość ramki sygnału.

---

int hopSize

Długość skoku (przesunięcia) ramki.

---

int leftoversSize

Ilość próbek pozostawianych do przetworzenia w następnym cyklu.

---

int numOfBands

Ilość pasm częstotliwościowych.

---

int *circularBufferIndex

Wskaźnik do tablicy indeksów bufora cyklicznego.

---

int *sampleIndex

Wskaźnik do tablicy indeksów bufora wejściowego.

---

int *bands

Wskaźnik do tablicy wartości granic pasm częstotliwościowych.

---

bool *noisePSisSet

Wskaźnik do tablicy określającej aktualność obliczonych widm mocy szumu.

---

fftw_complex *fftIn

Wskaźnik do tablicy wartości wejściowych szybkiej transformacji Fouriera.

---

fftw_complex *fftOut

Wskaźnik do tablicy wartości wyjściowych szybkiej transformacji Fouriera.

`fftw_complex *ifftIn`

Wskaźnik do tablicy wartości wejściowych odwrotnej transformacji Fouriera.

`fftw_complex *ifftOut`

Wskaźnik do tablicy wartości wyjściowych odwrotnej transformacji Fouriera.

`fftw_plan fftPlan`

Plan obliczania szybkiej transformacji Fouriera w przód.

`fftw_plan ifftPlan`

Plan obliczania szybkiej transformacji Fouriera w tył.

`static const bool` `noise =` **`true`**

Stała identyfikująca sygnał jako szum.

`static const bool` `speech =` **`false`**

Stała identyfikująca sygnał jako mowę.

`static const int` `mono = 1`

Stała identyfikująca system jednokanałowy jako mono.

`static const int` `stereo = 2`

Stała identyfikująca system jednokanałowy jako stereo.

`static const double` `pi`

Stała π.

`VstSpeakerArrangement *plugInput`

Wskaźnik do układu kanałów wejściowych.

`VstSpeakerArrangement *plugOutput`

Wskaźnik do układu kanałów wyjściowych.

`DenoiseProgramBank bank`

Bank programów wtyczki.

---

char `programName`

Nazwa aktualnie wykorzystywanego programu.

4.1.2. Metody publiczne

`Denoise (audioMasterCallback audioMaster)`

Konstruktor klasy.

`~Denoise ()`

Destruktor klasy.

---

virtual void `processReplacing (`**float**`** inputs, **float**`** outputs, VstInt32 sampleFrames)`

Przetwarzanie próbek w rozdzielczości 32bit.

---

virtual void `processDoubleReplacing (`**float**`** inputs, **float**`** outputs, VstInt32 sampleFrames)`

Przetwarzanie próbek w rozdzielczości 64bit.

---

virtual void `setProgram(VstInt32 program)`

Wybranie programu.

---

virtual void `setProgramName(`**char**`* name)`

Ustawienie nazwy aktualnie wybranego programu.

---

virtual void `getProgramName (`**char**`* name)`

Zwrócenie nazwy aktualnie wybranego programu.

---

virtual bool `getProgramNameIndexed (VstInt32 category, VstInt32 index, `**char**`* text)`

Zwrócenie kategorii, nazwy oraz indeksu aktualnie wybranego programu.

```
virtual VstInt32 getChunk (void **data, bool isPreset = false)
```

Zwrócenie bloku danych zapisanych w projekcie.

```
virtual VstInt32 setChunk (void *data, VstInt32 byteSize,
    bool isPreset = false)
```

Zapisanie bloku danych w projekcie.

```
virtual void setParameter (VstInt32 index, float value)
```

Ustawienie wartości wybranego parametru.

```
virtual float getParameter (VstInt32 index)
```

Zawrócenie wartości wybranego parametru.

```
virtual void getParameterLabel (VstInt32 index, char* label)
```

Zawrócenie jednostki wybranego parametru.

```
virtual void getParameterDisplay (VstInt32 index, char* text)
```

Zawrócenie wartości wyświetlanej wybranego parametru.

```
virtual void getParameterName (VstInt32 index, char* text)
```

Zawrócenie nazwy wybranego parametru.

```
virtual bool getEffectName (char* name)
```

Zawrócenie nazwy wtyczki.

```
virtual bool getVendorString (char* text)
```

Zawrócenie ciągu identyfikującego wytwórcę wtyczki.

```
virtual bool getProductString (char* text)
```

Zawrócenie ciągu identyfikującego produkt.

---

virtual VstInt32 getVendorVersion ()

Zawrócenie wartości określającej wersje produktu.

---

virtual VstPlugCategory getPlugCategory ()

Zawrócenie kategorii wtyczki.

---

virtual bool getSpeakerArrangement(VstSpeakerArrangement** pluginInput,
VstSpeakerArrangement** pluginOutput)

Zawrócenie ustawienia kanałów.

---

virtual bool setSpeakerArrangement(VstSpeakerArrangement* pluginInput,
VstSpeakerArrangement* pluginOutput)

Ustawienie specyfikacji kanałów.

---

virtual void resume ()

Wywoływane gdy wtyczka jest uruchamiana bądź przywracana po wywołaniu $suspend()$.

4.1.3. Metody prywatne

void initializePlugin()

Inicjalizacja wszystkich komponentów wtyczki.

---

void createWindow()

Utworzenie okna czasowego.

---

void createSynthesisWindow()

Utworzenie okna czasowego rekonstrukcji.

---

void initializeFFTW()

Inicjalizacja biblioteki FFTW.

---

bool determineSignalType(**int** channelNo)

Klasyfikacja ramki sygnału.

---

void `initializeCircularBuffer()`

Inicjalizacja bufora cyklicznego.

---

void `updateCircularBuffer(int channelNo)`

Zapis próbek do bufora cyklicznego.

---

void `computeSpeechSpectrums()`

Obliczenie widm sygnału mowy.

---

void `computeSpeechPowerSpectrum()`

Obliczenie widma mocy sygnału mowy.

---

void `computePhaseSpectrum()`

Obliczenie widma fazowego sygnału mowy.

---

void `initializeSpectrumBuffers`

Inicjalizacja buforów przechowujących widma.

---

void `computeNoisePowerSpectrum(int channelNo)`

Obliczenie estymacji widma mocy szumu.

---

void `subtractSpectrums(int channelNo)`

Obliczenie różnicy widm mocy.

---

void `reconstructSignal(int channelNo)`

Obliczenie sygnału wyjściowego w dziedzinie czasu.

---

void `computeSNR(int channelNo)`

Obliczenie wartości stosunków sygnału do szumu.

---

void `computeAlpha(int channelNo)`

Obliczenie nowych wartości mnożników α.

---

void initializeIOBuffers()

Inicjalizacja buforów wejścia/wyjścia.

---

void doProcess(**int** channelNo)

Wykonanie algorytmu przetwarzania.

---

void initializeSubtractionCoefficients()

Obliczenie współczynników wykorzystywanych podczas odejmowania nieliniowego.

---

void fillFrame(**int** channelNo)

Wypełnienie ramki odpowiednimi próbkami.

---

void fillOutputBuffer(**int** channelNo)

Wypełnienie bufora wyjściowego odpowiednimi próbkami.

---

void updateLeftovers(**int** channelNo)

Wypełnienie buforów próbek pozostałych do dalszego przetwarzania.

---

void initializeProgram()

Inicjalizacja programu wtyczki domyślnymi wartościami.

4.2. Klasa DenoiseProgram

```
#include "audioeffectx.h"
```

4.2.1. Atrybuty prywatne

char name

Nazwa programu.

---

double noiseMargin

Poziom dźwięku powyżej, którego ramki będą klasyfikowane jako sygnał mowy.

---

double gain

Współczynnik wzmocnienia dla ramek zakwalifikowanych jako szumowe

---

int hopSize

Długość skoku (przesunięcia) ramki.

---

int leftoversSize

Ilość próbek pozostawianych do przetworzenia w następnym cyklu.

---

int numOfBands

Ilość pasm częstotliwościowych.

4.2.2. Metody publiczne

DenoiseProgram()

Konstruktor klasy.

~DenoiseProgram()

Destruktor klasy.

4.3. Klasa DenoiseProgramBank

```
#include "denoiseProgram.h"
```

4.3.1. Atrybuty prywatne

`DenoiseProgram programs`

Tablica zapisanych programów wtyczki.

4.3.2. Metody publiczne

`DenoiseProgramBank()`

Konstruktor klasy.

`~DenoiseProgramBank()`

Destruktor klasy.

`DenoiseProgram getProgram(int programIndex)`

Zwrócenie programu o zadanym indeksie.

`bool setProgram(DenoiseProgram program, int programIndex)`

Ustawienie programu o zadanym indeksie.

---

5. Analiza wyników

Aby ewaluować działanie wykonanego programu wykorzystano grupy sygnałów składające się z czterech elementów: sygnału oryginalnego, sygnału zaszumionego, sygnału zaszumionego przetworzonego przez wtyczkę z ustawionym progiem ciszy na 0[dBFs] (wszystkie ramki klasyfikowane jako szumowe) oraz sygnału odszumionego. Tak przyjęta metoda jest motywowana faktem, że wtyczka podczas działania wprowadza do przetwarzanego sygnału artefakty dodatkowo degradujące jego jakość. Mając do dyspozycji w sygnały zdegradowane przez ten sam proces możemy bezpośrednio określić skuteczność algorytmu z pominięciem niedoskonałości implementacji.

Ocena działania algorytmu została wykonana z wykorzystaniem wersji testowej oprogramowania Sevana AQuA (Audio Quality Analyzer) (`http://www.sevana.fi`) pozwalającego na porównanie podobieństwa dwóch sygnałów mowy, referencyjnego i zdegradowanego, w celu oceny jakości przetwarzanie toru, przez który sygnał został przetworzony. Oprogramowanie AQuA implementuje system algorytm porównania bazujący na właściwościach psychoakustycznych ludzkiego słuchu [11] przystosowany głównie do oceny jakości cyfrowych systemów transmisji głosowej oraz. Do oceny brane są pod uwagę takie parametry jak: zniekształcenia częstotliwościowe uwzględnieniem maskowania, opóźnienie, różnice energii, zniekształcenia czasowe i synchronizacji. Wedle zapewnień producenta oceny generowane przez program są silnie zbieżne z wynikami otrzymywanymi w testach odsłuchowych dla dużej grupy badanych osób.

Program AQuA jest programem wywoływanym z linii komend. Aby dokonać porównania dwóch plików audio należy wywołać go w następującymi parametrami [10]:

```
VQcmd.exe -mode files -src file ORIGINAL_FILE -tstf REFERENCE_FILE
```

gdzie VQcmd jest nazwą odpowiedniej wersji programu. Do oceny wykorzystano pakiet `aqua-wb.exe`, implementujący algorytm porównania dla sygnałów szerokopasmowych (wide-band), ponieważ porównywane pliki audio były zapisane z częstotliwościową próbkowania równą 44100[Hz]. Na rysunku 5.1 przedstawiono przykładowe użycie programu AQuA.

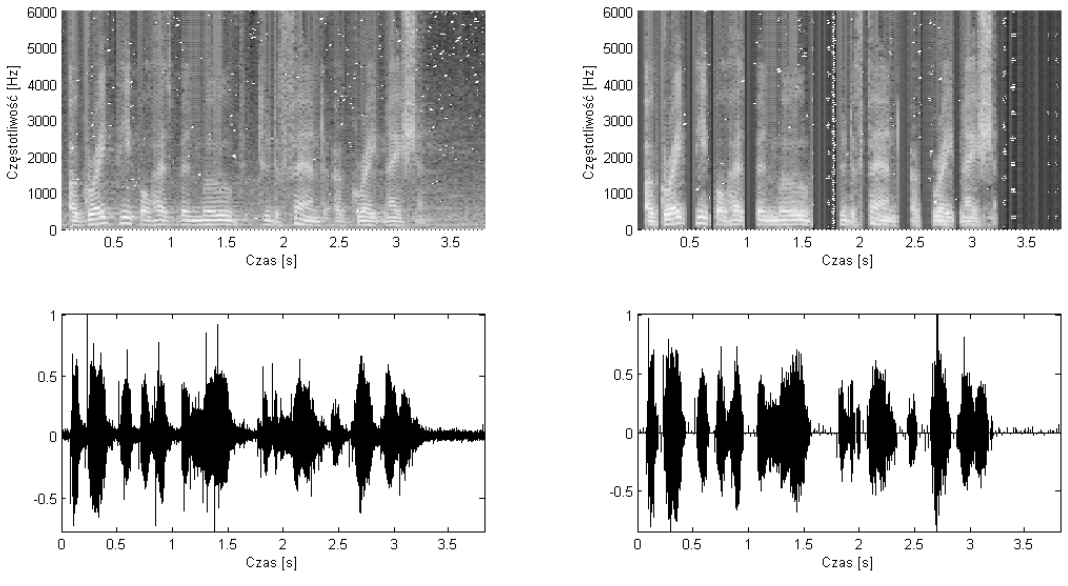
```
G:\VQcmd>aqua-wb.exe -mode files -src file GW.wav -tstf air5p.wav

Sevana Audio Quality Analyzer - AQuA Wideband v.2.1.0.151.
Copyright (c) 2009 by Sevana Oy, Finland. All rights reserved.
-------------------------------------------------------------
File Quality is
        Percent value   40.36
        MOS value       1.57

G:\VQcmd>aqua-wb.exe -mode files -src file GW.wav -tstf air5pp.wav

Sevana Audio Quality Analyzer - AQuA Wideband v.2.1.0.151.
Copyright (c) 2009 by Sevana Oy, Finland. All rights reserved.
-------------------------------------------------------------
File Quality is
        Percent value   68.83
        MOS value       3.54
```

Rysunek 5.1: Przykładowe użycie programu AQuA

System został poddany testom z wykorzystaniem trzech nagrań o różnym stosunku poziomu sygnału do szumu. Wykorzystane szum to nagranie tła akustycznego lotniska Stansted w Londynie. Nagranie szumu zostało zsumowane z sygnałem mowy ze wzmocnieniem gwarantującym określoną wartość SNR.

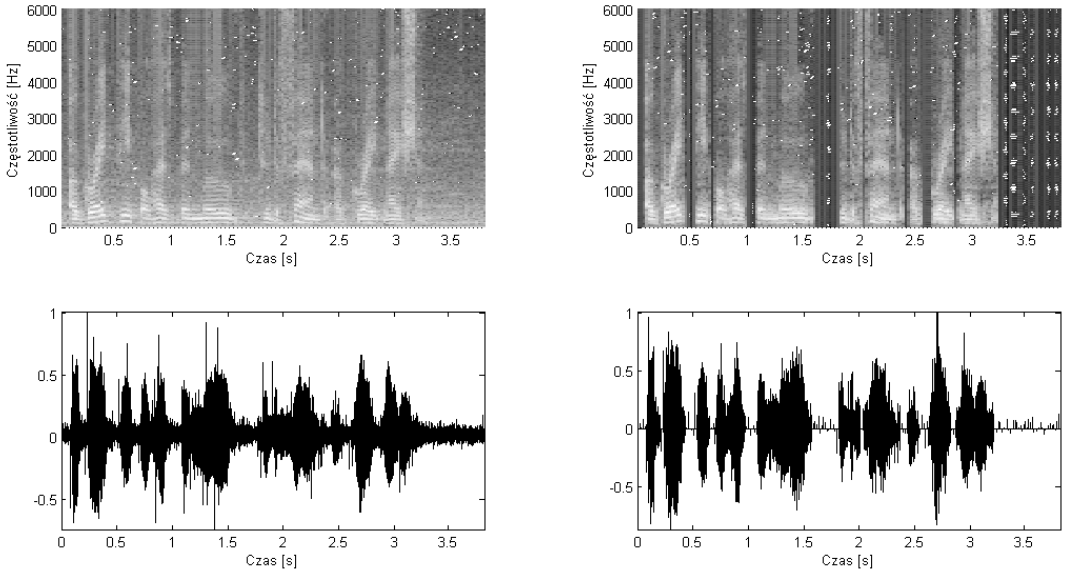
Poniższe ilustracje przedstawiają spektrogramy oraz przebiegi czasowe sygnałów przed i po zastosowaniu algorytmu redukcji szumu.

Sygnał zaszumiony o nominalnym snr = 15:



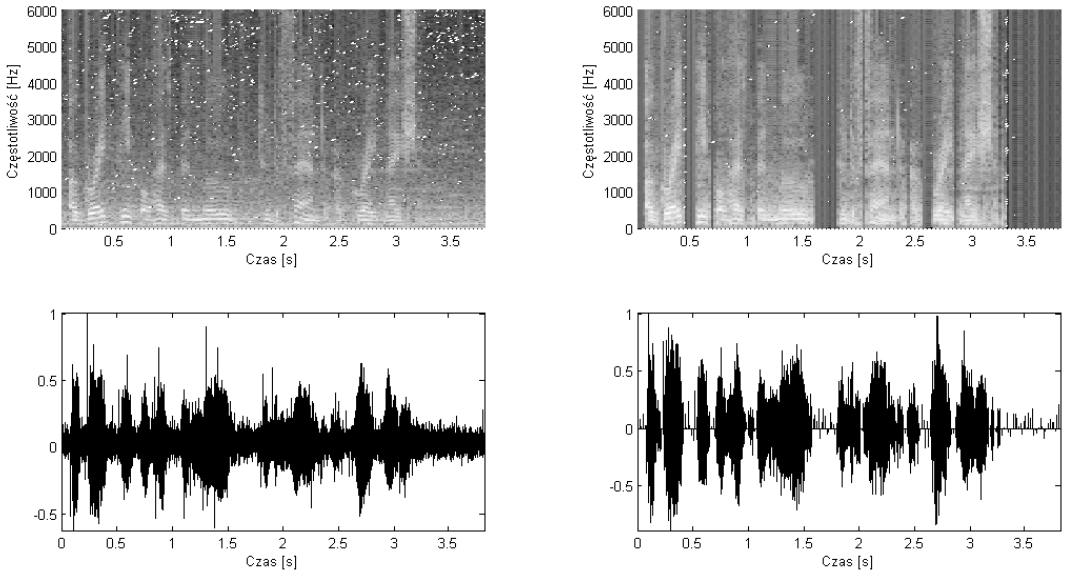
Przed przetworzeniem Po przetworzeniu

Sygnał zaszumiony o nominalnym snr = 10:



Przed przetworzeniem Po przetworzeniu

Sygnał zaszumiony o nominalnym snr = 5:



Przed przetworzeniem Po przetworzeniu

Poniższa tabela przedstawia wyniki pomiarów jakości przetworzonego sygnału:

Nominalny SNR [dB]	Zmiana SNR [dB]	Podobieństwo AQuA [%] sygnał + szum	Podobieństwo AQuA [%] sygnał + szum + artefakty	Podobieństwo AQuA [%] sygnał odszumiony
15	0.54	82.8	49.1	64.2
10	1.3	77.2	49	65
5	2.7	68.2	40.3	68.8

Na podstawie zebranych wyników możemy stwierdzić, że skuteczność zaimplementowanego algorytmu jest zdecydowanie większa gdy różnica pomiędzy poziomem szumu a mowy jest mniejsza (czym mniejsza wartość SNR tym odejmowanie widmowe pozwala na uzyskanie większej poprawy jakości sygnału). Możemy również stwierdzić, że artefakty wprowadzane w czasie przetwarzania maja większy wpływ na jakość mowy niż szum addytywny. W przypadku nagrania o wartości $SNR = 5$ uzyskano jakość odwzorowania sygnału oryginalnego na poziomie jakości sygnału przed wprowadzeniem artefaktów.

6. Podsumowanie

Celem niniejszej pracy było wykonanie systemu redukcji szumu pracującego w czasie rzeczywistym. Dokonano tego z wykorzystaniem technologii VST®, która pozwala na prostą integracje algorytmów przetwarzania dźwięku z programowymi systemami rejestracji i produkcji audio. Poprawność funkcjonowania interfejsu VST® została zweryfikowana z wykorzystaniem programu Cubase 4®. Wykazano również, że zastosowany algorytm pozwala na poprawę jakości sygnału mowy zakłóconego naturalnym szumem tła. Ze względu na niedoskonałą implementację, wtyczka w przedstawionej w niniejszej pracy wersji nie może być jeszcze wykorzystana praktycznie ale na podstawie przeprowadzonych testów można wnioskować, że po usunięciu błędów odpowiedzialnych za wprowadzanie artefaktów do przetwarzanego sygnału program będzie mógł być wykorzystywany do poprawy jakości nagrań mowy.

Zastosowany algorytm ze swojej natury może być skutecznie stosowany tylko gdy szum tła jest wolnozmienny (tylko wtedy zebrane dane w momentach ciszy są dostatecznie zbliżone do rzeczywistej charakterystyki tła w chwilach występowania sygnału mowy) oraz gdy jego poziom jest co najmniej o kilka decybeli niższy od poziomu sygnału mowy. Aby system mógł sprawnie działać również w przypadku gdy poziom szumu jest wyskoki należy zaimplementować jako cześć decyzyjną systemu algorytm VAD (ang. *Voice Activity Detection*) [12] pozwalający rozpoznanie ramek mowy od szumowych nawet dla ujemnych wartości stosunku poziomu sygnału do szumu. Aby dodatkowo zwiększyć skuteczność redukcji szumu niestacjonarnego można uzupełnić system o estymację widma szumu w czasie trwania ramek mowy używając algorytmu *Minimum Tracking* [13].

Bibliografia

[1] http://dea.brunel.ac.uk/cmsp/Home_Esfandiar/Spectral%20Subtraction.htm - *Sepctral Subtraction Basics*.

[2] Anuradha R. Fukane, Shashikant L. Sahare. *Different Approaches of Spectral Subtraction method for Enhancing the Speech Signal in Noisy Environments*, International Journal of Scientific & Engineering Research, Volume 2, Issue 5, May-2011.

[3] Sunil D. Kamath and Philipos C. Loizou. *A multi-band spectral subtraction method for enhancing speech corrupted by colored noise*, Proceedings of ICASSP-2002, Orlando, FL, May 2002.

[4] Ch.V.Rama Rao, M.B.Rama Murthy and K.Srinivasa Rao. *Noise reduction using mel-scale spectral subtraction with perceptually defined subtraction parameters - a new scheme*, Signal & Image Processing : An International Journal(SIPIJ) Vol.2, No.1, March 2011.

[5] http://www.gnu.org/copyleft/gpl.html - *GNU General Public License*

[6] http://www.fftw.org/fftw3_doc/ - *FFTW User Manual*.

[7] *Steinberg VST SDK 2.4 Documentation - revision 2*.

[8] Richard G. Lyons. *Wprowadzenie do cyfrowego przetwarzania sygnałów*. Wydawnictwo Komunikacji i Łączności 2006.

[9] Steven W. Smith. *Cyfrowe przetwarzanie sygnałów: Praktyczny poradnik dla inżynierów i naukowców*. Wydawnictwo BTC 2007.

[10] *Sevana AQuA - Audio Quality Analyzer 2.1 Manual*.

[11] http://www.sevana.fi/voice_quality_testing_measurement_analysis.php - *AQuA - competitve alternative for PESQ (P.862)*

[12] Michel Grimm and Kristian Kroschel. *Robust Speech Recognition and Understanding*, I-TECH Education and Publishing.

[13] E. Hansler, G.Schmidt. *Topics in Acoustic Echo and Noise Control*, Springer-Verlag Berlin Heidelberg 2006.