TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Redukce šumu v řeči pomocí pole mikrofonů a neuronových sítí

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Petr Bartoš**

Vedoucí práce: doc. Ing. Zbyněk Koldovský, Ph.D.

Liberec 2018

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Noise reduction in speech signals using an microphone array and neural networks

Master thesis

Study programme: N2612 – Electrotechnology and informatics

Study branch: 1802T007 – Information technology

Author: **Bc. Petr Bartoš**

Supervisor: doc. Ing. Zbyněk Koldovský, Ph.D.

Liberec 2018

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Petr Bartoš**
Osobní číslo:	**M15000157**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Redukce šumu v řeči pomocí pole mikrofonů a neuronových sítí**
Zadávající katedra:	**Ústav informačních technologií a elektroniky**

Zásady pro vypracování:

1. Seznamte se s evaluační kampaní CHiME-4 a CHiME-5 a metodami navrženými pro řešení definovaných úloh. V diplomové práci uveďte přehled současného poznání.
2. Vytvořte sadu vlastních trénovacích záznamů pro trénování hlubokých neuronových sítí pro detekci řečové aktivity tak, aby byla detekce funkční v reálných podmínkách vybrané místnosti.
3. Implementujte metody popsané v publikaci [1] pro zpracování záznamů v reálném čase, s využití detektoru z bodu 2. Použijte Audio System Toolbox pro Matlab nebo metodu implementujte v C++.
4. Optimalizujte systém vzhledem k výpočetní náročnosti a vzhledem k celkovému zpoždění mezi vstupem a výstupem.

Rozsah grafických prací:	**Dle potřeby dokumentace**
Rozsah pracovní zprávy:	**cca 40-50 stran**
Forma zpracování diplomové práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] **H. L. Van Trees, Optimum Array Processing: Part IV of Detection, Estimation, and Modulation Theory, John Wiley & Sons, Inc.,2002.**

[2] **Z. Koldovský, J. Málek, M. Boháč, J. Janský, ?CHiME4: Multichannel Enhancement Using Beamforming Driven by DNN-based Voice Activity Detection,? Proc. of the 4th Intl. Workshop on Speech Processing in Everyday Environments (CHiME 2016), San Francisco,**

[3] **J. Málek, Z. Koldovský, M. Boháč, ?Block-Online Multichannel Speech Enhancement Using DNN-Based Voice Activity Detection,? submitted, 2017.**

Vedoucí diplomové práce:	**doc. Ing. Zbyněk Koldovský, Ph.D.**
	Ústav informačních technologií a elektroniky
Konzultant diplomové práce:	**Ing. Jiří Málek, Ph.D.**
	Ústav informačních technologií a elektroniky

Datum zadání diplomové práce:	**19. října 2017**
Termín odevzdání diplomové práce:	**14. května 2018**

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

L.S.

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 19. října 2017

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14.5.2018

Podpis:

Abstrakt

Tato práce se zabývá způsoby redukce šumu pomocí pole mikrofonů v reálném čase. Jsou zde ukázány způsoby zpracování vícekanálových záznamů z evaluační kampaně CHiME-4 a CHiME-5. Seznámíme se s problematikou odhadu šumu pomocí pole mikrofonů. Je zde ukázána důležitost potřeby detektoru aktivity řeči VAD. Pro realizaci redukce šumu v reálném čase jsou zde navrženy dva systémy. Jeden využívá metody beamformingu za pomocí VAD a druhý pracuje na principu slepé separace signálu, přesněji pak extrakci nezávislého vektoru. Oba Systémy jsou realizovány pomocí programovacího jazyka C++ a knihovny PortAudio. První systém se v praxi ověřil jako velmi efektivní, zatím co druhý systém fungoval pouze omezeně. U obou systémů proběhla optimalizace s důrazem na minimální možnou hodnotu zpoždění a vytížení procesoru.

Klíčová slova: Redukce šumu, CHiME-4, VAD, real-time aplikace

Abstract

This thesis focuses on noise reduction in real-time using an microphone array. A number of techniques is shown being used in the processing of multichannel recordings from the CHiME-4 and CHiME-5 challenge. The solution to the issue of noise estimation using an microphone array is presented. Also the importance of a good VAD is shown. For the realization of noise reduction in real-time we designed two systems. One is based on beamforming aided by VAD, the other one is based on blind signal separation, more precisely on independent vector extraction. Both systems are implemented using the programming language C++ and PortAudio library. The first system is shown as highly effective, whereas the other one has limited functionality. Both were optimized with the emphasis on minimal latency and processor usage.

Keyword: Noise reduction, CHiME-4, VAD, real-time application

Poděkování

Chtěl bych poděkovat doc. Ing. Zbyňku Koldovskému PhD. za vedení mé diplomové práce, cenné rady, odborný dohled, vstřícnost při konzultacích a zapůjčení profesionálních mikrofonů.

Obsah

Seznam zkratek 13

1 Úvod **14**

2 Úvod do problematiky **16**

2.1 Šum 16

2.2 Model signálu 17

2.3 Pojmy spjaté s redukcí šumu 18

2.3.1 Vzájemná korelace (Cross correlation) 18

2.3.2 Enhancer 18

2.4 Evaluační kampaň CHiME-4 19

2.4.1 Konfigurace mikrofonů 19

2.4.2 Chyby na mikrofonech 20

2.4.3 Postup při nahrávání 21

2.4.4 Popis dat 21

2.4.5 Popis datových sad 21

2.4.6 BeamformIt 22

2.4.7 Odhadnutí referenčního kanálu 23

2.4.8 Automatická adaptivní eliminace kanálů 24

2.5 CHiME-5 24

2.5.1 Postup při nahrávání 24

2.5.2 Konfigurace mikrofonů 25

2.5.3 Popis dat 25

2.6 STFT 26

2.7 Beamforming . . . 27
2.7.1 Delay and sum Beamformer (DSB) . . . 27
2.7.2 MVDR beamformer . . . 27
2.7.3 Filter-and-sum beamformer . . . 28
2.8 Neuronové sítě . . . 28
2.9 Voice Activity Detector (VAD) . . . 30
2.10 sVAD . . . 31
2.11 dVAD . . . 31

3 Návrh řešení **32**
3.1 Koncept . . . 32
3.2 Metody . . . 33
3.2.1 Metoda 1 . . . 33
3.2.2 Výpočet relativní přenosové funkce . . . 34
3.2.3 Metoda 2 . . . 36

4 Realizace **40**
4.1 Vývojové prostředí . . . 40
4.2 API a ASIO4ALL . . . 40
4.3 Knihovny . . . 41
4.3.1 PortAudio . . . 41
4.3.2 Armadillo . . . 41
4.3.3 SigPack . . . 42
4.3.4 FFTW3 . . . 42
4.4 Program . . . 42
4.4.1 Běh aplikace . . . 42
4.4.2 Metoda paCallback pro metodu 1 . . . 45
4.4.3 Metoda getAsyncVADandRTF . . . 50
4.4.4 Metoda používající výpočet pomocí IVE . . . 53
4.4.5 Vzhled aplikace a ovládání . . . 54

5 Testování a výsledky **56**

5.1 Metoda používající beamformer 58

5.1.1 Latence při zpracování 58

5.1.2 Efektivita redukce šumu 58

5.1.3 Výpočetní náročnost 59

5.2 Metoda používající IVE 60

5.2.1 Latence při zpracování 60

5.2.2 Efektivita redukce šumu 60

5.2.3 Výpočetní náročnost 61

6 Závěr **62**

Literatura **64**

Přílohy **66**

A Obsah přiloženého CD **67**

Seznam obrázků

2.1 Ukázka nahrávacího zařízení s popisky pozic mikrofonů [8] 20
2.2 Ukázka překrytí okének [16] 26
2.3 Ukázka neuronové sítě [18] 29
2.4 Ukázka funkce VAD [5] 30

3.1 Schéma popisující zpracování dat 32

4.1 Ukázka funkce Pa_GetDefaultInputDevice() 43
4.2 Ukázka nastavení vstupních parametrů 43
4.3 Ukázka nastavení výstupních parametrů 44
4.4 Ukázka funkce `Pa_OpenStream()` 44
4.5 Diagram popisující průběh funkce paCallback 46
4.6 Ukázka rozdělení vstupních dat na levý a pravý kanál 47
4.7 Ukázka decimace v kódu 47
4.8 Ukázka rotace bufferu timeDataIn 48
4.9 Ukázka výpočtu STFT v kódu 48
4.10 Ukázka interpolace v kódu 49
4.11 Ukázka přiřazení dat na výstup v kódu 49
4.12 Spuštění funkce getAsyncVADandRTF v jiném vlákně 51
4.13 Ukázka výpočtu automatického zesílení uvnitř VADu 52
4.14 Ukázka výpočtu hodnot VADu 52
4.15 Ukázka aplikace po spuštění 54
4.16 Ukázka možností volby výstupu 55

5.1 Nahrání dat v programu Audacity 57
5.2 Porovnání 2 signálů před a po redukci šumu 57
5.3 Ukázka využití procesoru naší aplikací pomocí Profileru ve Visual Studiu 60

Seznam zkratek

FFT Fast Fourier Transformation (rychlá Fourierova transformace)

STFT Krátkodobá Fourierova transformace

VAD Voice activity detector (Detektor aktivity řeči)

ICA Independent component analysis (Nezávislá analýza komponent)

RTF Relative transfer function (relativní přenosová funkce)

Kapitola 1

Úvod

Problematika zlepšování řeči vzdáleného mluvčího pomocí vícekanálových záznamů je jednou ze současných témat v oblasti zpracování zvuku. Tento problém se vyskytuje na mnoha místech běžného života, kde chceme nahrávat zvuk a v okolí se nachází šum. Jedním z mnoha takových případů je například telefonování. V dnešní době má většina lidí u sebe chytrý telefon a tyto telefony už obsahují mikrofonní pole většinou skládající se ze 2 nebo 3 mikrofonů. Dalším příkladem může být nahrávání různých druhů konferencí, kde se chceme zaměřit na hlavního řečníka a ne na okolní hovory účastníků konference.

Hlavním důvodem výběru tématu této práce byla záliba v práci se zvukem. Zároveň mě vždy zajímaly způsoby jak zlepšit kvalitu řeči v zarušeném prostředí.

Cílem práce je seznámit se s problematikou redukce šumu pomocí vícekanálových záznamů a vytvořit aplikaci, která danou problematiku řeší v reálném čase. Seznámíme se zde s postupem řešení tohoto problému a s jednotlivými kroky tohoto řešení. V první řadě se seznámíme s databází 6ti kanálových nahrávek z evaluační kampaně CHiME-4.Také se seznámíme s evaluační kampaní CHiME-5. V práci se také seznámíme s metodou BeamformIt [2], která byla vybrána jako referenční metoda organizátory evaluační kampaně CHiME-4. Tato metoda představuje v současné době jednu z nejvýkonnějších realizací delay-and-sum beamformeru pro řešení problému zlepšování řeči vzdáleného řečníka. Ukážeme si způsoby jakými se řeší

problém odhadu šumu pomocí pole mikrofonů. Zjistíme jakou roli hraje detektor aktivity řeči a jak se řeší pomocí neuronových sítí.

Po dohodě s vedoucím práce jsme se rozhodli vynechat druhý bod zadání, tj. natrénovat vlastní detektor řeči. Místo toho se v práci pokusíme vyřešit problém redukce šumu pomocí metody extrakce nezávislých komponent. Tato metoda spadá pod slepou separaci signálů, tím pádem je nezávislá na trénovacích datech a mohla by fungovat lépe v obecném prostředí. Její nevýhodou, ale může být, že správně nezaměří cílový signál, popřípadě u něj nesetrvá.

Dále se seznámíme se systémem pro vylepšení řeči vypracovaným na ústav ITE [6]. Pro redukci šumu se implementovaly 2 druhy systému, jeden postavený na beamformeru a VAD, a druhý postavený na slepé separaci řeči. Implementace těchto systémů pro redukci šumu v reálném čase byla realizována pomocí programovacího jazyka C++ a knihovny PortAudio. Výsledné aplikace otestujeme a budeme se snažit co nejvíce snížit zpoždění mezi vstupem a výstupem a jejich výpočetní náročnost.

Kapitola 2

Úvod do problematiky

2.1 Šum

Signál se skládá vždy z užitečné složky a šumu. Užitečnou složkou ve zvukových signálech může být například řeč, pokud je naším záměrem nahrávat řečníka, nebo hudba pokud chceme nahrávat vystoupení kapely. Šumem může být cokoliv, např. okolní zvuky ve formě projíždějících aut na silnici, rozhovorů ostatních lidí v rušném místě jako je kavárna apod. Výskyt šumu v signálu se dá popsat následující rovnicí

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

kde $y(t)$ představuje pozorovaný signál v závislosti na čase, $x(t)$ je užitečná složka signálu, v našem případě řeč a $d(t)$ reprezentuje šum v nahrávce.

Na první pohled by se mohlo zdát, že odstranění šumu je jednoduchou záležitostí, ale opak je pravdou. Zejména v praktickém využití, kdy není známá užitečná složka signálu. Tento problém se řeší za pomocí více mikrofonů k odhadu šumu. Základním předpokladem pro toto řešení je fakt, že na obou mikrofonech (pokud máme pouze dva) se nachází užitečný signál, ale na jednom je modifikovaný pomocí relativní přenosové funkce h. Tato funkce představuje změnu cílového signál vlivem zpoždění

a odrazů v místnosti. Tento předpoklad můžeme vyjádřit v diskrétním čase pomocí následující rovnice

$$\begin{aligned} y_L[n] &= x[n] + d_L[n] \\ y_R[n] &= h * x[n] + d_R[n], \end{aligned} \tag{2.2}$$

kde $*$ představuje operaci konvoluce. V závislosti na pozici zdroje signálu získáme na levém a pravém mikrofonu signály $y_L[n]$ a $y_R[n]$ a naší snahou je odhadnout relativní impulzní odezvu h. Toto je hlavním problémem redukce šumu, jelikož správný odhad relativní impulzní odezvy znamená, možnost odečíst signály od sebe, a tím získat odhad šumu na těchto mikrofonech. Toto se dá popsat následující rovnicí.

$$\begin{aligned} d[n] &= \bar{h} * y_L[n] - y_R[n] = h * x[n] + h * d_L[n] - h * x[n] - d_R[n] \\ d[n] &= h * d_L[n] - d_R[n] \end{aligned} \tag{2.3}$$

Jak je vidět d[n] není úplně přesný odhad šumu, jelikož šum na levém mikrofonu je modifikovaný relativní přenosovu funkcí a bude mít jinou hodnotu hustoty výkonového spektra. Nicméně je toto dobrý odhad pro jeho redukci.

2.2 Model signálu

Nahrávku zašumělého směrového signálu, který je zachycen pomocí m mikrofonů se dá popsat pomocí následující rovnice

$$\boldsymbol{x}(k,l) = \boldsymbol{g}(k,l)s(k,l) + \boldsymbol{y}(k,l), \tag{2.4}$$

kde $\boldsymbol{x(k,l)}$ je vektor o rozměrech [m,1] reprezentující signály přijímané na mikrofonech, $s(k,l)$ je cílová řeč a $\boldsymbol{y(k,l)}$ představuje ostatní šum ať už řeč od jiného zdroje či okolní šum. Signál je popsán v krátkodobé frekveční doméně a proměnné k a l, reprezentují frekvenční index a index časového rámce.

Vektor $\boldsymbol{g(k,l)}$ určuje pozici cílového řečníka. Jeho prvky obsahují relativní přechodové funkce vzhledem k referenčnímu mikrofonu [7]. To znamená, že jednotlivé

prvky tohoto vektoru odpovídají přenosovým funkcím mikrofonu s odpovídajícím indexem $(1, ..., m)$ oproti zvolenému referenčnímu mikrofonu. Vzhledem k možnosti pohybu řečníka při promluvě je tento vektor proměnný v čase. Vychází se zde z předpokladu, že tyto změny nebudou náhlé a proto se hodnota $g(k, l)$ považuje za konstatní v několika málo po sobě jdoucích rámců.

2.3 Pojmy spjaté s redukcí šumu

V této sekci definujeme některé pojmy, které se budou používat dále v práci.

2.3.1 Vzájemná korelace (Cross correlation)

Vzájemná korelace dvou signálu je jeden ze způsobů porovnání dvou signálů. Nejčastěji se používá k zjištění posunu jednoho kanálu vůči druhému. Zjištěním posunutí signálu na jednom mikrofonu oproti signálu na druhém mikrofonu získáme informaci o zpoždění mezi těmito mikrofony. Tuto informaci můžeme použít při odhadnutí šumu. Popsaná je pomocí následující rovnice

$$R_{xy}(m) = E\{x_{n+m}y_n^*\} = E\{x_n y_{n-m}^*\} \tag{2.5}$$

kde $-\inf < n < \inf$, * značí komplexně sdružené číslo a E je operátor střední hodnoty. V praxi se používá odhad podle vzorce 2.6, jelikož je k dispozici pouze omezený počet vzorků.

$$\widehat{R}_{xy}(m) = \begin{cases} \sum_{n=0}^{N-m-1} x_{n+m}y_n^*, & m \geq 0 \\ \widehat{R}_{xy}^*(-m), & m < 0 \end{cases} \tag{2.6}$$

2.3.2 Enhancer

Je název pro systém zlepšující (angl. Enhancing) signál, podle různých kritérií, v oblasti zpracování zvuku je většinou tento systém používán pro snížení šumu v nahrávkách. Výstup enhanceru se dá dále použít např. pro rozpoznání řeči jako tomu je u evaluační kampaně CHiME-4.

2.4 Evaluační kampaň CHiME-4

Evaluační kampaň CHiME-4 je zaměřena na úlohu automatické rozpoznání řeči (ASR – angl. Automatic speaker recognition) a snížení šumu v nahrávkách. V této kampani je představena úloha, kde se pomocí více-mikrofonního pole na tabletu nahrávají nahrávky v každodenních, hlučných prostředích.

Tato soutěž se skládá z více částí, které se dají rozdělit podle počtu kanálů použitých při nahrávání následovně

- 1 kanálová
- 2 kanálová
- 6 kanálová

Nahrávky byly pořízeny ve 4 různých prostředích reprezentující běžná místa výskytu při pořizování audio záznamu, např. telefonování. Těmito prostředími jsou

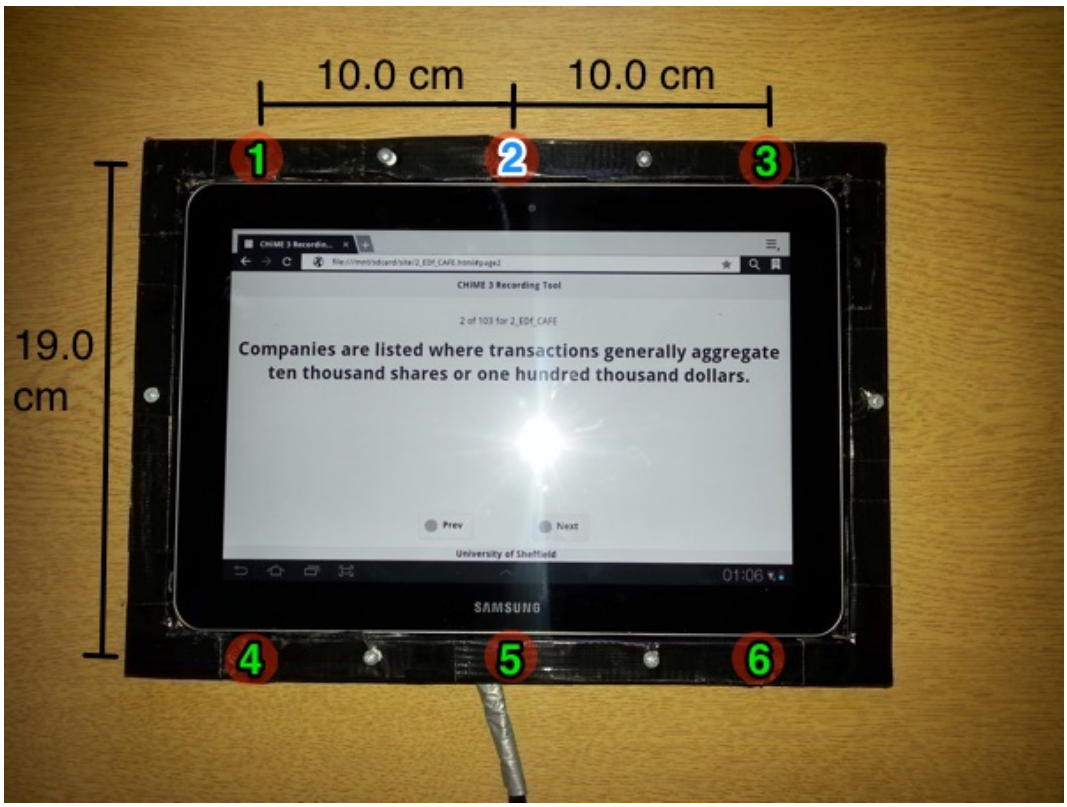
- kavárna
- rušná ulice
- městská hromadná doprava (autobus)
- pěší zóna

Nahrávání probíhalo následovně. Na tabletu se zobrazovaly věty k přečtení. Tyto věty byly přečteny různými řečníky a jejich řeč byla posléze zachycena, prostřednictvím 6 mikrofonů přidělaných k rámu tabletu, a nahrána v rozlišení 24 bitů na 48 kHz pomocí digitálního nahrávače TASCAM DR-680. Poté byly audio nahrávky podvzorkovány na rozlišení 16 bitů a 16 kHz.

2.4.1 Konfigurace mikrofonů

Na obrázku 2.1 je vyzobrazeno nahrávací zařízení s informacemi o poloze mikrofonů. Mikrofony jsou označeny čísly 1 až 6 korespondující s jednotlivými kanály v databázi.

Mikrofony popsány zelenou barvou jsou orientovány směrem k řečníkovi. Mikrofon číslo 2, popsán modrou barvou je orientován směrem dozadu na rámu tabletu.



Obrázek 2.1: Ukázka nahrávacího zařízení s popisky pozic mikrofonů [8]

Pro nahrávání všech 6 mikrofonů byl použit jeden TASCAM nahrávač, a proto jsou synchronizovány po vzorcích.

2.4.2 Chyby na mikrofonech

Stejně jako na komerčních zařízeních docházelo občas k chybám při nahrávání některých mikrofonů. Tyto chyby vznikaly zejména z důvodu hardwarových problémů nebo kvůli zakrytí mikrofonu rukou či oblečením řečníka. Tyto chyby se vyskytují v 12% reálných dat a jsou koncentrovány na kanálech 4 a 5, zejména pak v prostředích BUS (městská hromadná doprava), STR (rušná ulice) a PED (pěší zóna). Chyby v nahrávkách se nevyskytují pro úlohy zabývající se pouze 1 nebo 2 kanály, ale pro úlohu se 6ti kanály jsou tyto chyby už problémem.

Automatická detekce chyb na mikrofonech je jedna ze součástí soutěže ChiME-4. Účastníci mohli použít referenční enhancer (BeamformIt), který implicitně vyřazuje kanály, kde nalezl chybu. Tímto problémem se v této práci nezabýváme, jelikož předpokládáme, že se na mikrofonech chyby vyskytovat nebudou. Zejména z důvodu použití kvalitních nahrávacích zařízení.

2.4.3 Postup při nahrávání

Nahrávky byly nahrány 12 řečníky z USA (6 mužů a 6 žen). Každý řečník nejdříve pořídil nahrávky ve zvukově izolované místnosti a posléze ve 4 hlučných prostředích. V každém prostředí bylo nahráno okolo sta nahrávek. Řečníkům bylo doporučeno změnit polohu tabletu, např. tablet držen v ruce, položen v klíně, položen na stole. Polohu měli změnit po každých 10 nahrávkách.

Následné nahrávky byly rozděleny do trénovacích, vývojových a evaluačních sad. Každá sada obsahovala různé instance šumu ze stejných prostředí, tj. všechny datové sady obsahovaly nahrávky z prostředí kavárna, ale v každé sadě byla jiná specifická kavárna.[1]

2.4.4 Popis dat

Data se dělí na reálná a simulovaná. Reálná data představují nahrávky řeči pořízené v hlučném prostředí. Simulovaná data vznikla kombinací nahrávek hlučného prostředí s nahrávkami řeči nahraných v izolovaném prostředí, tj. bez okolního hluku a bez chyb na mikrofonech.

2.4.5 Popis datových sad

Data jsou dále rozdělena do několika datových sad. První je trénovací sada. Tato sada obsahuje 1600 reálných a 7138 simulovaných nahrávek, celkem 8738 nahrávek. Jak název sady napovídá tato sada slouží k natrénování neuronových sítí pro rozpoznání aktivity řeči (VAD). Druhá sada se nazývá vývojová a slouží k upřesnění parametrů VAD. Obsahuje 410 reálných nahrávek a 410 simulovaných nahrávek pro

každé ze čtyř prostředí. Poslední sada je testovací a slouží k zjistění efektivity natrénovaného VAD. Obsahuje 330 reálných a 330 simulovaných nahrávek pro každé ze čtyř prostředí.

2.4.6 BeamformIt

BeamformIt je systém dodán zadavateli soutěže, který slouží jako baseline pro vylepšení nahrávek, tj. pro snížení šumu a přípravu nahrávek pro rozpoznávač[2].

Tento systém je vytvořen na bázi weighted-delay & sum beamforming. Výstupní signál je popsán jako vážený součet různých kanálů následovně:

$$y\,[n] = \sum_{m=1}^{M} W_m[n] X_m[n - TDOA^{m,ref}[n]] \tag{2.7}$$

kde $W_m[n]$ představuje relativní váhu mikrofonu $\boldsymbol{m}$ (z $\boldsymbol{M}$ mikrofonů) pro okamžik $\boldsymbol{n}$, kde součet všech váh je 1. $x_m[n]$ je signál pro kanál $\boldsymbol{m}$ a $TDOA^{m,ref}[n]$ je relativní zpoždění mezi jednotlivými kanály a kanálem referenčním, abychom získali navzájem časově synchronizované signály a okamžik n. V praxi se $TDOA(m, ref)[n]$ odhaduje pomocí vzájemné korelace (cross korelace) po několika rámcích. V této implementaci má tento rámec délku 250 ms a je vypočítán pomocí GCC-PHAT (Generalized Cross Correlation with Phase Transform). Výpočet pro GCC-PHAT vypadá následovně:

$$\widehat{R}^{i,ref}_{PHAT}(d) = F^{-1}\left(\frac{X_i(f)[X_{ref}(f)]^*}{|X_i(f)[X_{ref}(f)]^*|}\right) \tag{2.8}$$

kde $X_i(f)$ a $X_{ref}(f)$ jsou Fourierovy transformace původních signálů x_{ref} a x_i, F^{-1} značí inverzní Fourierovu transformaci, $[]*$ značí komplexní sdružení a $||$ značí modulus. Výsledná hodnota $\widehat{R}^{i,ref}_{PHAT}(d)$ je korelační funkce mezi signály i a ref. Tato hodnota může nabývat hodnot v rozmezí 0 až 1, při normalizaci amplitud ve frekvenční doméně.

2.4.7 Odhadnutí referenčního kanálu

Referenční kanál by měl mít pozici co nejvíce ve středu všech mikrofonů a zároveň nejlepší kvalitu signálu.

K odhadnutí referenčního kanálu jsou použity vzájemné korelace, které jsou následně zprůměrovány přes více bloků. Tato hodnota je posléze časově průměrovaná přes 1 vteřinové segmenty následovně:

$$\overline{xcorr_i} = \frac{1}{K(M-1)} \sum_{k=1}^{K} \sum_{j=1, j\neq i}^{M} xcorr[i, j; k] \tag{2.9}$$

Kde M je celkový počet mikrofonů a K určuje celkový počet bloků k průměrování. $xcorr[i, j; k]$ je standardní vzájemná korelace mezi kanály$i$ a j pro blok k.

Time difference of arrival (TDOA) pro dva mikrofony (i a ref) je odhadnut následovně:

$$TDOA_1^i = \underset{d}{argmax}(\widehat{R}_{PHAT}^{i,ref}(d)) \tag{2.10}$$

Z důvodu různých pozic cílového řečníka vzhledem k poli mikrofonů je potřeba vyřešit problém odlišných úrovní signálu. Toto se řeší pomocí neustálého váhování jednotlivých kanálů. Výpočet vah jednotlivých kanálů m pro segment c $W_m[c]$ je vypočítán následovně:

$$W_m[c] = \begin{cases} \frac{1}{M} & c = 0 \\ (1-\alpha) \cdot W_m[c-1] + \alpha \cdot \overline{xcorr_m}[c] & ostatní \end{cases} \tag{2.11}$$

Kde α je adaptivní koeficient, který byl nastaven empiricky na $\alpha = 0.05$, c je segment, který je právě zpracováván a $\overline{xcorr_m}[c]$ je průměr vzájemných korelací kanálu m s ostatními kanály, které byly zpožděny pomoci hodnoty $TDOA^m[c]$ pro daný kanál. Takto se zaručí, že kanály, které mají nejlepší hodnotu $\overline{xcorr_m}[c]$, tj. signál na nich má nejlepší kvalitu budou zesíleny, zatímco kanály s nižšími hodnotami budou potlačeny.

2.4.8 Automatická adaptivní eliminace kanálů

Automatická adaptivní eliminace kanálů se používá v případech, kdy je signál kanálu tak porušen, že jeho zařazení do celkového součtu by zhoršilo celkovou kvalitu. Pro rozhodnutí o daném segmentu daného kanálu je použit $\overline{xcorr_m}[c]$ k vyhodnocení jeho kvality. Pokud $\overline{xcorr_m}[c] < 1/4M$, poté $W_m[c] = 0$. Po odstranění kanálů, které splňují tuto podmínku, jsou váhy přepočítány, aby dávaly součet 1.

2.5 CHiME-5

Evaluační kampaň CHiME-5[9] se zaměřuje na problém rozeznání vzdálené konverzační řeči v každodenním prostředí. Nahrávky v této kampani se pořizovali při oslavách s večeří v domech hostů. Nahrávání probíhalo pomocí několika mikrofonních polí. Jelikož se jedná o běžné prostředí domu dochází zde k zachycení okolního šumu od elektrických spotřebičů, ventilace, pohybu lidí apod.

2.5.1 Postup při nahrávání

Nahrávky se skládají z 20 různých oslav s večeří. Každá oslava měla čtyři účastníky, dva hostitele a dva hosty. Všichni účastníci se znali a bylo jim řečeno ať se chovají co nejvíce přirozeně. Oslavám se nechával volný průběh s dvěma podmínkami. První podmínkou bylo, že každá oslava by měla trvat alespoň 2 hodiny a měla by se skládat ze 3 částí. Každá tato část odpovídala jinému pokoji domu. Tyto pokoje byly následující:

- Kuchyň – příprava jídla
- Jídelna – konzumace večeře
- obývací pokoj – zábava po večeři

Účastníci oslavy se mohli pohybovat přirozeně mezi jednotlivými pokoji, s tím, že pobyt v určité místnosti by měl trvat alespoň třicet minut. Na téma jednotlivých konverzací nebyl brán ohled. Některé části nahrávek obsahující osobní údaje

byl odstraněn. Televizní vysílání a hudba byly zakázány kvůli zamezení nahrávání chráněného obsahu.

2.5.2 Konfigurace mikrofonů

Každá oslava byla nahrána pomocí sady šesti zařízení typu Microsoft Kinect. Tyto zařízení byly rozmístěny tak, aby z každé místnosti mohli nahrávat alespoň dva. Každé zařízení obsahuje pole 4 synchronizovaných mikrofonů a jedné kamery. Záznam z jednotlivých zařízení se nahrál na jednotlivé počítače. Navíc pro možnosti přepisu řeči měl každý účastník na sobě binaurální mikrofony typu Soundman OKM II Classic Studio, které byly připojeny do stereo záznamníků typu Tascam DR-05.

Na začátku každé oslavy se zahrálo pípnutí, pro umožnění synchronizace. Nicméně postupem času dochází k rozcházení synchronizace jednotlivých zařízení vlivem jiných taktů a kvůli výpadkům rámců na Microsoft Kinect. Tento problém (nesynchronizované signály z mikrofonů) byl součástí kampaně.

2.5.3 Popis dat

Data byly rozděleny na trénovací, vývojové a evaluační sady.

- Trénovací sada – záznam z 16 oslav, 32 mluvčích, 40 hodin 33 minut nahrávek se 79 980 různými slovy
- Vývojová sada – záznam z 2 oslav, 8 mluvčích, 4 hodiny 27 minut nahrávek se 7 440 různými slovy
- Evaluační sada – záznam z 2 oslav, 8 mluvčích, 5 hodin 12 minut nahrávek s 11 028 různými slovy

Každý záznam obsahuje celkem 32 nahrávek

- 2 nahrávky z binaurálních mikrofonů pro každého ze 4 účastníků
- 4 nahrávky pro každý ze 6 Kinect zařízení

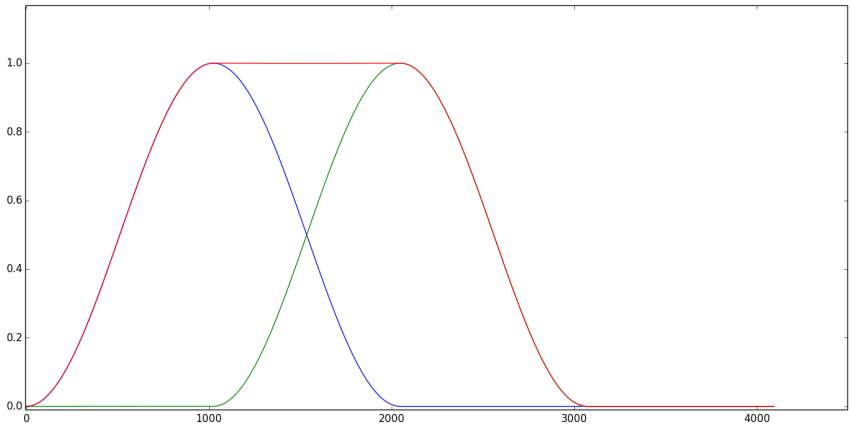
Tato kampaň přinesla navíc problematiku synchronizace nahrávek. K vylepšení nahrávek byl dodán referenční beamformer na bázi [2].

2.6 STFT

STFT neboli krátkodobá Fourierova transformace (z anglického short time Fourier transform) je druh výpočtu Fourierovy transformace po krátkých úsecích pro rychlý výpočet a pro umožnění provádění výpočtu v reálném čase, tj. po blocích vzorků. Nazývá se také jako okénková DFT a je popsaná následující rovnicí

$$X(\theta, t) = \sum_{n=mR}^{mR+N-1} o(n - mR)x(n)e^{-i2\pi(n-mR)\theta/N}, \tag{2.12}$$

kde N je délka okénka, R je krok posunutí sousedních okének a o(n) je okénkovací funkce. Typickou okénkovací funkcí je např. Hammingovo okénko. Posunutí sousedních okének tzv. overlap se používá pro zamezení artefaktů vznikajících na okrajích bloků, kvůli průběhu okénkovací funkce, kdy střed funkce je 1 a směrem ke krajům se snižuje k nule. Této funkce se využívá pro vytvoření spectrogramu signálu. Na obrázku 2.2 je ukázáno překrytí zeleného a modrého okénka a jejich součin červeně.



Obrázek 2.2: Ukázka překrytí okének [16]

2.7 Beamforming

Beamforming[19] je metoda pro zaměření zdroje signálu pomocí více senzorů (více mikrofonů). Umožňuje nám zaměřit se a vylepšit cílový signál, např. řeč, ve zhoršených podmínkách. Zaměřením cílového signálu můžeme odstranit nežádoucí šum. Ukážeme si několik metod, ze kterých se vycházelo při řešení evaluační kampaně CHiME-4.

2.7.1 Delay and sum Beamformer (DSB)

Toto je jeden z nejjednodušších beamformerů. Vychází z principu, že zdrojový signál doputuje na jednotlivé senzory se zpožděním. Toto zpoždění lze vypočítat a aplikací tohoto zpoždění pro jednotlivé mikrofony a jejich následným sečtením zesílíme právě zdrojový signál. Lze jej pospat pomocí následující rovnice:

$$y(t) = \sum_{m=0}^{M-1} w_m x_m(t - T_m) \tag{2.13}$$

Kde w_m představuje váhu beamformeru, x_m je signál na jednotlivých kanálech, M je celkový počet kanálů a T_m je zpoždění odpovídající úhlu dopadu, na který byl beamformer nastaven.

2.7.2 MVDR beamformer

MVDR (z angl. minimum variance distortionless beamformer) neboli beamformer s minimální odchylkou bez zkreslení je další často používaný beamformer. Za předpokladu, že beamformer zpracovává signál následovně

$$y[n] = \boldsymbol{w}^T \boldsymbol{x}[n] = \boldsymbol{w}^T \boldsymbol{a} s[n] + \boldsymbol{w}^T \boldsymbol{v}[n]. \tag{2.14}$$

MVDR beamformer počítá $\boldsymbol{w}$ s důrazem na minimální energii šumu na výstupu a zároveň zachování původní intenzity signálu. Tedy

$$\boldsymbol{w} = arg\ \min_{\boldsymbol{w}} \boldsymbol{w}^T \boldsymbol{C} \boldsymbol{w} \quad w.r.t. \quad \boldsymbol{w}^T \boldsymbol{a} = 1 \tag{2.15}$$

kde $\boldsymbol{C} = E[v[n]v[n]^T]$ (Kovarianční matice šumu)

Řešením $\boldsymbol{w}$ je poté [6]

$$\boldsymbol{w} = \frac{\boldsymbol{C}^{-1}\boldsymbol{a}}{\boldsymbol{a}^H\boldsymbol{C}^{-1}\boldsymbol{a}}, \tag{2.16}$$

zde $\cdot^H$ značí transpozici s komplexním sdružením. K efektivnímu využití tohoto beamformeru v praxi je zapotřebí odhadnout C a a s dostatečnou přesností.

2.7.3 Filter-and-sum beamformer

Kvůli náročnosti výpočtu inverzní matice v rovnici 2.16 a náchylnosti k chybám, byl v práci použit jednodušší beamformer. Tento beamformer funguje na principu filtrování jednotlivých vstupů a následného sečtení na jeden výstup. Potřebuje pro svojí činnost pouze odhad přenosové funkce g. Popis tohoto beamformeru je v rovnici 2.17

$$w_{FSB} = \frac{1}{m}(g^{-1})^*, \tag{2.17}$$

kde proměnná g^{-1} obsahuje reciproční hodnoty k prvkům g, tj. hodnoty relativních přenosových funkcí pro ostatní kanály vůči referenčnímu kanálu. V ideálním prostředí bez odrazů by pak tento beamformer odpovídal delay-and-sum beamformeru, jelikož jednotlivé hodnoty beamformeru odpovídají relativním přenosovým funkcím mezi jednotlivými kanály a reprezentovaly by pouze zpoždění zdrojového signálu na jednotlivých mikrofonech oproti referenčnímu mikrofonu.

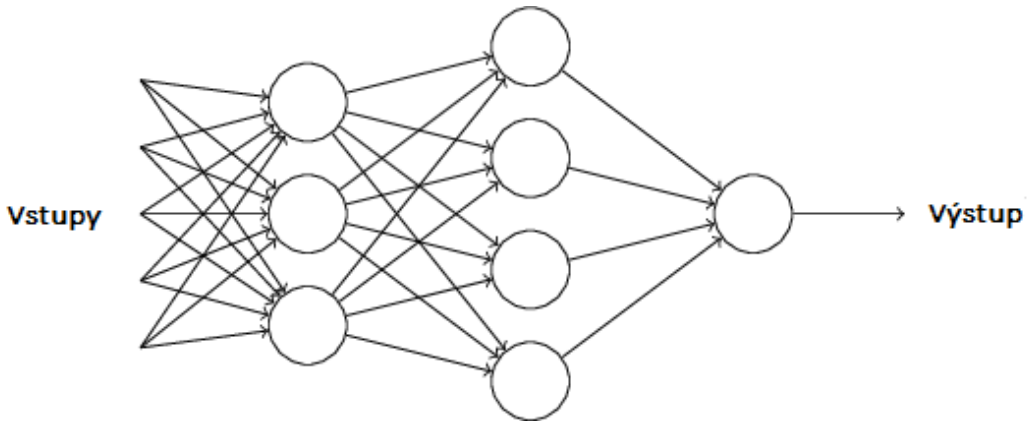
2.8 Neuronové sítě

Umělá neuronová síť se skládá z několika propojených uzlů, nazývaných umělé neurony[18]. Tyto neurony jsou propojeny a mohou mezi sebou přenášet informaci ve formě signálu (většinou ve formě reálného čísla). Jednotlivé neurony mohou mít několik vstupů, ale pouze jeden výstup, který je většinou popsán pomocí nelineární funkce součtu jeho vstupů. Většina spojů mezi neurony má danou váhu, která se

mění postupným učením, určující zesílení nebo potlačení signálu na daném spojení. Neurony mohou mají také nastavenou hodnotu odsazení (angl. bias). Neuron se dá popsat pomocí následující funkce

$$Y = S(\sum_{i=1}^{N}(w_i x_i) + B) \tag{2.18}$$

kde Y je výstup neuronu, w_i jsou váhy jednotlivých synapsí, x_i jsou vstupy neuronu, B je odsazení (angl. Bias), S je přenosová nebo také aktivační funkce. Propojením jednotlivých neuronů získáme síť , která reprezentuje vyhodnocení daného problému. Většinou sítě neobsahují smyčky, takovým to sítím se říká dopředné (angl. feedforward). Pokud síť obsahuje zpětnou vazbu, tak se jim říká rekurentní. Ukázka sítě je na obrázku 2.3.

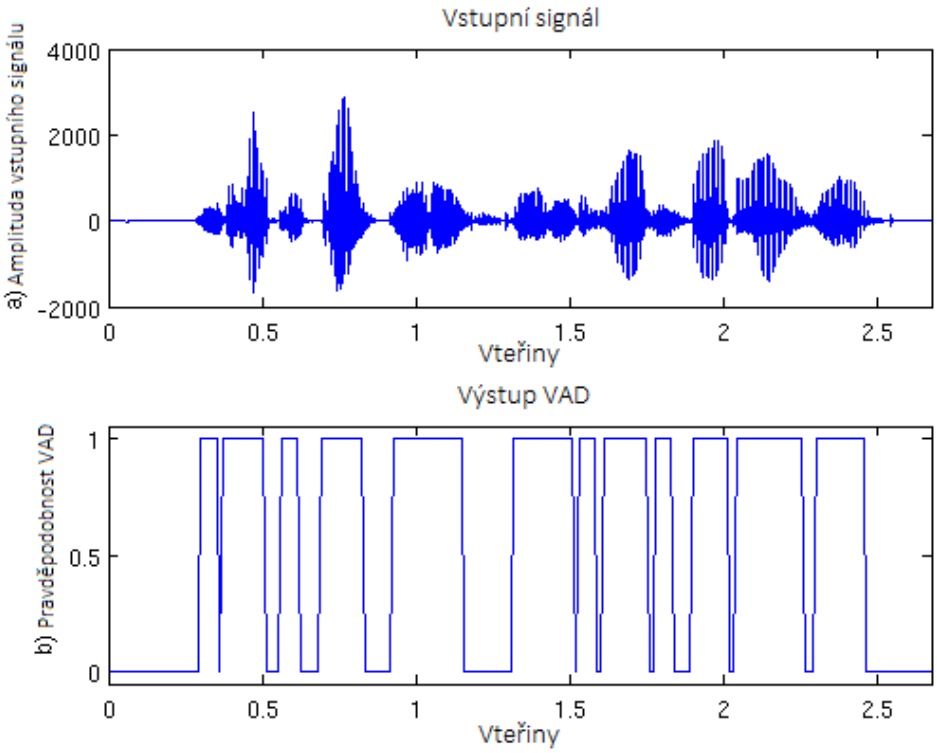


Obrázek 2.3: Ukázka neuronové sítě [18]

Každá síť se dá rozdělit do několika vrstev. Vrstvy síťe na obrázku můžeme popsat následovně. První vrstva, nazývaná též vstupní, vypočítá svůj výstup v závislosti na vstupech sítě. Druhá vrstva získává své vstupy z výstupů první vrsvty a výstupy neuronů v této vrstvě zase slouží jako vstup pro třetí vrstvu. Druhou vrstvu, případně další vrstvy jejíž výstupy jdou na vstup dalších vrstev nazýváme tzv. skryté vrstvy. Třetí vrstva je poslední a její výstup slouží jako výstup celé sítě, nazývá se proto jako výstupní.

2.9 Voice Activity Detector (VAD)

Voice Activity Detector neboli detektor aktivity hlasu je nástroj, který zjišťuje v nahrávkách, zda se v daném úseku vyskytuje hlas či nikoli. Tato funkcionalita se hodí zejména při odhadování šumu v nahrávkách. Pokud víme, že se v daném úseku nachází řeč, můžeme toho využít a vypočítat relativní přenosovou funkci mezi jednotlivými signály, a tím zaměřit zdrojový signál. K jeho vytvoření se použily neuronové sítě. VAD byl natrénován, aby odhadoval zesílení Wienerova filtru, tj. hodnoty od 0 do 1. Tato data projdou několika vrstvami hluboké neuronové sítě s tím, že každý průchod vylepšuje přesnost VAD. Na obrázku 2.4 je zobrazena ukázka funkce VAD.



Obrázek 2.4: Ukázka funkce VAD [5]

V práci budeme používat enhancer [6], který má 2 druhy VAD: sVAD a dVAD.

2.10 sVAD

Tento druh detektoru pracuje vždy nad jedním rámcem signálu. Byl natrénován pro odhadnutí zisků Wienerova filtru. Každý frame STFT je reprezentován pomocí výpočtu banky filtrů mel (angl. mel filter bank) a jeho 40 hodnot. Vstupní vektor spojuje současný frame s 10 frame před a 2 po. sVAD se skládá z 5 skrytých vrstev, tři po 256 neuronech a dvě po 128 neuronech. Všechny tyto neurony obsahují sigmoidní aktivační funkci. Výstupem je hodnota reprezentující pravděpodobnost výskytu řeči[6].

2.11 dVAD

Tento detektor pracuje také nad jedním rámcem, ale navíc také nad každým frekvenčním binem daného rámce. Skládá se z 6 menších DNN, které zpracovávají jedno ze šesti frekvenčních pásem. Výstupem každého DNN je vektor hodnot v intervalu od 0 do 1 reprezentující pravděpodobnost výskytu řeči v daném frekvenčním binu a rámci. Každý z těchto DNN se opět skládal z 5 skrytých vrstev. První dvě vrsvty se skládají z 350 neuronů, třetí vrstva z 256 neuronů a poslední dvě vrstvy z 128 neuronů. Všechny tyto neurony mají usměrňovací aktivační funkci[6].

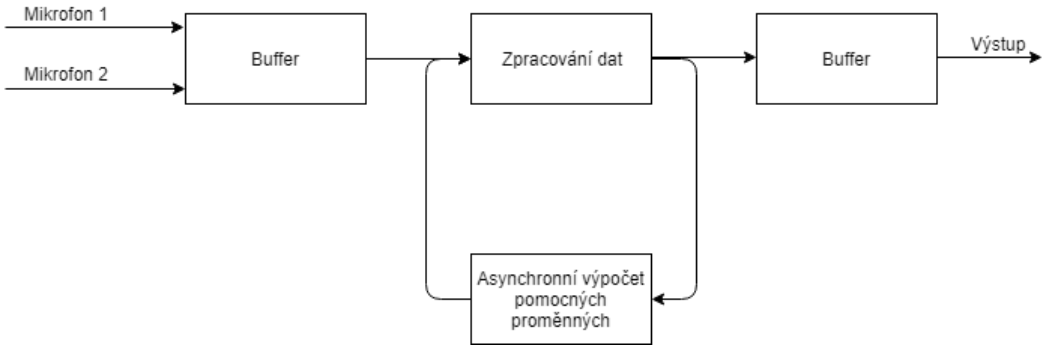
Kapitola 3

Návrh řešení

3.1 Koncept

Hlavním cílem aplikace bylo, aby fungovala v on-line režimu, tj. zpracovávala data v reálném čase. Za tímto účelem jsem vybral pro její zpracování programovací jazyk C++, který patří k jedněm z nejrozšířenějších. Hlavními důvody pro tuto rozšířenost jsou rychlost, přehlednost a velké množství knihoven vytvořených pro specifické účely. Zpracování v reálném čase přináší změnu, jelikož většina řešení pracuje s už pořízenými nahrávkami, a tak nemusí klást důraz na potřebu zpracovávat signál po malých blocích a dostatečně rychle, aby jsme si neomezovali načítání dalšího bloku.

Postup zpracování dat v aplikaci se dá popsat obecně pomocí následujícího schématu.



Obrázek 3.1: Schéma popisující zpracování dat

Data z mikrofonů se ukládají do bufferu a po dosažení určité hodnoty zaplnění jsou poslány ke zpracování. Zpracování dat se skládá z následujících kroků:

- Převzorkovaní dat ze vzorkovací frekvence vstupního zařízení na požadovanou vzorkovací frekvenci, ve které budeme data zpracovávat
- Převedení dat z časové domény do frekvenční domény pomocí STFT
- Výpočet metody pro redukci šumu a aplikace této metody na data
- Převedení dat zpět do časové domény pomocí STFT
- Převzorkování dat na vzorkovací frekvenci výstupního zařízení

Pro zajištění rychlého zpracování dat je možné některé náročné operace počítat asynchronně a po dokončení výpočtu aktualizovat proměnné, které se použijí při zpracování dalšího bufferu. Přesnější popis zpracování pro jednotlivá řešení je v kapitole 4.

3.2 Metody

3.2.1 Metoda 1

První metoda vychází ze systému k vylepšení nahrávek navrženém na ústavu ITE [6] pro evaluační kampaň CHiME4. Jelikož se tato metoda skládá z více možností pro vylepšení nahrávek, tj. různé způsoby výpočtu VAD a dva typy beamformeru, rozhodl jsem se implementovat pouze jeden VAD a jeden typ beamformeru. Zvolil jsem výpočet VAD pomocí metody mVAD, jelikož tato metoda je jednodušší na výpočetní náročnost. Beamformer jsem zvolil FSB, jelikož dosahoval lepších výsledků a byl jednodušší na implementaci. Tato metoda byla navržena pro vzorkovací frekvenci 16 kHz.

Vstupní signál převedeme z časové domény do frekvenční domény pomocí Krátkodobé Fourierovy transformace. S daty ve frekvenční doméně začneme počítat VAD.

Nejdříve vypočítáme hodnotu automatického zesílení VAD porovnáním energie současného bloku vstupů s výchozí hodnotou energie z VAD. Výpočet probíhá následovně

$$E = \sum_{n=0}^{N} |X(n)|^2$$
$$g = \sqrt{\frac{VAD_{AutoGain\_baseE}}{E}} \tag{3.1}$$

kde E představuje energii právě zpracovaného vstupu na referenčním mikrofonu, g je zesílení VAD. Tuto hodnotu ukládáme do kruhového bufferu VAD_AutoGain_Mem, kde uchováváme poslední 4 hodnoty. Hodnotu E uložíme do bufferu EHistory, který uchovává posledních 150 hodnot. Poté zjistíme zda současná energie má vyšší hodnotu než je práh ticha. Tento práh se vypočítá jako maximální hodnota z průměrů hodnot v bufferech EHistory a silHistory. Pokud je energie menší, uloží se tato hodnota do bufferu silHistory. Následně se absolutní hodnota signálu použije jako vstup pro neurony mVAD, který se skládá ze 4 vrstev, jejichž funkcionalitu si popíšeme.

V první vrstvě se od vstupu odečte průměrná hodnota inicializační energie a vydělí se směrodatnou odchylkou. Poté se vynásobí jednotlivé hodnoty váhami z matice W1 a přičte se bias B1. Následně se použije aktivační funkce tzv. usměrňovač (angl. Rectifier), tj. funkce, která vybere pouze kladné hodnoty. Výstup z této funkce se použije jako vstup do další vrstvy sítě. V další vrstvě se už vstup pouze vynásobí váhami W2 a přičte k nim bias B2. Následuje stejná aktivační funkce. Další vrstva funguje obdobně, ale používá parametry pro váhy W3 a bias B3. Poslední vrstva přijme vstup z předchozí, ale její výstup je použit pro hodnoty VAD.

3.2.2 Výpočet relativní přenosové funkce

Po výpočtu VAD přichází na řadu výpočet přenosové funkce RTF pomocí vzájemných hustot výkonného spektra na jednotlivých kanálech[7]. Tento výpočet vychází

z předpokladu, popsaném rovnicí , že zdrojový signál na druhém mikrofonu je pouze modifikovaný pomocí přenosové funkce, kterou se snažíme zjistit.

$$\begin{aligned} x(t) &= s(t) + u(t) \\ y(t) &= a(t) * s(t) + w(t) \end{aligned} \tag{3.2}$$

Ekvivalentní problém je vzít v potaz LTI systém jehož vstup x(t) a výstup y(t) jsou na sobě závislé, podle následujícího vzorce

$$y(t) = a(t) * x(t) + v(t) \tag{3.3}$$

kde $a(t)$ je přenosová funkce, kterou se snažíme zjistit a $v(t)$ je sečtený šum. Za těchto předpokladů se rozdělí pozorovaný interval na M podintervalů a pro každý interval $m = (m = 1, 2, ..., M)$ se vypočítá vzájemná hustota výkonného spektra $\phi_{yx}^{(m)}(\omega)$ mezi y a x následovně

$$\phi_{yx}^{(m)}(\omega) = A(\omega)\phi_{xx}^{(m)}(\omega) + \phi_{vx}^{(m)}(\omega) \tag{3.4}$$

kde $A(\omega)$ je Fourierova transformace $a(t)$, tj. relativní přenosová funkce systému, a $\phi_{vx}(\omega)$ je nezávislá na intervalu m v souvislosti se stacionaritou $v(t)$ a $u(t)$, a nekorelací mezi $v(t)$ a $s(t)$. Nechť $\hat{\phi}_{yx}^{(m)}(\omega)$, $\hat{\phi}_{xx}^{(m)}(\omega)$ a $\hat{\phi}_{vx}^{(m)}(\omega)$ jsou odhady pro $\phi_{yx}^{(m)}(\omega)$, $\phi_{xx}^{(m)}(\omega)$ a $\phi_{vx}^{(m)}(\omega)$. Poté

$$\begin{aligned} \hat{\phi}_{yx}^{(m)}(\omega) &= A(\omega)\hat{\phi}_{xx}^{(m)}(\omega) + \hat{\phi}_{vx}^{(m)}(\omega) \\ &= A(\omega)\hat{\phi}_{xx}^{(m)}(\omega) + \phi_{vx}^{(m)}(\omega) + \epsilon^{(m)}(\omega) \end{aligned} \tag{3.5}$$

kde

$$\epsilon^{(m)}(\omega) = \hat{\phi}_{vx}^{(m)}(\omega) - \phi_{vx}^{(m)}(\omega). \tag{3.6}$$

Přepis do maticového tvaru z [7]

$$\begin{aligned}\mathbf{z} &\triangleq \begin{bmatrix}\hat{\phi}_{yx}^{(1)}(\omega)\\ \hat{\phi}_{yx}^{(2)}(\omega)\\ \vdots\\ \hat{\phi}_{yx}^{(M)}(\omega)\end{bmatrix} = \begin{bmatrix}\hat{\phi}_{yx}^{(1)}(\omega) & 1\\ \hat{\phi}_{yx}^{(2)}(\omega) & 1\\ \vdots & \vdots\\ \hat{\phi}_{yx}^{(M)}(\omega) & 1\end{bmatrix} \begin{bmatrix}A(\omega)\\ \phi_{yx}^{(2)}(\omega)\end{bmatrix} + \begin{bmatrix}\epsilon^{(1)}(\omega)\\ \epsilon^{(2)}(\omega)\\ \vdots\\ \epsilon^{(M)}(\omega)\end{bmatrix} \\ &\triangleq \hat{\boldsymbol{G}}\boldsymbol{\theta} + \boldsymbol{\epsilon}\end{aligned} \tag{3.7}$$

Odhad $\boldsymbol{\theta}$ pomocí metody vážených nejmenších čtverců se získá pomocí

$$\begin{aligned}\begin{bmatrix}A(\omega)\\ \phi_{yx}^{(2)}(\omega)\end{bmatrix} &= \hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{argmin}(\boldsymbol{z} - \hat{\boldsymbol{G}}\boldsymbol{\theta})^H \boldsymbol{W}(\boldsymbol{z} - \hat{\boldsymbol{G}}\boldsymbol{\theta}) \\ &= (\hat{\boldsymbol{G}}^{\boldsymbol{H}}\boldsymbol{W}\hat{\boldsymbol{G}})^{-1}\hat{\boldsymbol{G}}^{\boldsymbol{H}}\boldsymbol{W}\boldsymbol{z}\end{aligned} \tag{3.8}$$

kde $\mathbf{W}$ je kladná Hermitovská matice vah, H označuje sdruženou transpozici a $\hat{\boldsymbol{G}}^{\boldsymbol{H}}\boldsymbol{W}\hat{\boldsymbol{G}}$ musí být regulární. Matice vah je posléze dána rovnicí

$$W_{mn} = \begin{cases} T_m, & m = n \\ 0, & m \neq n \end{cases} \tag{3.9}$$

kde T_m je délka podintervalu m, aby delší intervali měli větší váhy. V tomto případě se rovnice 3.8 redukuje na

$$\hat{A}(\omega) = \frac{\langle \hat{\phi}_{yx}(\omega)\hat{\phi}_{xx}(\omega)\rangle - \langle \hat{\phi}_{yx}(\omega)\rangle\langle \hat{\phi}_{xx}(\omega)\rangle}{\langle \hat{\phi}_{yx}^2(\omega)\rangle\langle \hat{\phi}_{xx}(\omega)\rangle^2} \tag{3.10}$$

Jakmile máme takto odhadnutou přenosovou funkci použijeme jednoduchý Filter and sum beamformer popsaný v kapitole 2.7.3.

3.2.3 Metoda 2

Druhá metoda použitá v práci vychází z [15], kde je navržena metoda pro extrakci nezávislých vektorů (IVE), která spadá pod problematiku slepé separace signálu. Tato metoda vychází z extrakce nezávislých komponent (ICE), která vychází z analýzy nezávislých komponent (angl. Independent component analysis, ICA). ICA je

postavena na principu, že signály na d senzorech jsou lineární směsí d originálních signálů, které jsou mezi sebou nezávislé. Model signálu se dá popsat následující rovnicí

$$\boldsymbol{x(n)} = \boldsymbol{A}\boldsymbol{u}(n) \tag{3.11}$$

kde $\boldsymbol{x(n)}$ je vektor smíchaných signálů o velikosti $d \times 1$, $\boldsymbol{A}$ je mixovací matice velikosti $d \times d$ a $\boldsymbol{u(n)}$ je vektor originálních signálů. Cílem analýzy nezávislých komponent je odhadnout $\boldsymbol{A}^{-1}$ pomocí $\boldsymbol{x(n)}, n = 1, ..., N$, skrze najití demixovací matice $\boldsymbol{W}$ takové, aby $\boldsymbol{y(n)} = \boldsymbol{W}\boldsymbol{u(n)}$ byly co nejvíce nezávislé.

Pro extrakci nezávislé komponenty (ICA) zvolíme cílový signál (např. řeč ve zvukových signálech) $s = u_1$ a a bude první sloupec $\boldsymbol{A}$. $\boldsymbol{A}$ pak můžeme rozdělit na $\boldsymbol{A} = [\boldsymbol{a}, \boldsymbol{A}_2]$. Poté x se dá přepsat do tvaru

$$\boldsymbol{x} = \boldsymbol{a}s + \boldsymbol{y} \tag{3.12}$$

kde $\boldsymbol{y} = \boldsymbol{A}_2\boldsymbol{u}_2$ a $u_2 = [u_2, ..., u_d]^T$. Jelikož ICE nemá za cíl zjistit $\boldsymbol{A}_2$ nebo rozložení $\boldsymbol{y}$ na jednotlivé nezávislé signály, můžeme mixovací matici zapsat jako $\boldsymbol{A}_{ICE} = [a, Q]$, kde Q je prozatím nevyjádřené.

Poté $\boldsymbol{x}$ vyjádříme jako

$$\boldsymbol{x} = \boldsymbol{A}_{ICE}\boldsymbol{v} \tag{3.13}$$

kde $\boldsymbol{v} = [s; \boldsymbol{z}]$, a $\boldsymbol{y} = \boldsymbol{Q}\boldsymbol{z}$. Inverzní matice $\boldsymbol{W_{ICE}}$ se rozdělí na

$$\boldsymbol{W}_{ICE} = \begin{pmatrix} \boldsymbol{w}^H \\ \boldsymbol{B} \end{pmatrix} \tag{3.14}$$

a $\boldsymbol{a}$

$$\boldsymbol{a} = \begin{pmatrix} \gamma \\ \boldsymbol{g} \end{pmatrix} \tag{3.15}$$

$\boldsymbol{B}$ musí být orthogonální k $\boldsymbol{a}$, tj. $\boldsymbol{B}\boldsymbol{a} = 0$, což zajistí, že signál rozdělený na dolní části $\boldsymbol{W}_{ICE}$, neobsahuje podíl s. $\boldsymbol{B}$ můžeme vybrat následovně

$$\boldsymbol{B} = (\boldsymbol{g} \quad -\gamma \boldsymbol{I}_{d-1}), \tag{3.16}$$

kde $\boldsymbol{I}_{d-1}$ je jednotková matice. $\boldsymbol{w}$ se rozdělí na

$$\boldsymbol{w} = \begin{pmatrix} \beta \\ \boldsymbol{h} \end{pmatrix} \tag{3.17}$$

matice $\boldsymbol{W}_{ICE}$ a $\boldsymbol{A}_{ICE}^{-1}$ jsou popsány následovně

$$\begin{aligned} \boldsymbol{W}_{ICE} &= \begin{pmatrix} \boldsymbol{w}^H \\ \boldsymbol{B} \end{pmatrix} = \begin{pmatrix} \bar{\beta} & h^H \\ \boldsymbol{g} & -\gamma \boldsymbol{I}_{d-1} \end{pmatrix} \\ \boldsymbol{A}_{ICE} &= \begin{pmatrix} \boldsymbol{a} & \boldsymbol{Q} \end{pmatrix} = \begin{pmatrix} \gamma & h^H \\ \boldsymbol{g} & \frac{1}{\gamma}(\boldsymbol{g}\boldsymbol{h}^H - \boldsymbol{I}_{d-1}) \end{pmatrix}, \end{aligned} \tag{3.18}$$

kde β a γ jsou závislé následovně

$$\bar{\beta}\gamma = 1 - \boldsymbol{h}^H \boldsymbol{g}. \tag{3.19}$$

Extrakce nezávislého vektoru, vychází z principu ICE, ale snaží se extrahovat vektor z k mixtur, místo jediné .

$$\boldsymbol{x}^k = \boldsymbol{A}_{ICE}^k \boldsymbol{v}^k, \qquad k = 1, ..., K. \tag{3.20}$$

V článku [15] je uveden algoritmus pro výpočet IVE, který předpokládá K mixtur a N vzorků. Tento algoritmus vezme iniciální hodnoty a následně vyhledá nezávislý signál a setrvá u něj. Snaha je aby tento signál byl náš cílový, tedy řeč. Algoritmus byl

přepsán do následujícího tvaru, aby mohl být použit pro online zpracování signálu. Jako mixtury K použijeme jednotlivé frekvence signálu X.

$$\begin{aligned}
\boldsymbol{C}^k &= (1-\lambda)\boldsymbol{C}^k + \lambda \boldsymbol{X}^k (\boldsymbol{X}^k)^H / N \\
a^k &= \frac{\boldsymbol{C}^k \boldsymbol{w}^k}{(\boldsymbol{w}^k)^H \boldsymbol{C}^k \boldsymbol{w}^k} \\
\bar{s}^k &= (\boldsymbol{w}^k)^H \boldsymbol{X}^k \\
\phi^k &= f^k(\bar{s}^1, ..., \bar{s}^k) \\
\xi^k &= (1-\lambda)\xi^k + \lambda(\phi^k(\bar{s}^k)^T)/N \\
\Delta^k &= a^k - \boldsymbol{X}^k * (\phi^k)^T / \xi^k / N \\
w^k &= w^k + \mu \Delta^k
\end{aligned} \tag{3.21}$$

Kde $\boldsymbol{C}(k)$ je odhad $\boldsymbol{C}_x = E[xx^H]$, λ určuje poměr aktualizace proměnných v souvislosti se současným vzorkem. Gradient $\Delta(k)$ upravuje v závislosti na proměnné μ separující vektor w, a tím mění výběr vektoru k extrakci. $\phi(k)$ je nelineární funkce závislá na všech K frekvencích. Popsaná následující rovnicí

$$f^k(s^1, ..., s^K) = \frac{\bar{s}^k}{\sum_{i=1}^{K} (s^i \bar{s}^i)}, \tag{3.22}$$

kde $\bar{s}^k$ značí komplexně sdružené číslo k s^k.

Kapitola 4

Realizace

4.1 Vývojové prostředí

Pro vývoj aplikace jsem použil vývojové prostředí Microsoft Visual Studio 2017 Enterprise. Toto vývojové prostředí je nejrozšířenější při vývoji aplikací na operačním systému Windows v programovacích jazycích C, C++, VB, C# a .NET. Verze Enterprise navíc umožňuje funkci Profiling, tj. funkce k analyzování výkonu aplikace, vytížení procesoru, paměti, jaké funkce jsou volány často apod. Tato funkcionalita mi přišla vhod zejména při ladění aplikace s důrazem na rychlost.

4.2 API a ASIO4ALL

Pro komunikaci mezi zvukovým zařízením a aplikacemi slouží tzv. API (application programming interface). Při práci s audiem v reálném čase potřebujeme minimalizovat latency, tj. zpoždění dat získaných na zařízení (mikrofon) a přijetím v aplikaci ke zpracování. Tato hodnota je ovlivněná velikostí bufferů, které jsou mezi jednotlivými prvky zpracování zvuku. Při nahrávání to jsou tyto prvky hardware, ovladač pro daný hardware (driver), zvukový engine operačního systemů. Ke zmenšení této latence je vhodné použít jiné API nebo jiný ovladač pro hardware. Jedním z nejrychlejší API je ASIO (Audio Stream Input/Output). Problém tohoto protokolu je

jeho závislost na hardwaru. Řešením je ovladač zařízení ASIO4ALL[14], který je univerzální a funguje na většine zvukových karet.

4.3 Knihovny

Pro realizaci aplikace bylo za účelem zrychlení a zpřehlednění aplikace použito několik knihoven. Tyto knihovny se zaměřovali na práci se zvukem, zpracování signálu, výpočty s maticemi.

4.3.1 PortAudio

PortAudio[10] je knihovna pro práci se vstupy a výstupy na zvukovém rozhraní počítače. Tato knihovna je cross-platform (tj. knihovna je nezávislá na operačním systému), open-source a poskytovaná bezplatně pod licencí MIT. Tato knihovna byla využita pro nahrání zvuku na mikrofonech a po úpravě následné přehrání upraveného zvuku na výstupu. K tomuto účelu používá tato knihovna tzv. stream, tj. neustálý proud dat ze zařízení. Tato data se předávají do a z naší aplikace pomocí asynchronní callback funkce, ve formě bufferu, který se musí nejdříve naplnit daty. Pro účely naší aplikace byla tato knihovna zkompilována pouze pro API ASIO (z důvodu rychlosti), takže pokud není k dispozici hardware podporující toto API, je potřeba stáhnout ovladač ASIO4ALL.

4.3.2 Armadillo

Armadillo[11] je knihovna vytvořená pro rychlou a přehlednou práci s lineární algebrou v programovacím jazyce C++. Tato knihovna využívá syntaxi podobnou vyšším programovacím jazykům jako jsou Octave nebo Matlab pro jednoduché vyjádření matematických vzorců známým způsobem. Poskytuje objekty reprezentující vektory, matice a krychle a operace nad těmito objekty. Pro rychlé výpočty používá integrované knihovny LAPACK a BLAS. V mém projektu jsem knihovnu BLAS nahradil knihovnou OpenBLAS[12], která byla optimalizovaná pro větší rychlost zpracování.

4.3.3 SigPack

Knihovna SigPack[13] poskytuje funkce pro zpracování signálů pomocí funkcí napodobující syntaxi Matlab[4]. Mezi tyto funkce patří vytvoření FIR a IIR filtrů, vytvoření okénka pro okénkovací funkci, např. Hamming okénko, Hanningovo okénko či Kaiserovo okénko.Mezi další funkce patří převzorkování signálu, vykreslování spektra a spektrogramu, funkce pro zpoždění signálu a další.

4.3.4 FFTW3

Fastest Fourier transform in the west[17], neboli nejrychlejší Fourierova transformace na západě je knihovna, která, jak její název napovídá, zprostředkovává rychlý výpočet diskrétní Fourierovy transformace. Zvládá jak reálná, tak komplexní vstupní data, jedno i vícerozměrné transformace. Tato knihovna se osvědčila jako velmi užitečná, jelikož výpočet Fourierovy transformace může být výpočetně náročný, ale díky této knihovně jsem na tento problém nenarazil.

4.4 Program

Aplikaci jsem se rozhodl vytvořit jako Windows Console Application, tedy aplikaci běžící v příkazovém řádku operačního systému Windows. U tohoto typu aplikací lze ovládat program pomocí zadáváním příkazů v textovém tvaru.

4.4.1 Běh aplikace

Při spuštění aplikace nejprve proběhne inicializace proměnných používaných pro výpočty, ukládání a práci s daty. Tato inicializace proběhne zavoláním funkce `initialize_var()`. Tato funkce je typu void a slouží pouze pro nastavení výchozích hodnot a inicializaci proměnných.

Poté se zavolá funkce `Pa_Initialize()`, která inicializuje knihovnu PortAudio. To spočívá v inicializaci vnitřních datových struktur a připravení zvukových rozhraní na komunikaci. Následně se musí zvolit zařízení, ze kterého se budou zís-

kávat vstupní data. V mém případě jsem použil přiřazení zařízení pomocí funkce `Pa_GetDefaultInputDevice()`, která zjistí výchozí zařízení dostupné pro zvolené zvukové API. Použití funkce vypadá následovně:

```
inputParameters.device = Pa_GetDefaultInputDevice(); /* default input device */
if (inputParameters.device == paNoDevice) {
    fprintf(stderr, "Error: No default input device.\n");
    goto error;
}
```

Obrázek 4.1: Ukázka funkce Pa_GetDefaultInputDevice()

Jakmile získáme vstupní zařízení, je nutné mu nastavit parametry, se kterými bude pracovat. Jakým způsobem se nastavují parametry je zobrazeno na obrázku 4.2.

```
inputParameters.channelCount = NCHANS;       /* stereo input */
inputParameters.sampleFormat = PA_SAMPLE_TYPE;
inputParameters.suggestedLatency = 0;//Pa_GetDeviceInfo(inputParameters.device)->defaultLowInputLatency;
inputParameters.hostApiSpecificStreamInfo = NULL;
```

Obrázek 4.2: Ukázka nastavení vstupních parametrů

Hlavními parametry, které se musejí nastavit, jsou `channelCount` neboli počet kanálů na zařízení, v našem případě je tento počet reprezentován konstantou `NCHANS` reprezentující dva vstupní kanály. Počet kanálů byl zvolen 2, jelikož jsem měl k dispozici mikrofonní pole s dvěma mikrofony Dále pak v jakém formátu jsou získávány jednotlivé vzorky (parametr `sampleFormat`), v našem případě konstanta `PA_SAMPLE_TYPE` reprezentuje datový typ float o velikosti 32 bitů. Parametr `suggestedLatency` neboli doporučená latence se nastaví pomocí volání parametru `defaultLowInputLatency`, který je jednou z informací získaných voláním funkce `Pa_GetDeviceInfo()`. Tato latence je tedy závislá na zvoleném zařízení a bude se lišit podle použitého hardwaru. Na mém počítači jsem získal hodnotu 8.7 ms. Parametr `hostApiSpecificStreamInfo` se nastavuje na `NULL`, jelikož je výchozí zvukové API nastaveno při sestavení knihovny PortAudio. V našem případě je toto zvukové API ASIO.

Obdobným způsobem se nastaví výstupní zařízení a jeho parametry.

```
outputParameters.device = Pa_GetDefaultOutputDevice(); /* default output device */
if (outputParameters.device == paNoDevice) {
    fprintf(stderr, "Error: No default output device.\n");
    goto error;
}
outputParameters.channelCount = 2;       /* stereo output */
outputParameters.sampleFormat = PA_SAMPLE_TYPE;
outputParameters.suggestedLatency = 0;//Pa_GetDeviceInfo(outputParameters.device)->defaultLowOutputLatency;
outputParameters.hostApiSpecificStreamInfo = NULL;
```

Obrázek 4.3: Ukázka nastavení výstupních parametrů

Po nastavení parametrů vstupního a výstupního zařízení se může spustit stream zpracovávající data pomocí funkce `Pa_OpenStream()`. Tato funkce je volána s několika parametry, ukázáno na obrázku 4.4.

```
err = Pa_OpenStream(
    &stream,
    &inputParameters,
    &outputParameters,
    SAMPLE_RATE,
    FRAMES_PER_BUFFER,
    0,
    paCallback,
    &data);
if (err != paNoError) goto error;
```

Obrázek 4.4: Ukázka funkce `Pa_OpenStream()`

Parametry, které se předávají, jsou odkaz na stream, představující tok dat mezi aplikací a jedním nebo více zvukovými zařízeními. Dále pak odkazy na vstupní a výstupní parametry, parametry nastavující vzorkovací frekvenci a počet framů (rámců) v jednom bufferu. Dalším možný parametr nastavuje různé flagy reprezentující např. vypnutí oříznutí dat (Clipping). Musí se také předat metoda použitá pro zpracování dat jakmile jsou k dispozici tzv. Callback metoda. V neposlední řadě se předá odkaz na proměnnou obsahující uživatelská data, kterou lze využít např. pro komunikaci mezi hlavním vláknem aplikace a Callback metodou.

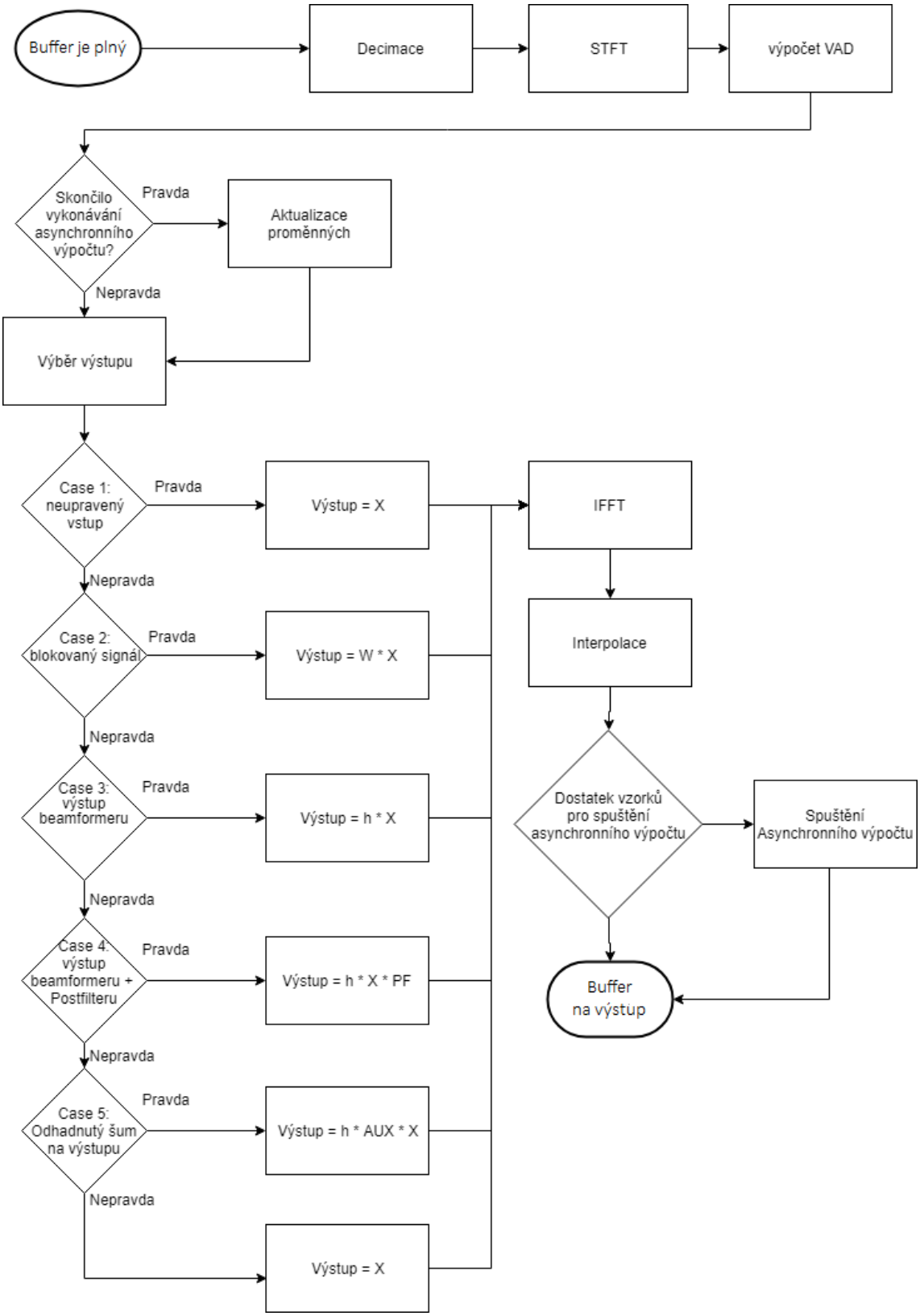
Odkazy na vstupní a výstupní data ukazují na proměnné popsané výše. Vzorkovací frekvence je nastavená pomocí konstanty `SAMPLE_RATE` reprezentující hodnotu 48 000 Hz. Tato hodnota byla zvolena, jelikož metoda je přizpůsobené vzorkovací frekvenci 16 000 Hz a mikrofony podporovaly pouze vzorkovací frekvence 44 100 Hz nebo vyšší. Nejnižší vzorkovací frekvence, která se dá převzorkovat pomocí celého

čísla byla tedy 48 000 Hz (poměr převzorkování mezi 48 a 16 kHz je 3). Počet rámců v bufferu je nastaven konstantou `FRAMES_PER_BUFFER` na hodnotu 384. Tato hodnota odpovídá trojnásobku 128, tedy posunu okénka v metodě. Parametr pro flagy je nastaven na nulu, tedy žádné flagy nejsou nastaveny. Další parametr odkazuje na metodu zpracovávající data s názvem paCallback, tuto metodu popíšeme později. Uživatelská data jsou předávána, ale neslouží žádnému účelu.

Jakmile máme takto inicializovaný stream, můžeme ho spustit pomocí funkce `Pa_StartStream(stream)`. Po zavolání této funkce se bude periodicky (po zaplnění bufferu) volat funkce paCallback až do zavolání `Pa_StopStream(stream)`, funkce na zastavení streamu. Nakonec se zavolají funkce `Pa_CloseStream(stream)` a `Pa_Terminate()`, které ukončí stream a respektivě vypne knihovnu PortAudio.

4.4.2 Metoda paCallback pro metodu 1

Tato metoda je základním bodem aplikace, protože zde probíhá většina výpočtů a transformací zvukového signálu. Průběh této metody je popsán pomocí diagramu na obrázek 4.5.



Obrázek 4.5: Diagram popisující průběh funkce paCallback

Nejdříve vezmeme signál na vstupu, který získáme ve formě pointeru, a přiřadíme jeho hodnoty dvěma vektorům reprezentujícím levý a pravý vstupní kanál.

```
for (i = 0; i < framesPerBuffer; i++)
{
    in_L(i) = *in++;/*Left*/
    in_R(i) = *in++;/*Right*/
}
```

Obrázek 4.6: Ukázka rozdělení vstupních dat na levý a pravý kanál

Jakmile máme takto rozděleny vstupní signály, decimujeme je ze vzorkovací frekvence 48 kHz na 16 kHz. Toto provedeme pomocí operace konvoluce signálu s vektorem reprezentující Low-pass filter (LPF). K realizaci této operace jsem využil metody `conv()` z knihovny Armadillo a k realizaci LPF jsem využil funkci `fir1()` z knihovny SigPack. LPF byl navrhnut jako filtr řádu 40 s mezní frekvencí nastavenou na 16 kHz tedy na třetinu vstupní frekvence 48 kHz. Výstupní vektor je ještě modifikován vektorem reprezentující krok metody overlap-add. Z takto upraveného vektoru `x_L` získáme nový vektor `buff_down_L`, skládající se pouze z každé třetí hodnoty původního vektoru. Následující obrázek ukazuje decimaci v kódu aplikace.

```
/*Downsampling*/
x_L = conv(in_L, LPF);
x_L = x_L + ola_down_L;
x_R = conv(in_R, LPF);
x_R = x_R + ola_down_R;

for (uword i = 0; i < FILTER_ORDER; i++) {
    ola_down_L(i) = x_L(FRAMES_PER_BUFFER + i);
    ola_down_R(i) = x_R(FRAMES_PER_BUFFER + i);
}

for (uword i = 0; i < FFTSHIFT; i++) {
    buff_down_L(i) = x_L(3 * i);
    buff_down_R(i) = x_R(3 * i);
}
```

Obrázek 4.7: Ukázka decimace v kódu

Vektory `buff_down_L` a `buff_down_R` jsou následně zařazeny do kruhových bufferů `timeDataIn_L` a `timeDataIn_R`, uchovávající poslední 4 iterace, pro další zpracování. Tyto buffery slouží k uchování hodnot, které zpracováváme pomocí STFT.

Proto uchovávají 512 hodnot (velikost okénka) a posouvají se o 128 hodnot (každý nový buffer).

```
//Shift
timeDataIn_L.shed_rows(0, FFTSHIFT - 1);
timeDataIn_L.insert_rows(384, buff_down_L);

timeDataIn_R.shed_rows(0, FFTSHIFT - 1);
timeDataIn_R.insert_rows(384, buff_down_R);
```

Obrázek 4.8: Ukázka rotace bufferu timeDataIn

Na tyto buffery je následně aplikována okénková funkce a vzniklý vektor `win_L` je použit jako vstupní parametr pro Fourierovu transformaci. Jelikož vstupní data Fourierovy transformace jsou reálná je výstupní vektor sdruženě symetrický a z vektoru `X_L_t` použijeme pouze prvních 257 hodnot.

```
/*STFT*/
win_L = timeDataIn_L % window;
X_L_t = ss.fft(win_L);
X_L = X_L_t.rows(0, NFREQS - 1);
win_R = timeDataIn_R % window;
X_R_t = ss.fft(win_R);
X_R = X_R_t.rows(0, NFREQS - 1);
```

Obrázek 4.9: Ukázka výpočtu STFT v kódu

Následuje výpočet VAD podle metody popsané v 3.2.1.

Po výpočtu VAD a pomocných proměnných přijde na řadu volba výstupu ve formě přepínače switch s 5 různými větvemi case. První větev dá na výstup původní neupravený signál. Druhá větev dá na výstup blokovanou část signálu. Třetí větev dá na výstup upravený signál pomocí beamformeru. Čtvrtá větev dá na výstup upravený signál pomocí beamformeru s použitím postfilteru. Pátá větev dá na výstup odhadnutý residuální šum na výstupu

- $OUTPUT = X$
- $OUTPUT = W * X$
- $OUTPUT = H * X$

- $OUTPUT = H * X * PF$
- $OUTPUT = H * AUX$

Po zvolení výstupu se provede zpětná Fourierova transformace a výsledný vektor je zařazen do kruhového bufferu `audioOutputMemory`. Tento kruhový buffer opět uchovává 4 poslední iterace podle metody overlap-add. Z tohoto bufferu vezmeme prvních 128 hodnot. Nad těmito hodnotami provedeme Interpolaci a aktualizujeme pomocnou proměnnou pro metodu Overlap-Add `ola_up`. Na následujícím obrázku je náhled na část kódu, který implementuje tuto funkcionalitu.

```
/*Upsampling*/
for (uword i = 0; i < FFTSHIFT; i++) {
    buff_up_L(3 * i) = x_out(i);
}
auout = conv(buff_up_L, LPF);
auout = auout + ola_up;

for (uword i = 0; i < FILTER_ORDER; i++) {
    ola_up(i) = auout(FRAMES_PER_BUFFER + i);
}
```

Obrázek 4.10: Ukázka interpolace v kódu

Výstup `auout` je posléze zesílen a poslán na výstup CallBack metody a zároveň uložen do bufferu pro ukládání nahrávek.

```
for (uword i = 0; i < FRAMES_PER_BUFFER; i++)
{
    vystup(i) = pomGain * auout(i);
}

for (i = 0; i < framesPerBuffer; i++)
{
    *out++ = vystup[i];
    *out++ = vystup[i];

    /*recording*/
    *wptr++ = vystup[i];  /* left */
    *wptr++ = vystup[i];  /* right */
}
```

Obrázek 4.11: Ukázka přiřazení dat na výstup v kódu

V poslední části metody `paCallBack` se ukládají hodnoty vstupu `X_2` a hodnoty `VAD` do kruhových bufferů a při nashromáždění dostatečného počtu hodnot, které byly vyhodnoceny jako obsahující hlas pomocí VAD, se zavolá metoda `getAsyncVADandRTF`, která probíhá asynchronně v jiném vlákně.

4.4.3 Metoda getAsyncVADandRTF

Tato metoda se volá po nashromáždění dostatečného počtu vzorků, které byly vyhodnoceny jako obsahující řeč. Vyhodnocení vzorků probíhá vždy po deseti rámcích, kteří tvoří tzv. podblok (angl. subblock). Jakmile se nashromáždí jeden podblok, jsou hodnoty jednotlivých rámců tvořící tento podblok sečteny a výsledná suma je porovnána s hodnotou `VADTresh`, která byla empiricky nastavená na 0.1, vynásobenou počtem efektivních frekvencí `NFREQS` a velikostí podbloku `subBlockSize`. Pokud jsou hodnoty VAD pro současný podblok vyhodnoceny jako obsahující řeč, tj. hodnota je vyšší než součin `VADThresh`, `NFREQS` a `subBlockSize`, jsou hodnoty VAD zařazeny do bufferu `VAD_batch` a hodnoty představující signál ve frekvenční doméně pro současný podblok do bufferu `X_batch`.

Počet hodnot v bufferech `VAD_batch` a `X_batch` potřebných k zavolání metody se liší podle stavu aplikace udávaném pomocí proměnné `initialStepIndex`, která je datový typ integer. Po úspěšném zavolání metody se hodnota `initialStepIndex` zvýší o 1. Při prvním spuštěním aplikace je tato proměnná nastavena na 0. Počet vzorků potřebných k jejímu zavolání je následující:

- `initialStepIndex` = 0, počet vzorků potřebných ke spuštění metody je 60
- `initialStepIndex` = 1, počet vzorků potřebných ke spuštění metody je 60
- `initialStepIndex` = 2, počet vzorků potřebných ke spuštění metody je 30
- `initialStepIndex` = 3, počet vzorků potřebných ke spuštění metody je minimálně 20

První tři stavy jsou nastaveny kvůli velikosti bufferů `X_batch` a `VAD_batch`, které mají velikost 150 prvků. Pro rychlejší naplnění těchto bufferů jsou proto použity

větší počty vzorků. Čtvrtý stav už volá metodu, když už jsou hodnoty v bufferech nahrazovány. Zároveň je pomocí proměnných typu bool zaručeno, že metody nebudou volány dříve, než skončí předchozí volání.

Zavolání metody proběhne v jiném vlákně, abychom dosáhli asynchronního výpočtu a nezpomalovali vykonávání metody `paCallback`. Při volání metody `getAsyncVADandRTF` se předávají parametry `nSubBatch`, udávající počet bloků v dávce, odkaz na `X_batch`, odkaz na `VAD_work` a hodnota udávající současný stav aplikace `initialStepIndex`. Ukázka volání metody `getAsyncVADandRTF` v jiném vlákně.

```
thread t1(getAsyncVADandRTF, 6, X_batch, VAD_work, 1);
t1.detach();
```

Obrázek 4.12: Spuštění funkce getAsyncVADandRTF v jiném vlákně

Na nově vytvořené vlákno zavoláme metodu `detach`, která zajistí, že současné vlákno nebude na něj čekat.

Uvnitř metody `getAsyncVADandRTF` se nejdříve spočítá VAD pro jednotlivé kanály na nových vzorcích. Pro výpočet zesílení se nejdříve zjistí energie na jednotlivých těchto vzorcích a odchylka oproti výchozí hodnotě podobně jako v rovnici 3.1. Zjistí se práh ticha pro jednotlivé vzorky na kanálu následovně

$$thr = silFact1 \times mean(Q); \tag{4.1}$$

Hodnoty `Q` se zfiltrují pomocí trojúhelníkové funkce a následně vynásobí proměnnou `VAD_AutoGain_innerGainFactor`, nastavenou empiricky na 0.75. Posléze se hodnoty, které nepřesahují vypočítaný práh ticha, utlumí pomocí `silAttFactor`, který byl empiricky nastaven na 0.1. Výpočet zesílení je znázorněn níže.

```
vec temp = { 0.25, 0.5, 1.0, 0.5, 0.25 };
mat e;
e = conv(Q.col(ch), temp);
e = 0.4 * e.rows(2, e.size() - 3);

mat AutoGain;
AutoGain = VAD_AutoGain_innerGainFactor * e;
AutoGain.elem(find(e < thr(0))) = AutoGain.elem(find(e < thr(0)))*silAttFactor;
```

Obrázek 4.13: Ukázka výpočtu automatického zesílení uvnitř VADu

Jakmile máme vypočítané zesílení, vypočítá se VAD pomocí mVAD popsané v 3.2.1. Ukázka implementace je na následujícím obrázku.

```
for (int frm = (ite - 1)*subBlockSize * nSubBatch; frm < ite*subBlockSize * nSubBatch - 1; frm++) {
    int index = frm - (ite - 1)*subBlockSize * nSubBatch;
    inputs = AutoGain(index) * abs(X_batch(span(frm, frm), span(ch, ch), span::all));

    mat layerIns;
    mat activated;

    activated.zeros(1, 2 * NFREQS);
    layerIns = trans((trans(inputs) - MEANin) % STDin) * W1 + B1;
    activated.elem((find(layerIns > 0.0))) = layerIns.elem(find(layerIns > 0.0));   // ReLU
    layerIns = activated * W2 + B2;
    activated.zeros(size(layerIns, 0), size(layerIns, 1));
    activated.elem((find(layerIns > 0.0))) = layerIns.elem(find(layerIns > 0.0));   // ReLU
    layerIns = activated * W3 + B3;
    activated.zeros(size(layerIns, 0), size(layerIns, 1));
    activated.elem((find(layerIns > 0.0))) = layerIns.elem(find(layerIns > 0.0));   // ReLU

    mat tempsl;
    tempsl = 1 / (ones(1, 2 * NFREQS) + exp(-(activated*W4 + B4)));
    matOut.col(0) = tempsl.t();

    VAD_batch_a(span(index, index), span(ch, ch), span::all) = matOut.rows(0, NFREQS - 1);
}
```

Obrázek 4.14: Ukázka výpočtu hodnot VADu

Dále se vypočítá medián hodnot VAD pro levý a pravý kanál přes jednotlivé frekvence a vzorky. Tento medián se přiřadí pro oba kanály. Jakmile máme hodnoty VAD je potřeba vypočítat relativní přenosové funkce. Toto proběhne pomocí

výpočtu uvedeného v rovnici 3.10, kde vzájemné hustoty výkonného spektra jsou upraveny odpovídajícími hodnotami VAD následovně

$$\begin{aligned} \phi_{LR}[m] &= \sum_{k=0}^{m \times subBlockSize} X[k,1]X[k,0]^H VAD[k,0] \\ \phi_{LL}[m] &= \sum_{k=0}^{m \times subBlockSize} X[k,0]X[k,0]^H VAD[k,0] \end{aligned} \tag{4.2}$$

kde $\phi_{LR}[m,0]$ představuje vzájemnou hustotu výkonného spektra mezi levým a pravým kanálem pro m-tý podblok, $X[k,0]$ je k-tý vzorek ve frekvenční oblasti na levém kanálu a $X[k,1]$ je k-tý vzorek na pravém kanálu.

4.4.4 Metoda používající výpočet pomocí IVE

Před spuštěním této metody je potřeba inicializovat jednotlivé proměnné na výchozí hodnotu. Inicializace proběhla obdobně jako u první metody pomocí funkce `initialize_var`, kde se nastavily proměnné na následující hodnoty.

- $\boldsymbol{w}(k) = [11]$, vektor pro zaměření zdroje nastavýme kolmo k poli mikrofonů
- $\boldsymbol{C}(k) = \boldsymbol{I_2}$, Autokovarianční matici se přiřadí jednotková matice o dimenzi 2
- $\xi(k) = 1$, pravděpodobnost cílového vektoru v signálu

Poté podobně jako v předchozí metodě se zavolá funkce `paCallback`, která nad signálem opět udělá základní kroky zpracování:

- Rozdělení vstupních dat na levý a pravý kanál
- Decimace signálu ze vzorkovací frekvence 48 kHz na 16 kHz
- Provedení krátkodobé Fourierovy transformace

Jakmile máme signál ve frekvenční doméně, můžeme začít se samotným výpočtem IVE. Zvolili jsme metodu, kdy výpočet probíhá nad posledními M vzorky, tudíž prvních M-1 vzorků se pouze přidá do kruhového bufferu a IVE se nad nimi

nepočítá. Jakmile přijde M-tá hodnota začne výpočet IVE vycházející z algoritmu popsaném v 3.21.

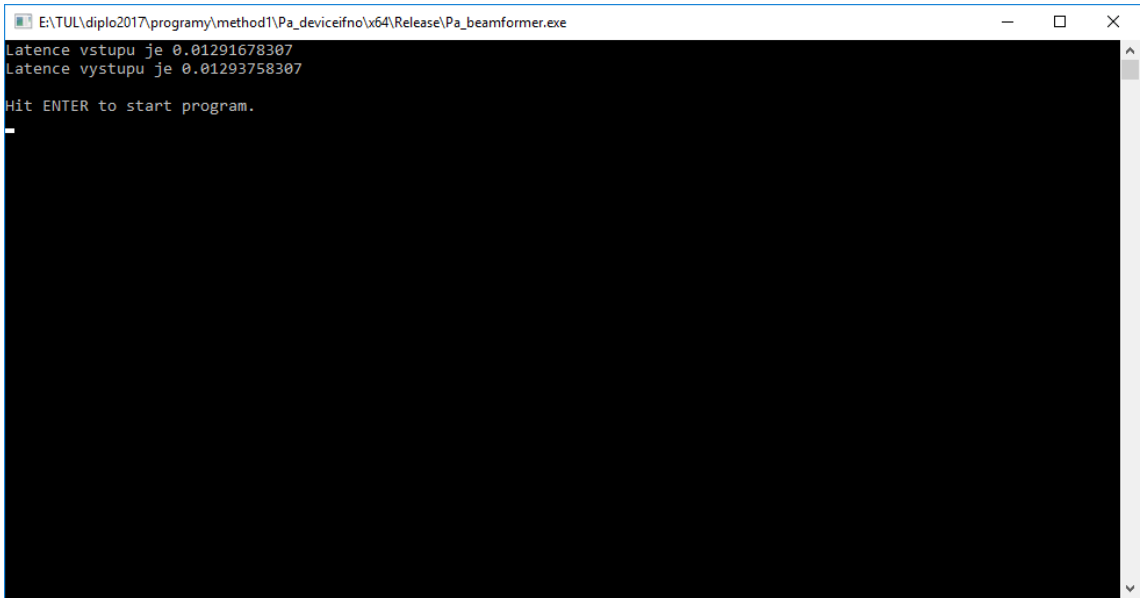
Odhadnutý cílový signál $\bar{s}_k[n]]$ je pak potřeba upravit pomocí směrového vektoru a následovně

$$y[k] = a_1[k]\bar{s}_k[n] \tag{4.3}$$

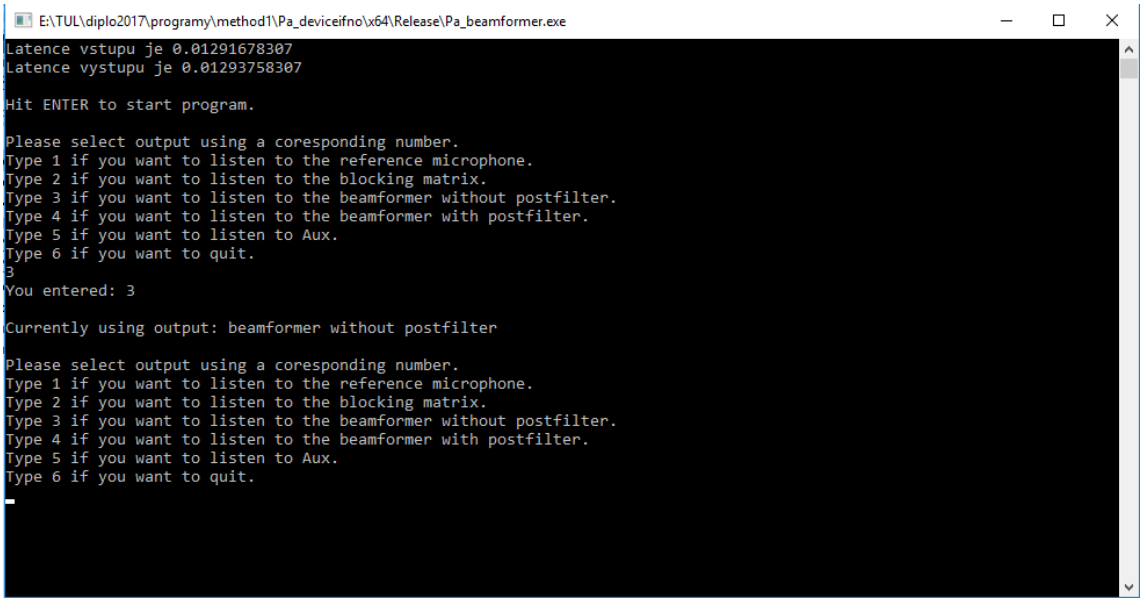
Nad vypočítaným výstupem se pak provede zpětná Fourierova transformace a interpolace na vzorkovací frekvenci 48 kHz.

4.4.5 Vzhled aplikace a ovládání

Při spuštění aplikace se nám v konzoli vypíšou odhady skutečné latence na vstupu a výstupu. Samotné zpracování zvuku spustíme pomocí klávesy enter. Při běhu aplikace je možné měnit výstupní signál nebo aplikaci ukončit zadáním čísla, odpovídajícího vybranému signálu z výpisu v konzoli, a stisknutím klávesy enter. Ukázka aplikace po spuštění je na obrázku 4.15 a ukázka volby signálu na obrázku 4.16.



Obrázek 4.15: Ukázka aplikace po spuštění



Obrázek 4.16: Ukázka možností volby výstupu

Pro účely testování se po ukočení aplikace automaticky ukládá výstupní signál do souboru s názvem recorded, který je typu raw, tedy neupravená data. Tento soubor se dá otevřít např. pomocí aplikace Audacity[3].

Kapitola 5

Testování a výsledky

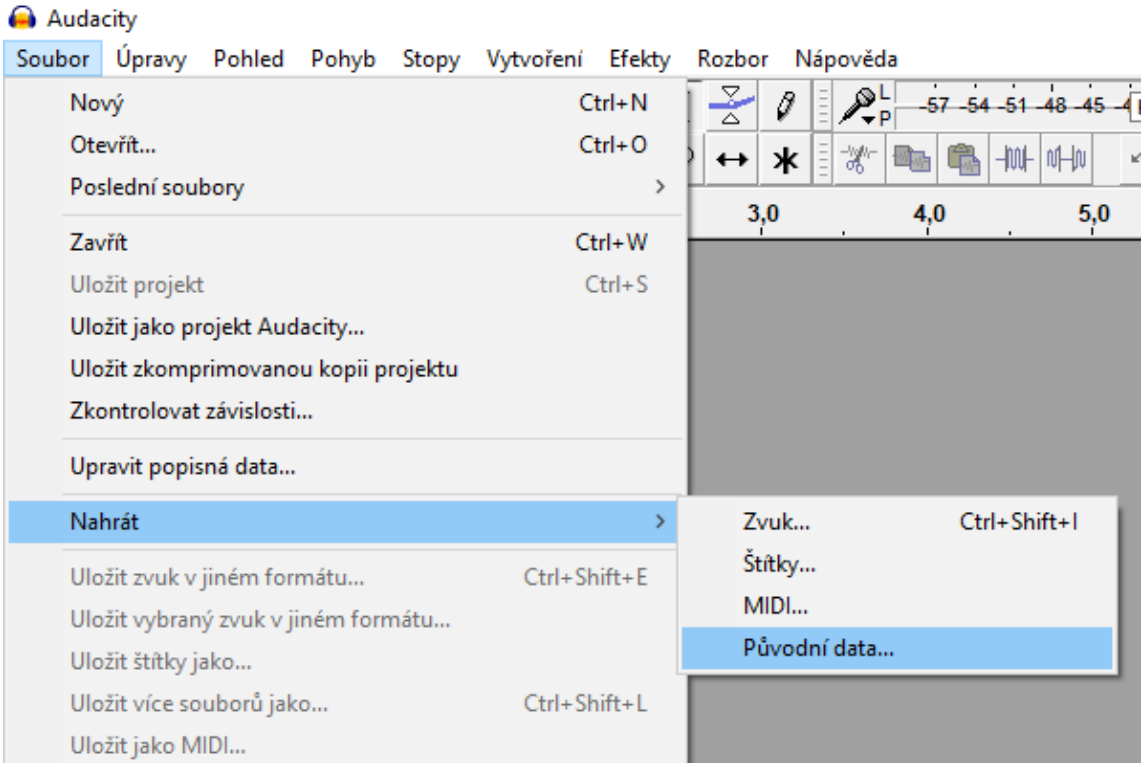
Pro testování aplikace byl použit počítač s operačním systémem Windows 10 a následujícím HW:

- 4 jádrový procesor Intel Core i5-4670k s rychlostí jádra 4.1 GHz
- 2 kanálový předzesilovač AudioBuddy
- 2 mikrofony typu The RØDE NTG1, které mají Hyperkardioidní směrovou charakteristiku

Testování probíhalo v uzavřené místnosti s minimálním okolním šumem. Pro testování bylo využito ovladače ASIO4ALL, který simuluje rozhraní ASIO na zařízeních, které nemají HW podporující toto rozhraní.

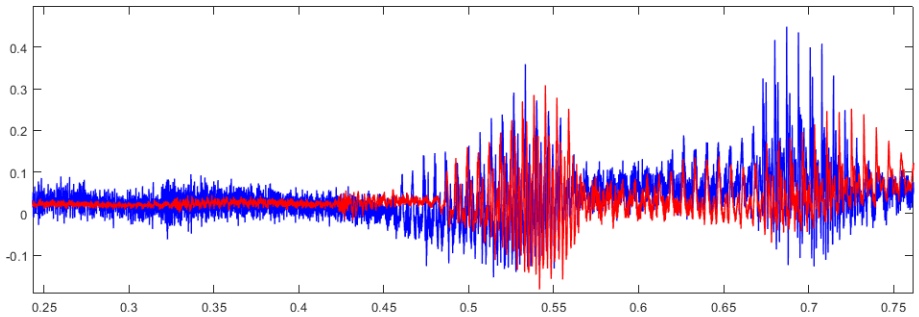
Ke kontrole výstupu byla použita aplikace Audacity[3], kde pomocí nahrání původních dat byly naimportovány data s následujícím nastavením:

- Kódování: 32-bit float
- Pořadí bytů: Bez endianness
- Kanály: 2
- Vzorkovací frekvence: 48 000 Hz



Obrázek 5.1: Nahrání dat v programu Audacity

Po nahrání se dají data uložit do libovolného formátu a prohlížet například v programu Matlab[4]. Ukázka porovnání dvou signálů před a po redukci šumu je na obrázku 5.2, kde modrý signál je původní zašuměný a červený je odšuměný pomocí metody s beamformerem.



Obrázek 5.2: Porovnání 2 signálů před a po redukci šumu

5.1 Metoda používající beamformer

5.1.1 Latence při zpracování

Doba zpracování signálu, tedy metoda `paCallback` byla měřena pomocí volání funkce `std::chrono::steady_clock::now()` na začátku a na konci této metody, po odečtení těchto hodnot jsme získali uplynulou dobu mezi těmito dvěma body. Z deseti spuštění aplikace a měření po dobu 1 minuty jsme získali následující minimální a maximální doby vykonávání metody `paCallback`.

- Maximální doba trvání: 1,689 ms
- Minimální doba trvání: 0,39 ms

Pomocí metody `Pa_GetStreamInfo`, která je součástí knihovny PortAudio, jsme získali odhady hodnot odezvev na vstupu a výstupu callback funkce. Latence streamu vycházela 20,2 ms, což je o 12,5 ms víc než doporučená hodnota systému. Pokud sečteme hodnoty na vstupu a výstupu společně s nejdelší dobou samotné callback metody dostáváme se na zpoždění okolo 42 ms. Toto je dosti vysoké, a tak jsem se pokusil snížit latenci vstupu a výstupu změnou parametru `suggestedLatency` na 0.

Změna parametrů se projevila a skutečná odezva se zmenšila na 12,9 ms jak pro vstup, tak pro výstup. Sečtením těchto hodnot a zahrnutím maximální doby trvání zpracování signálu jsme se dostali na 27,5 ms. Další parametr, který by mohl zmenšit odezvu systému, je buffer offset, který je součástí nastavení ovladače ASIO4ALL. Tento parametr má výchozí hodnotu 4 ms. Bohužel při snížení tohoto parametru docházelo k výpadkům a zrnění na výstupu, způsobeným nedostatkem času, ke zpracování dat. Hodnotu už dál nešlo snížit jelikož zbylých 8,9 ms pocházelo převážně z velikosti bufferu (počet vzorku vůči vzorkovací frekvenci) $384/48000 = 0,008ms$ a zbytek je zpoždění operačního systému.

5.1.2 Efektivita redukce šumu

Bylo vyzkoušeno několik situací a na nich vyhodnocena efektivita metody.

- Zdroj: Řečník Šum: Odrazy místnosti
- Zdroj: Řečník Šum: Hudba
- Zdroj: Řečník Šum: Druhý řečník

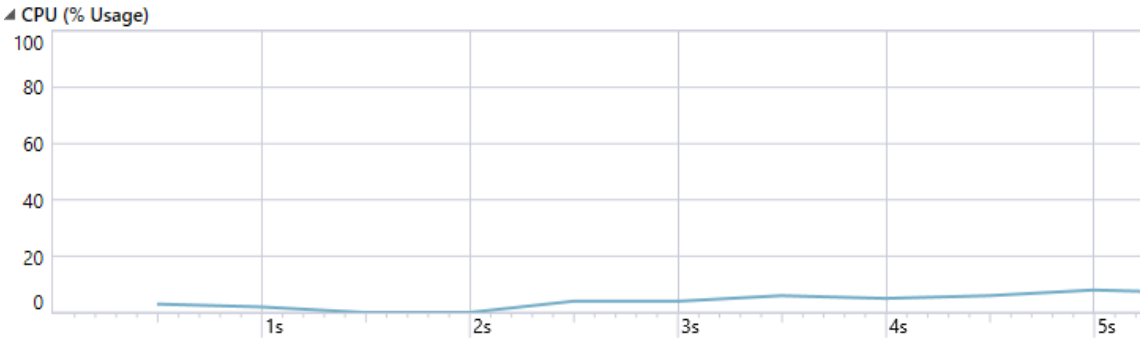
V prvním případě se metoda ověřila jako efektivní. Při poslechu blokované části signálu se vyskytovaly odrazy místnosti a zvuk větráků počítačové skříně, když se poslouchal signál upravený pomocí beamformeru byl slyšet pouze hlas řečníka.

V druhém případě se podařilo hudbu potlačit, samozřejmě ideální by bylo hudbu eliminovat, ale ve skutečném prostředí v to doufat nemůžeme. Při poslechu blokované části signálu bylo poznat, že se blokuje i část hlasu (odrazy + nedokonalost metody). Při přerušení řeči a opětovném započetí na jiné pozici se metoda dobře adaptovala a pokračovala v redukci hudby.

V třetím případě jsem vyzkoušel jak metoda zareaguje na změnu řečníka, tj. střídal jsem chvíle, kdy mluvili oba řečníci s chvíli, kdy mluvil jenom jeden, aby proběhlo jeho zaměření. Toto se osvědčilo jako možné, pokud byli oba řečníci ve stejné vzdálenosti. Pokud byl jeden řečník blíž k mikrofonům, tak se metoda zaměřila na něj.

5.1.3 Výpočetní náročnost

Výpočetní náročnost této metody byla díky využití knihovny OpenBLAS výrazně snížena (oproti výchozí knihovně BLAS, která je součástí knihovny Armadillo). V průměru se pohybovala okolo 7 % celkového využití procesoru, jak ukazuje graf na obrázku 5.3.



Obrázek 5.3: Ukázka využití procesoru naší aplikací pomocí Profileru ve Visual Studiu

5.2 Metoda používající IVE

5.2.1 Latence při zpracování

U této metody probíhal výpočet zpoždění obdobným způsobem jako u předešlé metody. Opět měřením po dobu 1 minuty a deseti opakováních jsme získali následující minimální a maximální doby trvání metody.

- Maximální doba trvání: 3,288 ms
- Minimální doba trvání: 0,8 ms

Jak vyplývá z hodnot výše, tato metoda je pomalejší na výpočet než metoda používající k výpočtu beamformer. Nicméně 3 ms jsou pořád dostačující dobra zpracování metody a po přičtení zpoždění na vstupu a výstupu (toto zpoždění je pro obě metody stejné) získáme zpoždění 29,1 ms.

5.2.2 Efektivita redukce šumu

Jaku o první metody bylo vyzkoušeno několik situací pro vyhodnocení efektivity metody.

- Zdroj: Řečník Šum: Odrazy místnosti
- Zdroj: Řečník Šum: Hudba

- Zdroj: Řečník Šum: Druhý řečník (nahrávka řeči puštěná z reproduktoru)

V prvním případě metoda nedokázala zachytit šum v podobě tikání hodin a větráku počítače. Výsledný signál byl tedy celkově potlačen, takže zde nedošlo k velkému zlepšení. Pravděpodobně kvůli odlišným frekvencím šumu a hlasu, se oba vyhodnotili jako zdrojové.

V druhém případě už jsme dosahovali lepších výsledků a to zejména při nízkých úrovní šumu. Jak se zvyšovala úroveň šumu (hudby), tím se zhoršovala míra potlačení. Toto se dá zdůvodnit nedokonalostí metody a špatnému odhadu cílového vektoru.

V třetím případě jsem jako další hlas pro testování použil nahrávky z Open Speech Repository, což je databáze nahrávek zdarma k použití. V tomto případě opět metoda nefungovala, tak jak by měla. Nepodařilo se správně zaměřit cílového řečníka a tím potlačit druhého řečníka.

5.2.3 Výpočetní náročnost

Výpočetní náročnost této metody byla nižší díky jednoduššímu výpočtu a jelikož v této metodě nebylo potřeba provádět asynchronní výpočty. V průměru se výpočtní náročnost této metody pohybovala okolo 4 % celkového zatížení procesoru.

Kapitola 6

Závěr

Na začátku práce jsme se seznámili s problematikou zpracování zvuku, zejména pak potlačením šumu. Ukázali jsme si metody odhadu šumu pomocí porovnání vícekanálových signálů. Přiblížili jsme si evaluační kampaň CHiME-4, s jakými daty tato kampaň pracovala, jaké byly její cíle a způsoby řešení. V menší míře jsme se seznámili s evaluační kampaní CHiME-5. Navrhli jsme způsob řešení v reálném čase vycházející z jednoho řešení této kampaně. Toto řešení vycházelo z principu porovnání 2 signálů obsahující cílovou řeč a následnou synchronizací se dosáhlo odhadnutí šumu na jednotlivých kanálech a zaměřením zdroje cílové řeči, tzv. beamforming. Ke správné funkci, ale bylo zapotřebí vědět, že se v daný okamžik na těchto kanálech řeč vyskytuje. Pro tyto účely jsme se seznámili s detektorem aktivity řeči, tzv. VAD.

Poté přišlo na řadu vytvořit aplikaci, která by prováděla samotné zlepšení nahrávek a beamforming v reálném čase. K realizaci jsme použili programovací jazyk C++ a vývojové prostředí Microsoft Visual Studio. K nahrávání jsme použili pole dvou mikrofonů a software ASIO4ALL pro uplatnění zvukového rozhraní ASIO na počítači bez podporovaného hardwaru. K naprogramování aplikace jsme využili knihovny Armadillo, pro zpracování lineární algebry, SigPack pro úpravu signálů, FFTW3 pro výpočet rychlé Fourierovy transformace a v neposlední řadě knihovnu PortAudio, která umožňuje přehledně zpracovávat zvuk v reálném čase. K rychlému zpracování dat jsme použili tzv. callback metodu, volanou jakmile se na vstupu

nashromáždí dostatečný počet dat. Metodu jsme otestovali za přítomnosti různých druhů šumu. Tato metoda fungovala velmi dobře ve všech případech.

V další části jsme implementovali metodu založenou na extrakci nezávislých vektorů, která je výpočetně jednodušší než předešlá metoda. Udělali jsme několik experimentů v závislosti na poloze řečníka a různých druhů šumu. Bohužel tato metoda v reálných podmínkách nefungovala tak, jak v simulovaných. Metoda fungovala za přítomnosti hudby s nízkou úrovní hlasitosti. Se stoupající hlasitostí postupně začala ztrácet zaměření cílového řečníka, a tím i potlačení šumu. Možným řešením problému špatného zaměření cílového řečníka by byla implementace algoritmu pro kontrolu konvergence metody. Současná implementace spoléhá pouze na počáteční nastavení a nekontroluje konvergenci metody.

V poslední části jsme se zaměřili na optimalizaci obou implementovaných metod. Dokázali jsme snížit vstupní a výstupní odezvu na 12,9 ms a celkovou odezvu jednotlivých aplikací do 30 ms. Výpočetní náročnost obou systémů se po použití knihovny OpenBLAS dostala pod 10 % vytížení procesoru.

Literatura

[1] Emmanuel Vincent, Shinji Watanabe, Aditya Arie Nugraha, Jon Barker, Ricard Marxer. An analysis of environment, microphone and data simulation mismatches in robust speech recognition. *Computer Speech and Language* [online], Elsevier, 2016. <hal-01399180> Dostupné z: https://hal.inria.fr/hal-01399180

[2] Xavier Anguera, Chuck Wooters and Javier Hernando, Acoustic beamforming for speaker diarization of meetings, *IEEE Transactions on Audio, Speech and Language Processing*, volume 15, number 7, pp.2011-2023, 2007 Dostupné z: http://www.xavieranguera.com/papers/transactions_taslp_2007.pdf

[3] Audacity(R) software is copyright (c) 1999-2017 Audacity Team. [Web site: http://audacity.sourceforge.net/. It is free software distributed under the terms of the GNU General Public License.] The name Audacity(R) is a registered trademark of Dominic Mazzoni.

[4] MATLAB Release 2016a, The MathWorks, Inc., Natick, Massachusetts, United States.

[5] Voice Activity Detection (VAD) Tutorial. Practical Cryptography [online]. [cit. 2018-05-07]. Dostupné z: http://practicalcryptography.com/miscellaneous/machine-learning/voice-activity-detection-vad-tutorial/

[6] Koldovský, Zbyněk, Jiri Malek and Marek Boháč. “CHiME4: Multichannel Enhancement Using Beamforming Driven by DNN-based Voice Activity Detection.” (2016).

[7] S. Gannot, D. Burshtein, and E. Weinstein, “Signal enhancement using beamforming and nonstationarity with applications to speech,” IEEE Transactions on Signal Processing, vol. 49, no. 8, pp. 1614–1626, Aug 2001.

[8] The 4th CHiME Speech Separation and Recognition Challenge [online]. [cit. 2018-05-04]. Dostupné z: http://spandh.dcs.shef.ac.uk/chime_challenge/chime2016/overview.html

[9] The 5th CHiME Speech Separation and Recognition Challenge. Speech and Hearing [online]. [cit. 2018-05-04]. Dostupné z: http://spandh.dcs.shef.ac.uk/chime_challenge/overview.html

[10] PortAudio - an Open-Source Cross-Platform Audio API [online]. [cit. 2018-04-25]. Dostupné z: http://www.portaudio.com/ J. W. Demmel. Applied Numerical Linear Algebra. SIAM, 1997.

[11] Conrad Sanderson and Ryan Curtin. Armadillo: a template-based C++ library for linear algebra. Journal of Open Source Software, Vol. 1, pp. 26, 2016.

[12] Wang Qian, Zhang Xianyi, Zhang Yunquan, Qing Yi, AUGEM: Automatically Generate High Performance Dense Linear Algebra Kernels on x86 CPUs, In the International Conference for High Performance Computing, Networking, Storage and Analysis (SC’13), Denver CO, November 2013.

[13] SigPack - the C++ signal processing library [online]. [cit. 2018-05-04]. Dostupné z: http://sigpack.sourceforge.net/index.html

[14] ASIO4ALL - Universal ASIO Driver For WDM Audio [online]. [cit. 2018-05-04]. Dostupné z: http://www.asio4all.org/index.html

[15] KOLDOVSKÝ, Zbyněk a Petr TICHAVSKÝ. Gradient Algorithms for Complex Non-Gaussian Independent Component/Vector Extraction. 2018. 1803.10108.

[16] Using an auditory-inspired representation for speech | seaandsailor [online]. 2014 [cit. 2018-05-10]. Dostupné z: http://www.seaandsailor.com/gammatone.html

[17] Matteo Frigo and Steven G. Johnson, "The Design and Implementation of FFTW3," Proceedings of the IEEE 93 (2), 216–231 (2005). Invited paper, Special Issue on Program Generation, Optimization, and Platform Adaptation

[18] Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

[19] H. L. Van Trees, Optimum array Processing: Part IV of Detection, Estimation, and Modulation Theory, John Wiley & Sons, Inc., 2002.

Dodatek A

Obsah přiloženého CD

- text diplomové práce ve formě pdf - diplomova_prace_2018_Bartos_Petr.pdf
- zdrojové kódy implementací:
  - Pa_beamformer
  - Pa_ive
- spustitelné aplikace s potřebnými knihovnami
- ukázkové nahrávky výstupů aplikací