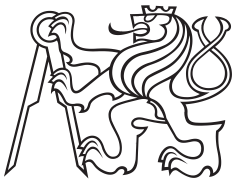
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Zpracování signálu z mikrofonních polí s digitálními MEMS mikrofony

Martin Helmich

Vedoucí: Ing. Petr Honzík, Ph.D.
Studijní program: Softwarové inženýrství a technologie
Specializace: Technologie pro multimédia a virtuální realitu
Květen 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Helmich** Jméno: **Martin** Osobní číslo: **510654**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

Specializace: **Technologie pro multimédia a virtuální realitu**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Zpracování signálu z mikrofonních polí s digitálními MEMS mikrofony

Název bakalářské práce anglicky:

Processing of signals from microphone arrays with digital MEMS microphones

Pokyny pro vypracování:

Seznamte se s principy mikrofonních polí s lineární a cirkulární geometrií. Na dostupných prototypech mikrofonních polí ověřte přenos digitálních signálů do hostitelského počítače pomocí dostupných HW prostředků. Zkoumejte možnosti zpracování přijatých signálů z hlediska směrovosti mikrofonního pole. Navrhněte a implementujte zpracování signálů z mikrofonního pole realizující směrový systém a zkoumejte jeho možné praktické aplikace (například detekce směru příchodu signálu). Výsledek otestujte na datové sadě reálných naměřených signálů.

Seznam doporučené literatury:

[1] Jacob Benesty, Jingdong Chen, and Israel Cohen. Design of Circular Differential Microphone Arrays. Springer Cham, Leden 2015. ISBN 978-3-319-14842-7.
[2] Elisabet Tiana Roig. Eigenbeamforming array systems for sound source localization. PhD thesis, Technical University of Denmark, Listopad 2014.
[3] Boris Clénet. Circular microphone array based beamforming and source localization on reconfigurable hardware, Září 2010. Master thesis, Graz University of Technology.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Petr Honzík, Ph.D. katedra radioelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **01.02.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

Ing. Petr Honzík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

Poděkování

Rád bych poděkoval Ing. Petru Honzíkovi, Ph. D. za vedení, trpělovost a pravidelné konzultace během semestru, zejména v oblasti teorie zpracování signálu. Dále děkuji Ing. Janu Šedivému za ochotu a pomoc při práci s mikrofonním polem. Také bych rád poděkoval své rodině a přítelkyni za podporu během studia a při psaní bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

Praha, 22. května, 2024

Abstrakt

Práce se zabývá analýzou a zpracováním signálu z prototypu cirkulárního mikrofonního pole s digitálními MEMS mikrofony, vyvinutého Davidem Vagnerem a Janem Šedivým v rámci jejich diplomových prací. Konkrétně se zaměřuje na techniky Delay and Sum Beamforming a diferenciální mikrofonní pole prvního řádu. Zprvu byla provedena analýza směrovosti zmíněného prototypu, která byla ověřena na naměřených datech z bezodrazové komory. Dále byl vyvinut algoritmus pro lokalizaci nejsilnějšího zdroje zvuku v prostoru za použití zmíněných technik, který byl otestovaný v audiovizuálním studiu Katedry radioelektroniky Fakulty elektrotechnické ČVUT v Praze. Práce uvádí možnosti dalšího vývoje.

Klíčová slova: delay and sum beamforming, diferenciální mikrofonní pole, Python

Vedoucí: Ing. Petr Honzík, Ph.D.

Abstract

This thesis focuses on the analysis and processing of signals from a prototype of circular microphone array with digital MEMS microphones developed by David Vagner and Jan Šedivý as part of their master's theses. Specifically, it addresses the techniques of Delay and Sum Beamforming and first-order differential microphone array. Initially, an analysis of the directivity of the mentioned prototype was conducted, which was validated using measured data from an anechoic chamber. Furthermore, an algorithm for locating the strongest sound source in the space using the mentioned techniques was developed and tested in the audiovisual studio of the Department of Radioelectronics, Faculty of Electrical Engineering, CTU in Prague. The thesis concludes with possibilities for further development.

Keywords: delay and sum beamforming, differential microphone array, Python

Title translation: Processing of signals from microphone arrays with digital MEMS microphones

Obsah

Obrázky

Úvod

Tato práce navazuje na diplomové práce Davida Vagnera [1] a Jana Šedivého [2]. V rámci jejich prací byl vyvinutý prototyp cirkulárního mikrofonního pole, který přes FPGA desku posílá výstupní audio sinál z jednotlivých mikrofonů do uživatelského počítače. Cílem této práce je nalézt praktické využití pro zmíněný prototyp a implementovat vhodný algoritmus pro zpracování jeho výstupního signálu.

V rámci analýzy prototypu padla pozornost na techniky pro ovlivňování směrovosti mikrofonního pole, tj. zesilování či zeslabování zvukových vln přicházejících z různých směrů. To nám v praxi dává například možnost částečné izolace mluvčího v místnosti od nežádoucích ozvěn, šumů a dalších rušivých elementů. Požadavky na taktovéto vlastnosti mikrofonu či soustavy mikrofonů vznikají například při použití v konferenčních místnostech, které buď nejsou dobře akusticky ošetřeny, anebo v nich je hodně různých menších nežádoucích zdrojů zvuku.

Pro dosažení tohoto cíle pak bylo využito technik sumačního a diferenciálního mikrofonního pole. V úvodu byla provedna analýza toho, jak se prototyp cirkulárního mikrofonního pole bude chovat při zpracování jeho výstupu těmito technikami. Následně byla provedena jejich implementace pro zpracování signálu v reálném čase. Jako nadstavba byl pak vyvinut a implementován algoritmus, který pomocí stejných technik dokáže lokalizovat nejsilnější zdroj zvuku ve svém okolí. Výstupem této práce je pak sada skriptů, které slouží jako Proof of Concept (PoF) pro budoucí vývoj softwarové aplikace, která dokáže nalézt směr nejsilnějšího zdroje zvuku v prostoru a následně ho pomocí zkoumaných technik částečně odizolovat od nežádoucích okolních zvuků.

Část I

Teoretická část

Kapitola 1

Techniky pro zpracování signálu z mikrofonních polí

1.1 Mikrofonní pole

Jako mikrofonní pole se označuje jakákoliv soustava dvou a více mikrofonů, které jsou umístěné v různých bodech v prostoru [3]. Jedním z hlavních využití mikrofonních polí je provádění takzvaného beamformingu, který spočívá v zaměřování signálu přicházejícího z námi určeného směru. Analogicky je pak naším cílem zeslabit veškerý signál, který přichází z ostatních nežádoucích směrů. Tato technika se v praxi využívá zejména v telekomunikacích na zaměření radiových vln, nicméně lze jí využívat i pro mikrofonní pole na zaměření zvukových vln.

Mikrofonní pole lze rozlišovat podle mnoha parametrů, podle kterých je pak můžeme rozdělovat do různých kategorií. Mezi jeden takovýto parametr patří pak rozdělení dle geometrického uspořádání jednotlivých mikrofonů. Zde rozlišujeme 3 základní kategorie - pole lineární, cirkulární a sférická. Každá z geometrií má svojí unikátní prostorovou charakteristiku, se kterou se pojí její výhody i nevýhody. Z matematického pohledu se pak liší zejména ve způsobu výpočtu zpoždění na i-tém mikrofonu. To je většinou vyjádřeno vzhledem k mikrofonu, na který dopadne zvuková vlna jako první. Tomuto bodu říkáme referenční bod.

Dalším parametrem je pak rozdělení z hlediska zpracování signálu. V tomto kontextu rozlišujeme mezi poli sumačními a diferenciálními. Jak vyplývá z názvů, tak v případě pole sumačního budou výstupní signály z mikrofonů sčítány a v případě diferenciálního budou signály od sebe odčítány. Tato práce se bude zabývat jak sumační, tak diferenciální technikou zpracování signálu. Mikrofonní pole pak bude využito k lokalizaci a izolaci zdroje zvuku pomocí techniky zvané beamforming. Konkrétně bude rozebráno diferenciální

mikrofonní pole prvního řádu a Delay and Sum Beamforming. Tyto metody patří mezi nejjednodušší a nejstarší techniky beamformignu [4].

V reálném prostředí se zvuk obvykle šíří formou kulové vlny, avšak pro zjednodušení výpočtů budeme v této práci počítat s tím, že se všechny vlny šíří rovninným způsobem. Pro veškeré výpočty bylo počítáno s rychlostí zvuku $c = 343$ m/s.

1.2 Delay and Sum Beamforming

Cílem techniky Delay and Sum Beamforming (DSB) je zesílení signálu přicházejícího z námi určeného směru. Tím zároveň dojde k zeslabení signálů přicházejících ze všech ostatních směrů. Největšího využití DSB nachází v oblasti radiokomunikace, kde je mikrofonní pole reprezentováno anténami.

1.2.1 Obecná definice DSB

Jak bylo zmíněno v předchozí kapitole, tak jedním z podstatných parametrů pro beamforming je zpoždění τ_{m_i}, za které signál dorazí na i-tý mikrofon. Pro jeho výpočet zaveďme vektor $\hat{\kappa}$, který reprezentuje souřadnice zaměřovaného signálu, a vektor $\mathbf{r_{m_i}}$, který reprezentuje souřadnice i-tého mikrofonu v prostoru. Analogicky tak počítáme zpoždění vzhledem ke středu souřadnic (tj. středu mikrofonního pole), který v tuto chvíli slouží jako referenční bod. Po výpočtu skalárního součinu těchto 2 vektorů získáme vzdálenost, kterou vlna musí urazit k i-tému mikrofonu. Tuto vzdálenost pak převedeme na čas tím, že ji vydělíme rychlostí zvuku c. Matematicky lze zpoždění vyjádřit vzorcem [4]

$$\tau_{m_i}(\hat{\kappa}) = \frac{\hat{\kappa} \cdot \mathbf{r_{m_i}}}{c}. \tag{1.1}$$

Vzhledem k námi zvolenému referenčnímu bodu ovšem vyjde polovina zpoždění záporných, jelikož na ně signál dorazí dřív než na referenční bod. To je v praxi neaplikovatelné, jelikož nelze aplikovat negativní zpoždění na mikrofon. Abychom se tohoto efektu zbavili, tak zvolme jako nový referenční bod nejbližší mikrofon ke zvukovému zdroji, který označme jako m_c a jeho zpoždění $\tau_{m_c}(\hat{\kappa})$. Zpoždění $\tau_{m_c m_i}$ mezi mikrofony m_c a m_i pak jednoduše spočítáme vzorcem

$$\tau_{m_c m_i}(\hat{\kappa}) = \tau_{m_c}(\hat{\kappa}) - \tau_{m_i}(\hat{\kappa}). \tag{1.2}$$

Z toho plyne, že v praktické implementaci musíme počkat, než vlna dopadne na mikrofon, který je nejvzdálenější od zdroje zvuku. Se znalostí zpoždění na i-tém mikrofonu můžeme již napsat rovnici DSB v časové doméně, která je formulována vzorcem

$$b(t, \hat{\kappa}) = \sum_{i=0}^{M} w_{m_i} p_{m_i}(t - \tau_{m_c m_i}(\hat{\kappa})), \tag{1.3}$$

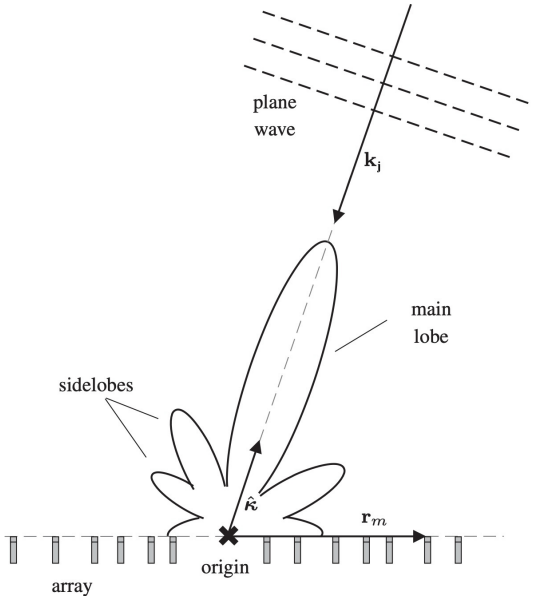
kde w_{m_i} je amplitudové váhování, p_{m_i} je tlak naměřený na i-tém mikrofonu a t je diskrétní časový index. Vzoreček je převzat z [4].

Při analýze jakéhokoliv mikrofonního pole nás zajímá jeho graf směrovosti, tzv. beampattern. Ten je zpravidla uváděn ve frekvenční doméně, jelikož je proměnlivý pro každou frekvenci. Beampattern je počítán tak, že se změří amplituda pro vlny přicházející ze směrů $\mathbf{k_j}$, zatímco zpoždění na mikrofonech m_i jsou nastavená ve směru $\hat{\kappa}$. Vypočítané amplitudy pro konkrétní frekvenci a $\mathbf{k_j}$ pak tvoří funkci směrovosti. Výpočet amplitudy ze směru $\mathbf{k_j}$ je dán vzorcem

$$D_{DSB}(j\omega, \mathbf{k_j}) = \sum_{i=1}^{M} e^{j\omega(\tau_{m_c m_i}(\hat{\kappa}) - \tau_{m_c m_i}(\mathbf{k_j}))}, \tag{1.4}$$

kde j je komplexní jednotka $j = \sqrt{-1}$ a ω je úhlová rychlost přímo úměrná své frekvenci f, kde $\omega = 2\pi f$. Vzoreček byl převzat z [5]. Příklad grafického znázornění grafu směrovosti je dostupný na obrázku 1.1.

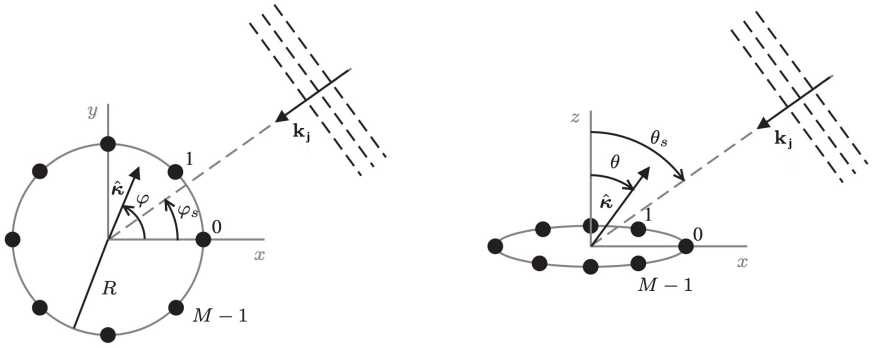
Při analýze grafu směovosti si můžeme všimnout, že v některých směrech je signál silnější a v jiných je zase slabší. Směry, ve kterých signál dosahuje svého lokálního maxima, označme jako laloky. Laloku, ve kterém vzniká globální maximum, říkejme hlavní lalok (main lobe) a všem ostatním říkejme postranní laloky (side lobes), viz obrázek 1.1. Můžeme si všimnout, že hlavní lalok vzniká vždy ve směru $\hat{\kappa}$. Druhou částí beamformingu je pak co největší zeslabení všech postraních laloků. Tato technika se obecně označuje jako Side-lobe cancelling, v této práce se jí ovšem již zabývat nebudeme.



Obrázek 1.1: Grafické znázornění grafu směrovosti pro lineární mikrofonní pole. Na obrázku je vyznačený hlavní lalok (main lobe) a postranní laloky (side lobes). Převzato a upraveno z práce [4].

1.2.2 DSB pro cirkulární mikrofonní pole

Cirkulární mikrofonní pole (CMA) předpokládá, že se všech M mikrofonů nachází na kružnici o poloměru R. Grafické znázornění je vidět na obrázku 1.2.



(a) Top view (b) Side view

Obrázek 1.2: Grafické znázornění geometrie cirkulárního mikrofonního pole. Převzato a upraveno z práce [4].

Jako φ označme úhel, pod kterém leží vektor $\hat{\kappa}$ v rovině xy a jako φ_s označme takový úhel, pod kterým leží vektor $\mathbf{k_j}$. To stejné platí pro úhly θ a θ_s v rovině xz. Díky znalosti úhlů φ a θ dokážeme určit vektor $\hat{\kappa}$ a pozici mikrofonů r_{m_i} následujícím způsobem [4]

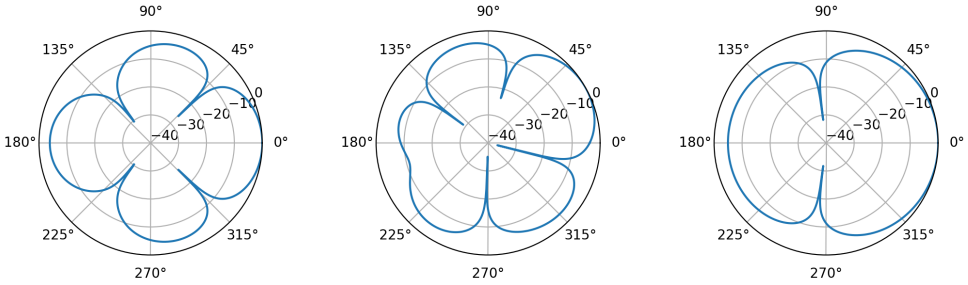
$$\hat{\kappa} = \begin{bmatrix} \sin\theta\cos\varphi \\ \sin\theta\sin\varphi \\ \cos\theta \end{bmatrix}, \tag{1.5}$$

$$\mathbf{r_{m_i}} = R \begin{bmatrix} \cos 2\pi i/M \\ \sin 2\pi i/M \\ 0 \end{bmatrix}. \tag{1.6}$$

Pro výpočet grafu směrovosti potřebujeme znát vektor $\mathbf{k_j}$, který jednoduše získáme výměnou úhlů θ za θ_s a φ za φ_s ze vzorce pro výpočet $\hat{\kappa}$ 1.5. Matematicky lze $\mathbf{k_j}$ vyjádřit jako

$$\mathbf{k_j} = \begin{bmatrix} \sin\theta_s\cos\varphi_s \\ \sin\theta_s\sin\varphi_s \\ \cos\theta_s \end{bmatrix}. \tag{1.7}$$

Příklady grafů směrovosti pro cirkulární mikrofonní pole jsou vidět na obrázcích 1.3. Kód pro vygenerování grafů je dostupný v příloze C.



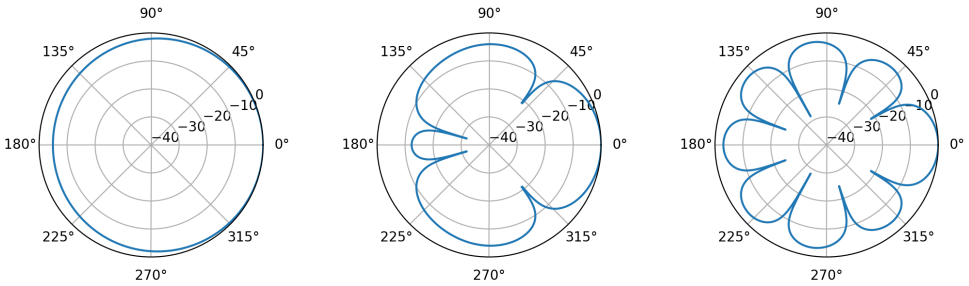
(a) : $\varphi = 0°$ a $\theta = 90°$ **(b) :** $\varphi = 30°$ a $\theta = 90°$ **(c) :** $\varphi = 0°$ a $\theta = 30°$

Obrázek 1.3: Ukázka grafů směrovosti pro cirkulární mikrofonní pole s 8 mikrofony, poloměrem 5 cm a frekvencí 3,5 kHz. Vyjádřeno v decibelech.

Směrovost cirkulárního mikrofonního pole se liší v případech, kdy je vektor $\hat{\kappa}$ natočen mezi polohy mikrofonů $\mathbf{r_{m_i}}$ a $\mathbf{r_{m_{i+1}}}$. Tato změna je vidět mezi grafy 1.3a a 1.3b, kde se mikrofon m_0 nachází pod úhlem 0° a m_1 pod úhlem 45°. V tomto případě se jeden z postranních laloků dokonce rozdělil na 2 menší. Z toho plyne, že pro snadně předvídatelnou směrovost je vhodné mít mikrofonní pole o co nejvíce mikrofonech, jinak budou vznikat takovéto nežádoucí artefakty.

Směrovost cirkulárního mikrofonního pole zároveň ovlivňuje jeho poloměr R, jak můžeme vidět na obrázku 1.4. Na obrázku 1.4a můžeme pozorovat,

že směrovost není téměř žádná (nevyskytují se zde žádné postranní laloky), zatímco na 1.4c je počet postranních laloků hned 7. Při návrhu cirkulárního mikrofonního pole je proto potřeba přizpůsobit poloměr R tak, aby pole mělo adekvátní směrovou charakteristiku pro požadované frekvence.

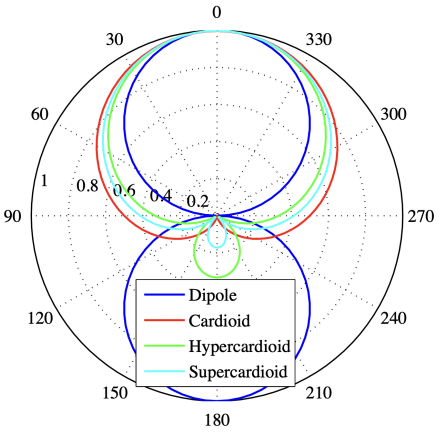


(a) : $R = 5$ cm **(b) :** $R = 15$ cm **(c) :** $R = 25$ cm

Obrázek 1.4: Ukázka grafů směrovosti pro cirkulární mikrofonní pole s 8 mikrofony pro různé poloměry R, kde $\varphi = 0°$ $\theta = 30°$ a $f = 2$ kHz. Vyjádřeno v decibelech.

1.3 Diferenciální mikrofonní pole prvního řádu

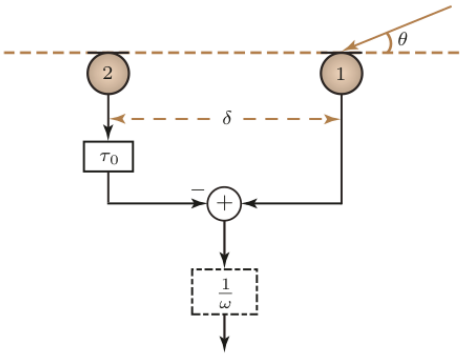
Narozdíl od sumačního mikrofonního pole, princip diferenciálního mikrofonního pole spočívá v odčítání výstupních signálů od sebe z jednotlivých mikrofonů. Za speciální případ diferenciálního mikrofonního pole (DMA) budeme považovat DMA prvního řádu, které se skládá ze 2 mikrofonů. Jeho pomocí lze modelovat 4 základní směrové charakteristiky, jako je dipól, kardioida, superkaridoida a hyperkardioida (viz 1.5). Tato práce se bude zaměřovat na tvorbu kardioidní charakteristiky, která disponuje schopností efektivně potlačovat signály přicházející zezadu. [6]



Obrázek 1.5: Možné směrové charakteristiky diferenciálního pole prvního řádu. Převzato z práce [6].

1.3.1 Tvorba kardioidní směrové charakteristiky

Princip tvorby kardioidní směrové charakteristiky spočívá v odečtení signálu zpožděného o čas, který odpovídá šíření vlny od jednoho mikrofonu k druhému. Implementační schéma je vidět na obrázku 1.6.



Obrázek 1.6: Implementační schéma DMA prvního řádu pro tvorbu kardioidní směrové charakteristiky ve frekvenční doméně. Převzato z práce [7].

Označme d jako šikmou vzdálenost, kterou vlna musí urazit mezi 2 mikrofony pod úhlem θ. Zpoožděnı́ mezi mikrofony pak můžeme analogicky vypočítat pomocí vzorce

$$\tau = \frac{d}{c} = \frac{\delta \cos\theta}{c}, \tag{1.8}$$

kde c je rychlost zvuku a δ je vdálenost mezi 2 mikrofony (viz obrázek 1.6). Pro vyjádření výstupu kardioidy ve frekvenční doméně nejdříve uvažujme harmonickou vlnu dopadající na mikrofon 1 o úhlové frekvenci $\omega = 2\pi f$ jako

$$s_1(t) = p_1 e^{j\omega t}, \tag{1.9}$$

kde p_1 je amplituda. Pro vyjádření signálu $s_2(t)$ z mikrofonu 2 musíme započítat zpoždění τ, se kterým na něj budou zvukové vlny přicházet vzhledem k mikrofonu 1. Pro vyjádření tohoto zpoždění musíme signál $s_1(t)$ vynásobit exponenciálou s negativním znaménkem a započteným zpožděním τ. Výsledný zápis bude vypadat následovně

$$s_2(t) = e^{-j\omega\tau} p_1 e^{j\omega t}. \tag{1.10}$$

Po započtení zpoždění τ_0 podle schématu 1.6 dostáváme zpožděný signál

$$s_{2,delayed}(t) = e^{-j\omega\tau}e^{j\omega(t-\tau_0)}, \tag{1.11}$$

kde τ_0 je zpoždění pro úhel $\theta = 0$. Po odečtení signálů získáváme výstup

$$U(\theta,\omega) = s_1(t) - s_{2,delayed}(t) = p_1e^{j\omega t} - p_1e^{-j\omega\tau}e^{j\omega(t-\tau_0)}. \tag{1.12}$$

V kompaktnější formě lze pak výstup pomocí ekvivalentních úprav zapsat jako

$$U(\theta,\omega) = p_1e^{j\omega t}(1 - e^{\frac{\delta}{c}\cos\theta+\tau_0}). \tag{1.13}$$

Vzhledem k tomu, že zpoždění na mikrofonu 2 má záporné znaménko v expomentu, tak v reálné implementaci budeme muset zpožďovat celou soustavu o zpoždění τ_0, jelikož není možné přistupovat ke vzorkům před tím, než byli pořízeny. Z toho důvodu budeme v časové doméně zpoždění τ_0 ve výsledku zpožďovat signál z prvního mikrofonu.

Přenosová funkce a graf směrovosti

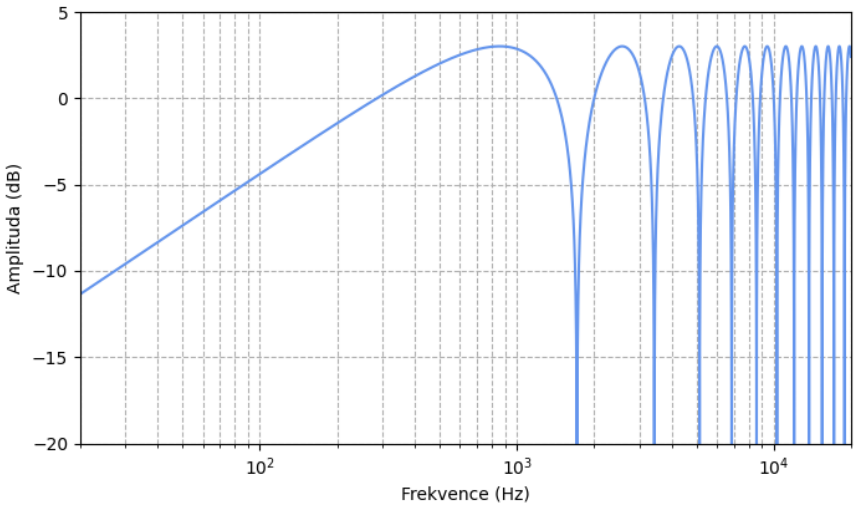
Výpočet grafu směrovosti a přenosové funkce lze pak získat pomocí vzorců [8]

$$D(\theta,\omega) = \frac{U(\theta)}{U(0)} = \frac{1 - e^{\frac{\delta}{c}\cos\theta+\tau_0}}{1 - e^{\frac{\delta}{c}+\tau_0}}, \tag{1.14}$$

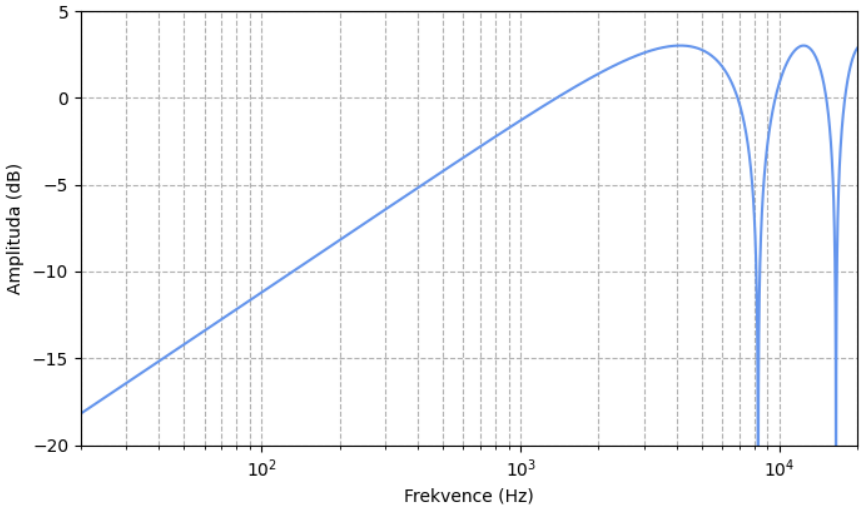
$$TF(\theta,\omega) = 1 - e^{\frac{\delta}{c}\cos\theta+\tau_0}. \tag{1.15}$$

Přenosová funkce nám udává hlasitost výstupu mikrofonního pole proměnlivou na různých frekvencích. Ukázku přenosové funkce nabízí grafy na obrázcích 1.7 a 1.8. Z toho můžeme pozorovat, že hlasitost výstupu je na nízkých frekvencích velmi nízká a s rostoucí frekvencí se zvyšuje. Tento efekt je možné

kompenzovat integrátorem $\frac{1}{\omega}$, tato operace ovšem bude vést k současnému zvýšení hladiny šumu. Zároveň si můžeme všimnout výskytu míst, na kterých je hlasitost v decibelech do nekonečna. Na těchto místech se vlny od sebe navzájem odečtou, jelikož se zde jejich vlnové délky rovají polovině vzdálenosti mezi mikrofony δ. Mikrofonní pole má pak zhruba kardioidní charakteristiku na frekvencích do výskytu prvního nekonečna ve směru $\theta = 0°$. Jak můžeme vyčíst z grafu 1.7, tak ideální vzdálenost δ mezi mikrofony by měla být pro účely tvorby karidioidy co nejmenší, jelikož rozšiřuje frekvenční spektrum před vytvořením prvního nekonečna. Na obrázku 1.8 můžeme pozorovat, jak přenosová funkce bude vypadat z úhlu $\theta = 90°$.

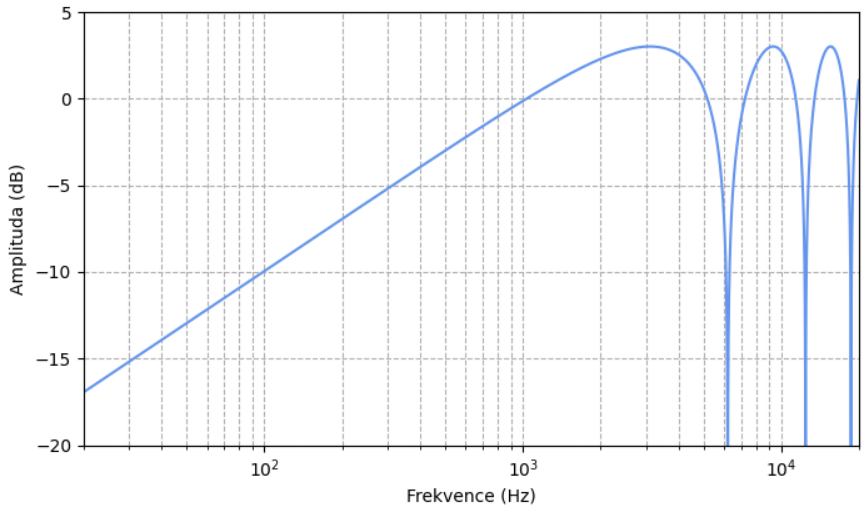


(a) : $\delta = 10$ cm



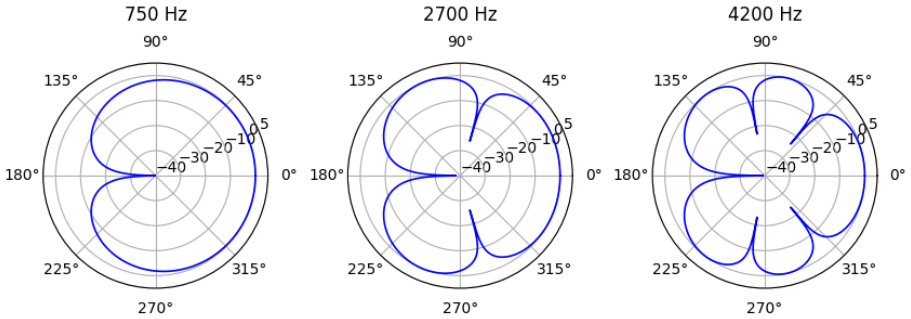
(b) : $\delta = 2$ cm

Obrázek 1.7: Příklady přenosových funkcí pro různé vzdálenosti δ mezi mikrofony a úhlech dopadu vlny $\theta = 0°$. Na ose x je vyobrazena frekvence (Hz) a na ose y amplituda (dB).



Obrázek 1.8: Příklad přenosové funkce DMA prvního řádu pro úhel $\theta = 90°$a $\delta = 10$ cm. Na ose x je vyobrazena frekvence (Hz) a na ose y amplituda (dB).

Vedle přenosové funkce můžeme zároveň sestavit grafy směrovosti, které jsou vyobrazené na obrázku 1.9. Z grafů můžeme pozorovat, že s vyšší frekvencí narůstá počet postranních laloků. Ten je určen tím, v jakém bloku určeného nekonečny přenosové funkce se frekvence nachází. To můžeme vyčíst z grafu 1.7a



Obrázek 1.9: Porovnání grafů směrovosti pro kardioidu DMA prvního řádu o frekvencích 750, 2700 a 4200 Hz a rozestupu mezi mikrofony $\delta = 10$ cm. Vyjádřeno v decibelech.

1.4 Zpožďování digitálního signálu

1.4.1 Zpožďování o celé vzorky

Digitální signál je v počítači ukládán jako pole čísel (vzorků), které reprezentují amplitudu signálu v konkrétním čase t. Množství provedených vzorků za

jednu sekundu určuje vzorkovací frekvence, která standardně bývá v hodnotách 44.1 kHz nebo 48 kHz.

Nejjednodušší způsob, jak zpožďovat digitální signál, je jeho posunutí o n vzorků v čase dozadu. Z toho plyne, že signál můžeme zpožďovat pouze o násobky času, který trvá mezi pořízením 2 vzorků. To vede k zaokrouhlení t_q, které lze získat jako

$$t_q = \frac{1}{f_{vz}}, \tag{1.16}$$

kde f_{vz} je vzorkovací frekvence. Pro vzorkovací frekvence 44.1 kHz toto zaokrouhlení vychází zhruba na 22.68 mikrosekund a pro frekvenci 48 kHz na $20.8\overline{3}$ mikrosekund. Ve valné většině případů je toto zaokrouhlení tak malé, že je snadno zanedbatelné. V případě mikrofonních polí jsou ovšem zpoždění na mikrofonech tak malá, že takto na první pohled malé zaokrouhlení může výrazně změnit jeho výslednou směrovost. Toto zaokrouhlení lze obejít zpožďováním signálu pomocí lineární interpolace, které je sice značně výpočetně náročnější, ovšem zato přesnější.

1.4.2 Lineární interpolace

Jak bylo zmíněno v předchozí kapitole, tak při zpožďování signálu o celé vzorky dochází k zaokrouhlení, které nemusí být v případě mikrofonních polí žádoucí. Pro provedení přesnějšího zpožďování v praxi signál obvykle nejdříve zpozdíme celý počet vzorků a zbytek doladíme lineární interpolací. Lineární interpolaci lze totiž provádět pouze mezi 2 vzorky.

Řekněme, že máme signál $y(n)$ a zpoždění η v intervalu $\eta \in \langle 0, 1\rangle$. η lze pak chápat jako bod v čase mezi dvěma vzorky, mezi kterými chceme signál interpolovat. Zpožděný signál $\hat{y}(n+\eta)$ lze pak vyjádřit jako [9]

$$\hat{y}(n+\eta) = (1-\eta)y(n) + \eta y(n+1). \tag{1.17}$$

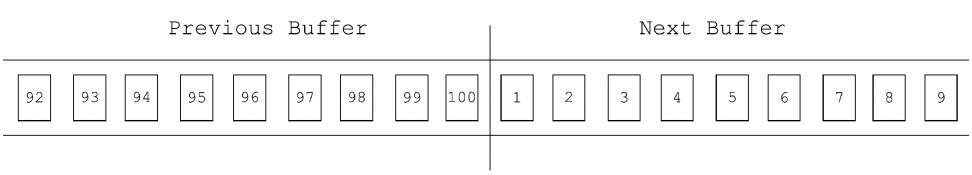
Je však třeba pamatovat, že zpoždění pomocí lineární interpolace je výpočetně náročná operace, která může omezit výkon naší aplikace. Používáme jí proto pouze v nezbytných případech.

1.4.3 Zpožďování v reálném čase

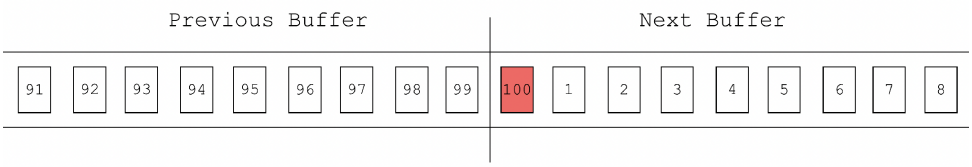
Digitální signály v reálném čase jsou obvykle zpracovávány za použití bufferu, který slouží jako dočasné úložiště umožňující efektivní manipulaci se signálem. Hlavním parametrem audio bufferu je jeho délka, která je definována počtem načtených vzorků. K obnovení bufferu dochází v momentě, kdy systém načte dostatečný počet vzorků nezbytných k naplnění nového bufferu. V této fázi je původní audio buffer vyprázdněn (tj. obvykle přesměrován do výstupního zařízení nebo uložen do souboru) a ihned poté je načten nový buffer, čímž se celý cyklus opakuje.

Zpracování signálu musí tak proběhnout v časovém intervalu, který odpovídá době potřebné k načtení dalšího bufferu. Například při vzorkovací frekvenci 48 kHz a velikosti bufferu 4800 vzorků má systém přibližně 0.1 sekundy na realizaci všech potřebných operací zpracování signálu.

Pokud chceme ovšem signál zpožďovat v reálném čase, tak nastává problém, jelikož pro posunutí o n vzorků potřebujeme znát koncové vzorky předchozího bufferu. Z tohoto důvodu si musíme alokovat další blok paměti, ve kterém si data z předchozího bufferu budeme přechodně ukládat. V Pythonu lze tento problém například řešit globální proměnnou. Vizualizace je dostupná na obrázku 1.10.



(a) : Signál bez aplikovaného zpoždění



(b) : Signál s aplikovaným zpoždění

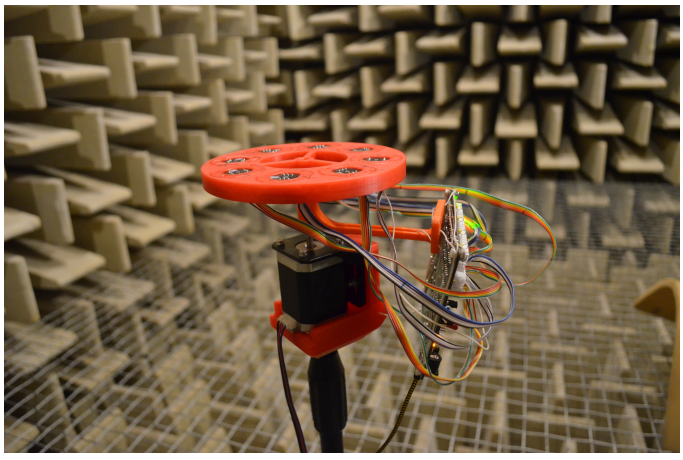
Obrázek 1.10: Vizualizace zpožďování signálu v reálném čase za využití bufferu o velikosti 100 vzorků. Na 1.10a je vidět tok vzorků bez aplikovaného zpoždění a na 1.10b je vidět přetékající poslední vzorek předchozího bufferu do dalšího.

Kapitola 2

Použité technologie a HW

2.1 Prototyp cirkulárního mikrofonního pole z FEL ČVUT

V rámci práce [1] byl Davidem Vagnerem vytvořen prototyp cirkulárního mikrofonního pole, se kterým bylo pracováno v rámci této práce (viz obrázek 2.1). Prototyp byl se skládá z 8 MEMS mikrofonů, které jsou rozmístěné po kružnici o poloměru 5 cm. Konstrukce prototypu byla vyrobena pomocí 3D tisku. Data z mikrofonů jsou vyváděna pomocí I2S sběrnice do FPGA desky, která byla vyrobena Janem Šedivým v rámci práce [2]. FPGA deska komunikuje s počítačem pomocí USB kabelu, ve kterém se chová jako class complient zvuková karta. Takto přijímaná data lze pak dále zpracovávat například pomocí DAW nebo individuálními Python skripty.



Obrázek 2.1: Ukázka prototypu cirkulárního mikrofonního pole vyvinutého Davidem Vagnerem v rámci práce [1] spolu s FPGA deskou vyvinutou Janem Šedivým v rámci práce [2] na FEL ČVUT.

2.2 Python

Pro implementaci DSB algoritmu a provádění veškerých ostatních analýz v této práci byl zvolen jazyk Python 3. Python je interpretovaný jazyk, jehož nejrozšířenější implementace CPython je napsaná převážně v jazyce C. Tento jazyk byl vyhodnocen jako vhodná volba pro svojí jednoduchost, dostupné knihovny a rozšířenost v IT komunitě. Podle Stack Overflow survey z roku 2023 byl vyhodnocen jako 3. nejpoužívanější jazyk na světě s daty o 87 585 respondentech z oblasti IT [10]. V oblasti zpracování signálu se hodí spíše na prototypování a analýzu, než na konečné komerční implementace v reálném čase, kvůli svým horším vlastnostem v oblasti výkonu, jak uvádí zdroj [11].

Pro práci byli využity externí knihovny Matplotlib [12], SoundDevice [13], NumPy [14] a SciPy [15]. Matplotlib poskytuje nezbytné nástroje pro tvorbu 2D a 3D grafů, které byli použity zejména pro zobrazení a analýzu grafů směrovosti. Sounddevice byl použitý pro své funkce na zpracování audia v reálném čase a NumPy pro matematické operace a práci s audio signálem, který je reprezentovaný jako `NumPy.ndarray`.

2.2.1 Distribuované výpočty v Pythonu

Distribuování výpočtů aplikace mezi více vláken či procesů často vede k zvýšení výkonu, jelikož tím počítači dáváme možnost efektivně využít více jader svého procesoru (CPU). Problém ovšem nastává tehdy, kdy se více vláken snaží přistupovat ke sdíleným datům ve stejnou chvíli. To může vést ke korupci dat zvlášť ve chvíli, co se například 2 vlákna snaží upravovat sdílená data najednou. Popsaná situace se odborně nazývá race condition. Tento problém se řeší zavedením zámku, který rozhoduje o tom, které z vláken může v jednu chvíli ke sdíleným datům přistupovat. Zabezpečení aplikace proti race-condition je klíčovým krokem k docílení mezivláknové bezpečnosti (thread-safety).

Vzhledem k tomu, že implementace CPython nemá bezpečně implementovanou mezivláknovou komunikaci, došlo v počátcích vývoje jazyka Python k zavedení globálního interpretačního zámku (GIL). Ten funguje tak, že pokud chce nějaké vlákno vykonávat svůj bajtkód, tak si nejdříve musí obstarat globální interpretační zámek, a po jeho dokončení ho zase uvolnit dalším vláknům. Díky tomuto mechnismu je aktivní vždycky pouze jedno vlákno v jednu chvíli a eliminuje se tak možnost výskytu race-conditions. Z toho ovšem plyne, že jakákoliv mezivláknová aplikace implementovaná v Pythonu se na výstupu chová jako jednovláknová. Tento zabezpečovací mechanismus tak omezuje možnost paralelizace výpočtů na úrovni procesoru pomocí vláken a z tohoto důvodu je není možné používat pro zvýšení výkonu (například jejich

implementaci modulem `threading`). [16]

Pro skutečné paralelizování výpočtů v Pythonu je tak potřeba používat procesy. Tím se nám podaří obejít globální interpretační zámek, jelikož každý proces spustí novou instanci Python interpreteru s vlastním globálním interpretačním zámkem. Tento způsob nám zajistí efektivní paralelizování náročných výpočtů mezi více jader procesoru. Toto je možné učinit například pomocí knihovny `multiprocessing`, která nabízí nástroje pro efektivní správu procesů včetně meziprocesové komunikace. [17]

Knihovna multiprocessing

Knihovna `multiprocessing` nabízí účiné nástroje pro inicializovaní procesů a správu sdílené paměti. Proces je možné spustit pomocí `multiprocessing.Process` a pro sdílení dat mezi procesy pak slouží implementace datových struktur, mezi které mimo jiné patří například fronta (`multiprocessing.Queue`), pole (`multiprocessing.Array`) nebo sdílená paměť (`multiprocessing.Shared_memory`). Knihovna nabízí zároveň vlastní implementaci zámku (`multiprocessing.Lock`) a události pro komunikaci mezi různými procesy (`multiprocessing.Event`).

Stěžejní funkcí knihovny `multiprocessing` je pak třída `multiprocessing.Pool`, která nabízí inteligentní distribuování výpočtů předepsané funkce do určitého množství uživatelem definovaných procesů. Díky této funkci uživateli odpadá zodpovědnost za správu procesů, o které se postará `multiprocessing.Pool`.

2.2.2 NumPy

Stěžejní roli při vývoji aplikace na lokalizaci nejsilnějšího zdroje zvuku hrála knihovna NumPy [14]. Středobodem této knihovny je pak datová struktura `numpy.ndarray`, která nabízí implementaci vícerozměrného pole. Nad těmito poli lze pak provádět vektorizované operace, které jsou implementované přímo v jazyce C, což podstatně snižuje jejich čas výpočtu oproti provádění operací pomocí Python cyklů. Velké rychlosti těchto operací například napomáhá i to, že jsou tato pole implementována jako kontinuální bloky paměti, což značně omezuje množství paměťových skoků, jako je to například v základní Python implementaci pole `List`.

Vzhledem k těmto vlastnostem se `numpy.ndarray` hodí na reprezentaci vícekanálového digitálního audio signálu, který je přednostně upravován pomocí vektorizovaných operací v NumPy.

Část II

Praktická část

Kapitola 3

Analýza prototypu cirkulárního mikrofoního pole z FEL ČVUT

V této kapitole byl zanalyzován prototyp cirkulárního mikrofonního pole, který byl vytvořen v rámci prací [1] a [2]. V nich bylo zároveň provedeno měření, jehož data byla zanalyzována a vyhodnocena v kapitole 3.4. Cílem této analýzy je zjistit, zda je prototyp použitelný pro dostatečnou izolaci zvukových vln z nežádoucích směrů, a zda ho lze případně použít pro algoritmus na lokalizaci nejsilnějšího zdroje zvuku.

Analýza byla provedena pro 2 techniky zpracování signálu, Delay and sum beamforming a diferenciální mikrofonní pole prvního řádu. Pro obě techniky byla analyzovaná směrovost, která byla ověřena na naměřených datech z bezodrazové komory.

3.1 Struktura kódu pro analýzu

Pro analýzu mikrofonního pole bylo využito jazyka Python 3 rozšířeného primárně o knihovny NumPy [14] a Matplotlib [12]. Pro zajištění co nejmenšího kopírování stejného kódu po různých skriptech se veškeré podstatné funkce nachází v souboru `ComputationFunctions.py`. Tento soubor je importován do většiny skriptů na tvorbu diagramů či jiných potřebných výpočtů. Importování bylo prováděno na počítači s operačním systémem Linux. Na jiných operačních systémech bude nejspíše nutné upravit syntaxi importů v hlavičkách skriptů, aby odpovídala syntaxi daného operačního systému. Veškerý zdrojový kód je dostupný v příloze C.

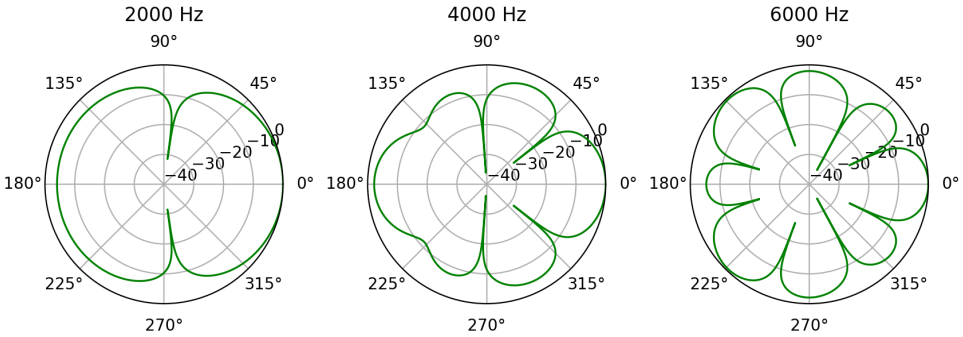
Důležitým parametrem veškerých funkcí na výpočet grafů směrovosti je `resolution`. Tento parametr určuje, kolik hodnot na ose x bylo vypočítáno

pro sestavení polárních grafů.

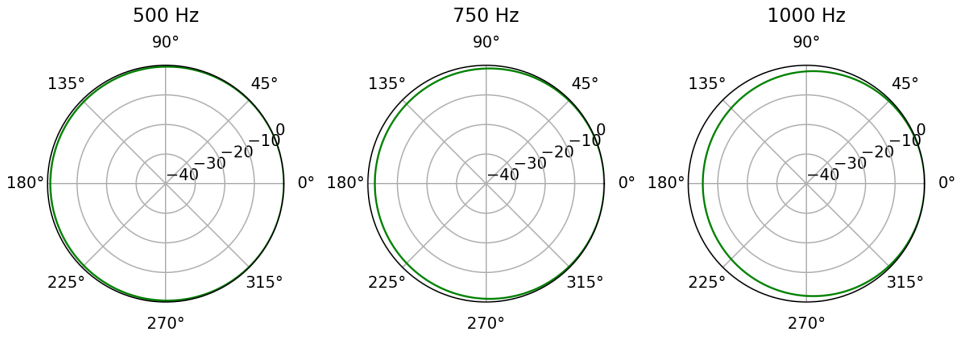
3.2 Delay and Sum beamforming

3.2.1 Směrová charakteristika

Hlavním bodem analýzy prototypu cirkulárního mikrofonního pole je výpočet jeho směrovost. Beampattern byl spočítaný podle vzorce 1.4 pro $\mathbf{k_j}$ s parametry $\theta_s = 90°$ a $\varphi_s \in \langle 0, 2\pi)$. Počet vektorů $\mathbf{k_j}$ je v kódu určen parametrem `resolution`. Každý takto získaný úhel φ_s pak reprezentuje směr jemu přiřazeného vektoru $\mathbf{k_j}$. Hodnoty $D_{DSB}(j\omega, \mathbf{k_j})$ jsou následně ukládány do pole a zobrazeny v polárním grafu pomocí knihovny Matplotlib [12]. Na obrázku 3.1 je vidět, jak vypadá graf směrovosti pro frekvence 2000, 4000 a 6000 Hz a `resolution` = 2000.

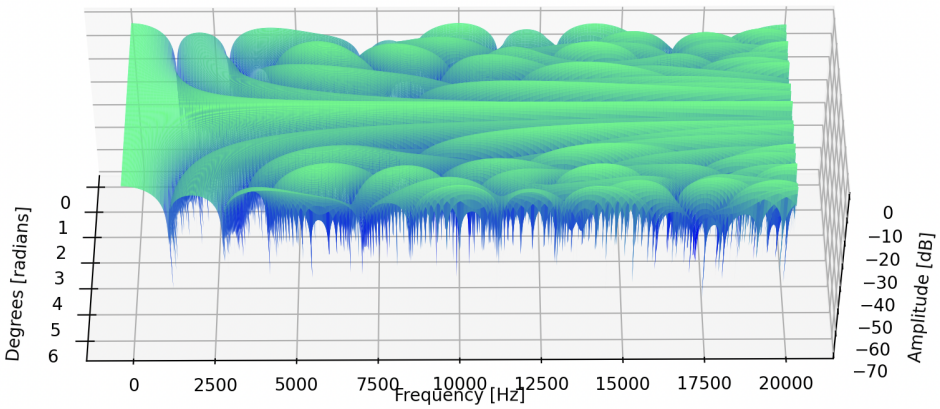


Obrázek 3.1: Grafy směrovosti pro CMA o parametrech M = 8, $\theta = 90$, $\varphi = 0$ a radius = 5 cm. Grafy jsou sestrojené pro frekvence 2000, 4000 a 6000 Hz. Vyjádřeno v decibelech.



Obrázek 3.2: Grafy směrovosti pro CMA o parametrech M = 8, $\theta = 90$, $\varphi = 0$ a radius = 5 cm. Grafy jsou sestrojené pro frekvence 500, 750 a 1000 Hz. Vyjádřeno v decibelech.

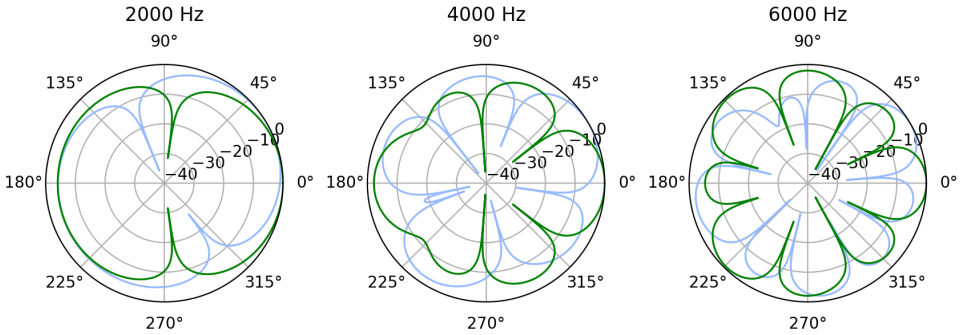
Pro získání lepšího přehledu o směrovosti prototypu cirkulárního mikrofonního pole napříč frekvencemi byl vytvořen 3D graf, který má na ose x úhly dopadu φ_s, na ose y frekvence od 20 do 20000 Hz a na ose z amplitudu v decibelech. Graf je vidět na obrázku 3.3.



Obrázek 3.3: 3D graf směrovosti protoypu církulárního mikrofonního pole napříč frekvencemi 20-20000 Hz. Směr vektoru $\hat{\kappa}$ odpovídá $\varphi = 180°$.

Z grafu 3.3 je vidět, že momentální implementace směrovosti DSB algoritmu má nejlepší směrovost v rozmezí zhruba 1.5 až 10 kHz. Pro frekvence nižší než 1 kHz směrovost nemá téměř žádnou ve více jak 10 kHz se tvoří mnoho postranních laloků. Pro lepší představu toho, jak vypadá směrovost na frekvencích nižších než 1 kHz, jsou tyto frekvence vyobrazeny v grafu 3.2

Dosavaď byli testované případy, kdy byl vektor $\hat{\kappa}$ nastaven stejným směrem, kterým leží nějaký z mikrofonů m_i (například $\varphi = 0°$). Dále bylo ostestováno, jak se směrovost bude chovat v případech, že $\hat{\kappa}$ bude nastavená jiným směrem, než kterým leží mikrofon m_i. Na obrázku 3.4 je vidět, jak směrovost vypadá pro $\varphi = 30°$ v porovnání s $\varphi = 0°$ na frekvencích 2000, 4000 a 6000 Hz.



Obrázek 3.4: Graf směrovosti pro $\hat{\kappa}$, kde $\varphi = 0°$ (zeleně) a $\varphi = 30°$ (světle modře).

Jak je vidět z obrázku 3.4, tak v místech, ve kterých neleží mikrofony m_i, vznikají artefakty v grafu směrovosti. Tento problém lze řešit pouze přidáním dalších mikrofonů do mikrofonního pole, jak je zmíněno v kapitole 1.2.2.

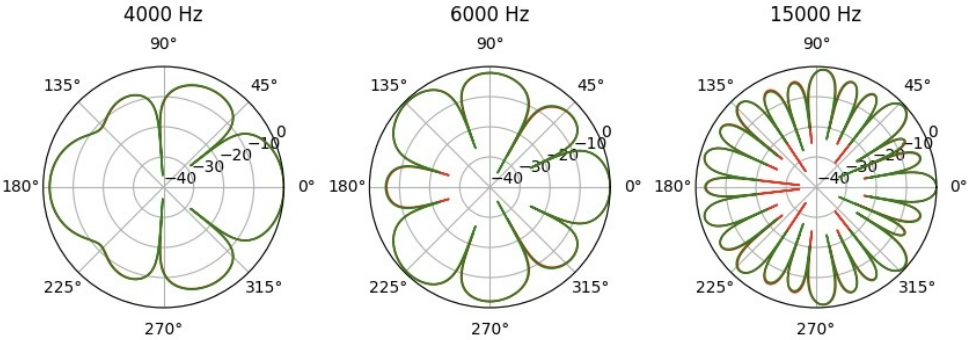
Pomocí Python skriptů uvedených v příloze C lze grafy ze všech obrázků v této kapitole vygenerovat pro jakoukoliv frekvenci či jiné parametry.

3.2.2 Kvantizace zpoždění

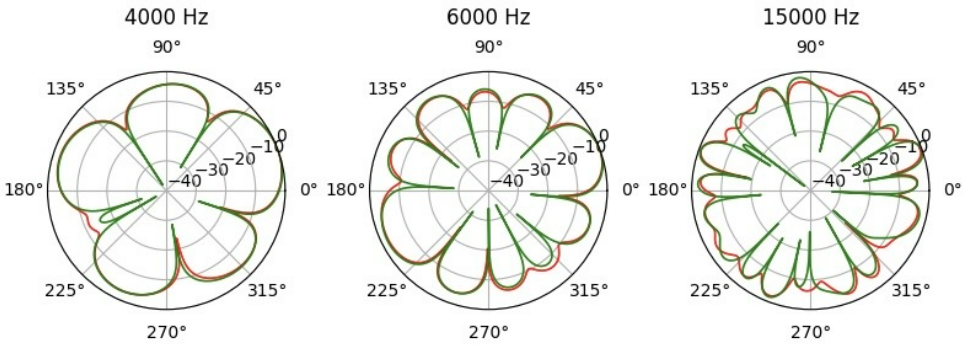
Vzorkovací frekvence MEMS mikrofonů je 48 kHz [1], což znamená, že zpožďováním o celé vzrorky lze signál zpožďovat o násobky $\frac{1}{48000}$ sekund nebo $20.8\overline{3}$ mikrosekund. Vzhledem k tomu, že se zpoždění na mikrofonech m_i pohybuje v desítkách mikrosekund, tak by to mohlo způsobovat problém v nepřesnosti, případně úplné nefunkčnosti DSB algoritmu. Pro zjištění vlivu zpoždění na směrovost prototypu cirkulárního mikrofonního pole byli v této kapitole sestaveny grafy směrovosti, na kterých bylo zaokrouhlení aplikováno.

Grafy směrovosti byli sestaveny pro různé frekvence a natočení úhlu φ. Na obrázcích 3.5, 3.6 a 3.7 jsou vyobrazeny grafy pro frekvence 4, 6 a 15 kHz a úhly $\varphi = 0, 20, 30$ stupňů.

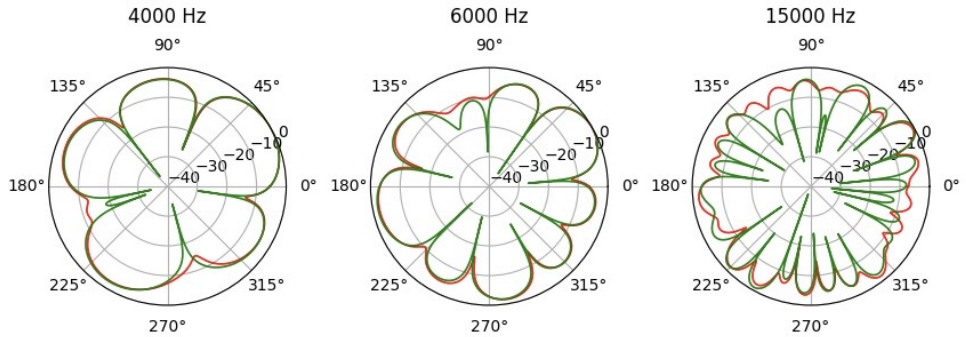
Z grafů je vidět, že pro nižší frekvence je směrovost ovlivněná minimálně. To ovšem neplatí pro vyšší frekvence, zejména v úhlech $\varphi = 20°$ a $30°$. Na obrázku 3.8 je proto vidět, jak by směrovost byla ovlivněna na 14, 16 a 18 kHz. Zajímavým zjištěním pak je, že směrovost se horší především ve směrech, ve kterých není umístěn fyzický mikrofon.



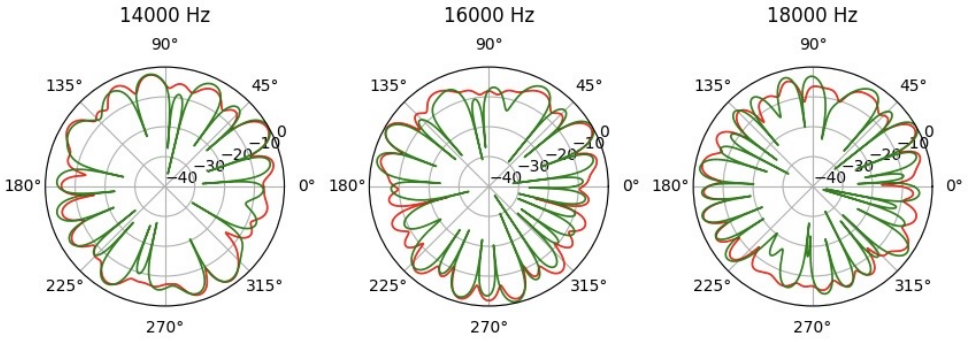
Obrázek 3.5: Kvantizace zpoždění pro vzorkovací frekvenci 48kHz a úhlem $\varphi = 0°$. Zeleně je vyznačen graf bez zaokrouhleného zpoždění a červeně se zaokrouhleným zpožděním.



Obrázek 3.6: Kvantizace zpoždění pro vzorkovací frekvenci 48kHz a úhlem $\varphi = 20°$. Zeleně je vyznačen graf bez zaokrouhleného zpoždění a červeně se zaokrouhleným zpožděním.



Obrázek 3.7: Kvantizace zpoždění pro vzorkovací frekvenci 48kHz a úhlem $\varphi = 30°$. Zeleně je vyznačen graf bez zaokrouhleného zpoždění a červeně se zaokrouhleným zpožděním.



Obrázek 3.8: Kvantizace zpoždění pro vzorkovací frekvenci 48kHz a úhlem $\varphi = 30°$. Zeleně je vyznačen graf bez zaokrouhleného zpoždění a červeně se zaokrouhleným zpožděním. Vyobrazeno na vyšších frekvencích.

Hlavním závěrem z pozorování grafů směrovosti je, že nevznikají žádné velké anomálie, které by negativně ovlivňovali směrovost pro posluchače.

Například hlavní lalok zůstává vždy zachován, směrovost je pouze lehce ovlivněna v určitých směrech oproti výpočtům. To pro posluchače v praxi nemá příliš velký význam, jelikož pro něho směrovost bude v oblastech postranních laloků stejně těžce předvídatelná jak v různých směrech, tak i na různých frekvencích. Posluchač by tedy neměl slyšet signifikantní rozdíl mezi kvantizovanými a nekvantizovanými zpožděními. Z toho plyne, že není nutné přecházet k metodám jako je zpožďování signálu pomocí lineární interpolace, jak je popsáno v kapitole 1.4.2, jelikož na posluchače nebude mít zásadní efekt.

3.3 Diferenciální mikrofonní pole prvního řádu

Ačkoliv má vyrobený prototyp cirkulární geometrii, tak na jednotlivé protilehlé páry mikrofonnů lze nahlížet jako na speciální případ DMA, a to sice DMA prvního řádu. Důvodem použití této techniky je její schopnost vytvořit kardioidní směrovou charakteristiku, která oproti DSB lépe potlačuje zvukové vlny přicházející zezadu. V předchozí kapitole bylo zjištěno, že směrovost DSB na frekvencích pod 1 kHz není moc signifikantní (viz obrázek 3.2), z toho důvodu by ji mohla kardioidní směrová charakteristika DMA prvního řádu značně vylepšit.

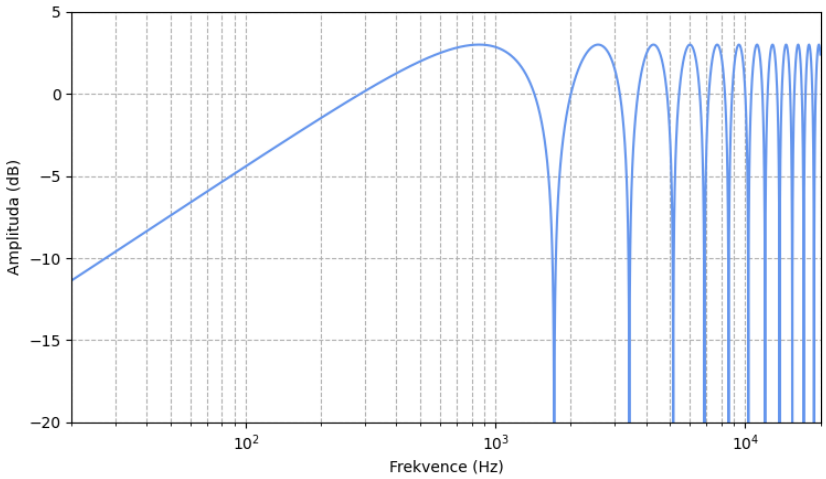
Cílem vývoje tohoto algoritmu je především zlepšit směrovou charakteristiku pro algoritmus hledající nejhlasitější zdroj zvuku v prostoru.

3.3.1 Směrová charakteristika

Jak bylo zjištěno v kapitole 1.3, tak DMA bude mít zhruba kardioidní charakteristiku do takové frekvence, jejíž polovina vlnové délky se rovná vzdálenosti mezi mikrofony. Výpočet této frekvence lze odvodit jednoduchým výpočtem daným vzorcem

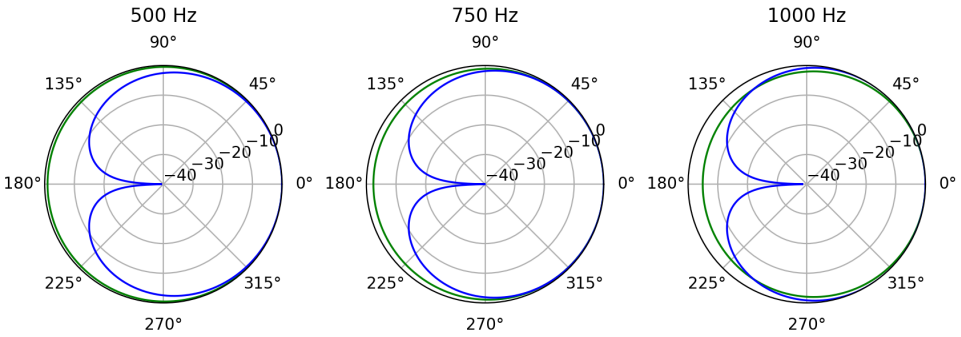
$$\frac{\lambda}{2} = \frac{343}{2 * 0,1} = 1715. \tag{3.1}$$

Z toho plyne, že na každém celočíselném násobku frekvence 1715 Hz bude výstup ze směru 0° nulový. S každým násobkem této frekvence zároveň bude stoupat počet postranních laloků. Pro lepší vizualizaci byla sestrojena přenosová funkce, která je vidět na obrázku 3.9.



Obrázek 3.9: Přenosová funkce ve směru 0° pro prototyp cirkulárního mikrofonního pole z FEL ČVUT.

Podle vzorečku na graf směrovosti z kapitoly 1.3 byli sestaveny grafy pro parametry prototypu cirkulárního mikrofonního pole. Ty jsou včetně jejich porovnání s DSB vyobrazeny na obrázku 3.10.



Obrázek 3.10: Porovnání grafů směrovosti pomocí technik DSB (zeleně) a DMA prvního řádu (modře) pro frekvence 500, 750 a 1000 Hz.

3.3.2 Aproximace mikrofonních párů v dalších směrech

Vzhledem k tomu, že máme v prototypu cirkulárního mikrofonního pole pouze 8 mikrofonů, tak by se mohlo zdát, že můžeme efektivně izolovat zvukové vlny pouze v osmi směrech, tj. pouze v místech, kde se fyzicky nachází mikrofonní páry. Tak tomu ovšem není, jelikož díky cirkulární geometrii prototypu mikrofonního pole můžeme aproximovat mikrofony i v místech, ve kterých se fyzicky nenacházejí. To nám umožňuje pomocí DMA prvního

řádu rozlišovat až nekonečné množství směrů po ose celého cirkulárního mikrofonního pole.

Pro lepší pochopení si představme úsečku, která propojuje sousední 2 mikrofony s indexy 1 a 2 v rámci cirkulárního mikrofonního pole. Pro získání aproximovaného mikrofonu mezi nimi stačí signály z mikrofonů $s_1(t)$ a $s_2(t)$ vynásobit vahami odpovídající bodu na úsečce, na které chceme nový mikrofon aproximovat. Bod, na kterém se bude nacházet aproximovaný mikrofon, označme zlomkem v, kdy $v \in (0, 1)$. Tento zlomek pak vyjadřuje vzdálenost na úsečce vzhledem k mikrofonu s indexem 1. Zlomek v zároveň bude sloužit jako váha, kterou budeme násobit signál $s_1(t)$. Druhou váhu pak získáme jako doplněk zlomku v do jedné. Signál $s_{approx}(t)$ následně získáme pomocí sečtení signálu $s_1(t)$ a $s_2(t)$ s aplikovanými vahami. Dohromady nám tyto operace vyjádřit vzorcem

$$s_{approx}(t) = vs_1(t) + (1 - v)s_2(t). \tag{3.2}$$

Díky tomuto mechanismu můžeme aproximovat nekonečné množství dalších párů mikrofonů v rámci mikrofonního pole, což nám dává lepší zaměření zdroje zvuku v okolí.

Vzhledem k tomu, že aproximované mikrofony již neleží na kružnici cirkulárního mikrofonního pole, je třeba vzít na vědomí, že mezi sebou budou mít kratší rozestup než fyzické mikrofony ležící na ní. Tuto vzdálenost lze jednoduše spočítat pomocí kosinovy věty. Uvažujme proměnnou $2R$ jako průměr cirkulárního mikrofonního pole. Délku úsečky $d_{m_i,m_{i+1}}$, která spojuje 2 sousední mikrofony na kružnici, vypočítáme jako

$$d_{m_i,m_{i+1}} = \sqrt{(2R)^2 - (2R)^2 \cos\frac{\pi}{4}}. \tag{3.3}$$

Díky znalosti $d_{m_i,m_{i+1}}$ pak získáme rozestup δ_{approx} mezi aproximovanými mikrofony jako

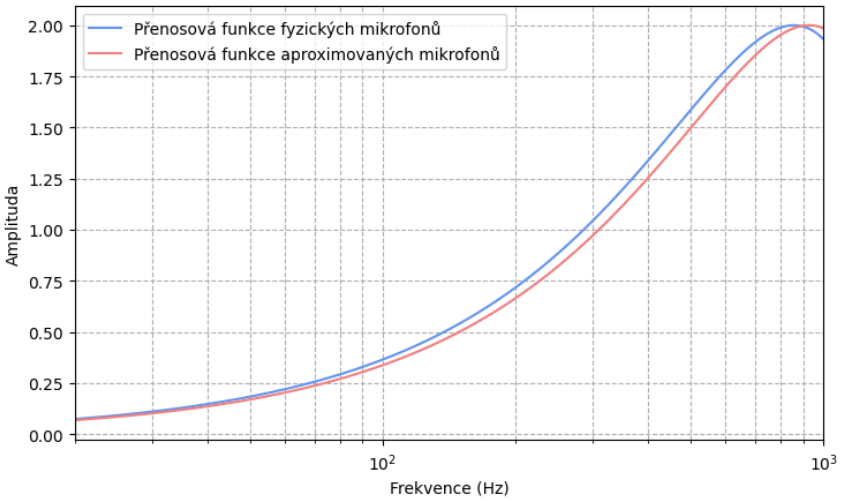
$$\delta_{approx} = \sqrt{(vd_{m_i,m_{i+1}})^2 + (2R)^2 - 2Rvd\cos\frac{3\pi}{8}}. \tag{3.4}$$

Ve skriptu je na to pak vytvořená funkce `approximation_radius`, která jako parametry bere průměr cirkulárního pole a zlomek v, na kterém má najít vzdálenost. Pro představu je při průměru cirkulárního mikrofonního pole 10 cm rozestup mezi aproximovanými mikrofony v bodě $v = \frac{1}{2}$ zhruba 9.2 cm.

Analýza přenosové funkce aproximovaných mikrofonů

Díky tomu, že jsme zkrátili vzdálenost mezi aproximovanými mikrofony, jsme zkrátili i čas, který zvuková vlna urazí při pohybu mezi nimi. To vede k tomu, že přenosové funkce budou vypadat trochu jinak, jak je vidět na obrázku 3.11.

To může způsobovat problém, jelikož jak je vidět z obrázku, tak na frekvencích menších než cca 850 Hz bude výstup z aproximovaných mikrofonů slabší než z fyzických. Ačkoliv bychom mohli usoudit, že takovéto změny jsou zanedbatelné, tak při provádění měření byl rozdíl viditelný. Tento problém se vyskytnul při testování algoritmu pro lokalizaci nejsilnějšího zdroje zvuku, který do podrobna rozebírá kapitola 5. Při testování nebyla rozdílná vzdálenost zohledněna, díky čemuž nefungoval správně.



Obrázek 3.11: Porovnání přenosových funkcí fyzických (modře) a aproximovaných mikrofonů s váhou $v = \frac{1}{2}$ (červeně). Zahrnuty jsou pouze frekvence pod 1 kHz. Signál je pro viditelnější zdůraznění změn zobrazen v lineárním měřítku, tj. není převeden na decibely.

Jedno z řešení tohoto problému spočívá ve vyrovnání přenosových funkcí aproximovaných a fyzických mikrofonů tak, aby měli ideálně stejný frekvenční průběh. Řekněme, že budeme kompenzovat přenosovou funkci aproximovaných mikrofonů tak, aby měla stejný frekvenční průběh jako u ta fyzických mikrofonů. Pro dosažení tohoto efektu můžeme využít přímé úměry mezi zpožděním τ a vzdáleností δ mezi mikrofony (viz vzorec 1.8). Ta je pro připomenutí daná jako

$$\tau_0 = \frac{\delta \cos 0}{c} = \frac{\delta}{c}. \tag{3.5}$$

Pro kalkulaci kompenzovaného zpoždění τ_{comp} použijme vzálenost mezi fyzickými mikrofony $2R$ (dvojnásobek poloměru cirkulárního mikrofonního pole). Musíme ovšem ještě zohlednit to, že aproximované mikrofony jsou si k sobě blíže, díky tomu tak vlna k druhému aproximovanému mikrofonu dorazí dříve, než je tomu u fyických mikrofonů. Pro efektivní kompenzaci zpoždění k nim proto ještě musíme přičíst rozdíl vzdáleností mezi mikrofony $2R - \delta_{approx}$. Výsledný vzorec pro výpočet kompenzovaného zpoždění τ_{comp} na aproximovaných mikrofonech pak můžeme vyjádřit jako

$$\tau_{comp} = \frac{2R + (2R - \delta_{approx})}{c} = \frac{4R - \delta_{approx}}{c} = \frac{4R}{c} - \frac{\delta_{approx}}{c} = 2\tau - \tau_{approx}. \tag{3.6}$$

Provedení takovéto kompenzace nám srovná frekvenční odezvu DMA prvního řádu fyzických a aproximovaných mikrofonů, což pak můžeme využít k srovnávání hlasitostí jejich výstupů. Konkrétní implementace a druhý způsob řešení tohoto problému jsou rozepsány v kapitole 5.4.

3.4 Zpracování naměřených dat

V rámci práce [1] bylo provedeno měření v bezodrazové komoře na Fakultě Elektrotechnické ČVUT v Praze. Pro měření byl Davidem Ringsmuthem vytvořen přípravek v rámci práce [18]. Ten je tvořen krokovým motorem a jednotkou jeho řízení, která komunikuje s PC pomocí seriové linky. Motor je možné otáčet do 200 různých úhlů v rámci jedné otáčky. Pro účely měření vytvořil Jan Šedivý v rámci své diplomové práce [2] skript, který integruje řízení přípravku a reproduktoru se záznamem výstupu z mikrofonního pole. Skript po svém spuštění nastaví motor na předurčený úhel a následně v reproduktoru přehraje 600 ms dlouhý sine sweep signál. Signál zaznamenaný mikrofonním polem pak uloží do souboru. Měření probíhalo pro 100 úhlů, výstupem tedy bylo 100 8-mi kanálových souborů, kde každý z kanálů odpovídá svému mikrofonu m_i.

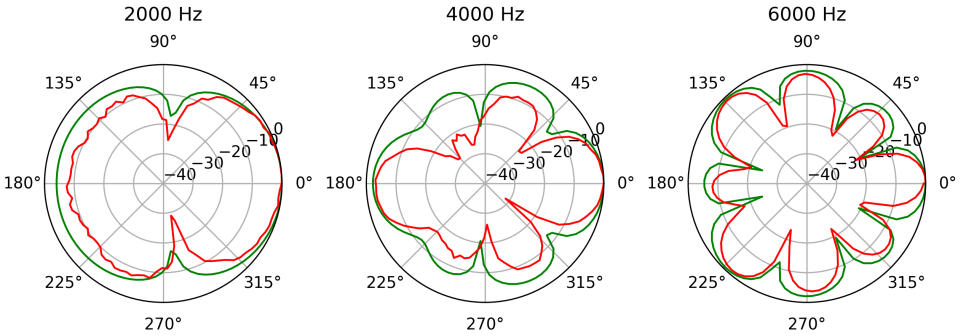
Na naměřená data byli v jazyce Python aplikovány algoritmy pro DSB a DMA. Zpracovaný WAV soubor pak prošel transformací do frekvenční domény pomocí rychlé fourierovy transformace (FFT) a následně byl vypočítán graf směrovosti. Pro konkrétní frekvenci f byly z výstupu FFT extrahovány

odpovídající amplitudy, které byly zobrazeny na polárním grafu. Veškeré amplitudy byli normalizovány k 0 dB pro příjemější analýzu.

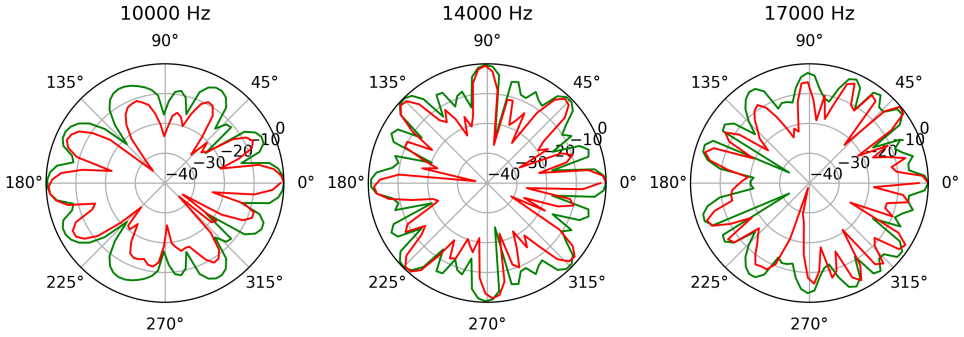
Analýzou naměřených dat byli sledovány 2 věci - zda konkrétní algoritmy (tj. DSB nebo DMA prvního řádu) fungují správně a zda je vyrobený prototyp cirkulárního mikrofonního pole dokáže korektně zpracovat.

3.4.1 DSB

Pro srovnání mezi naměřenými a vypočtenými daty byl do grafu zahrnut i graf směrovosti se zahrnutou kvantizací na 48 kHz a `resolution` = 100. Grafy jsou videt na obrázcích 3.12 a 3.13. Pro zpracování dat byl využit skript `DSB_processing.py` a pro zobrazení byl nakonfigurován skript `Measurements_processing.py`.



Obrázek 3.12: Srovnání grafů směrovosti mezi naměřenými daty (červeně) a vypočtenými daty (zeleně) pro frekvence 2, 4 a 6 kHz.

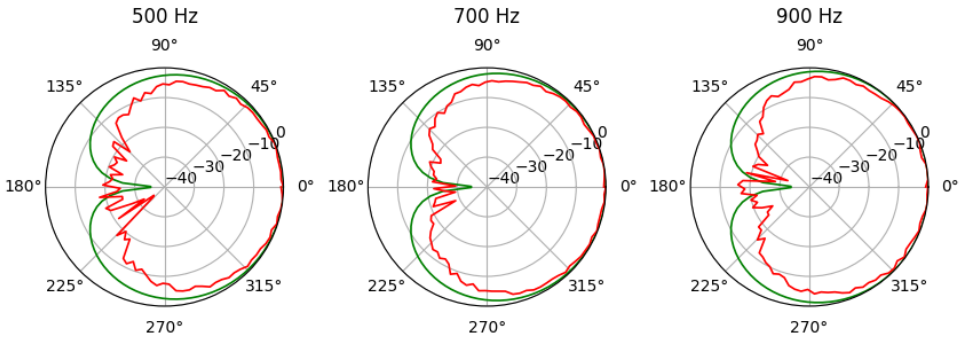


Obrázek 3.13: Srovnání grafů směrovosti mezi naměřenými daty (červeně) a vypočtenými daty (zeleně) pro frekvence 10, 14 a 18 kHz.

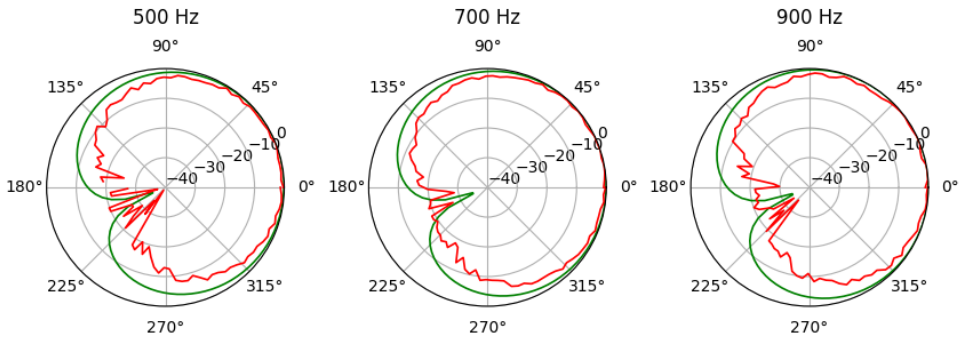
Porovnáním naměřených a vypočítaných dat z obrázků 3.12 a 3.13 je zjevné, že všechny hlavní a vedlejší laloky odpovídají i přes drobné nepřesnosti vypočítaným datům. Z toho můžeme usoudit, že DSB algoritmus pracuje korektně.

3.4.2 DMA prvního řádu

Cílem zpracování dat za využití DMA prvního algoritmu bylo ověřit, zda aproximace mikrofonů ve směrech, kde nejsou umístěny fyzické mikrofony, funguje správně. Testy byli provedené na frekvencích, kde má DMA prvního řádu kardioidní směrovou charakteristiku, tj. například frekvence 500, 700 a 900 Hz. Porovnání mezi naměřenými a vypočtenými daty jsou vidět na obrázcích 3.14 a 3.15. Pro zpracování dat byl využit skript `DMA_processing.py` a pro zobrazení byl nakonfigurován skript `Measurements_processing.py`.



Obrázek 3.14: Srovnání grafů směrovosti mezi naměřenými daty (červeně) a vypočtenými daty (zeleně) pro DMA prvního řádu ve směru fyzicky umístěného mikrofonu. Vyjádřeno v decibelech.

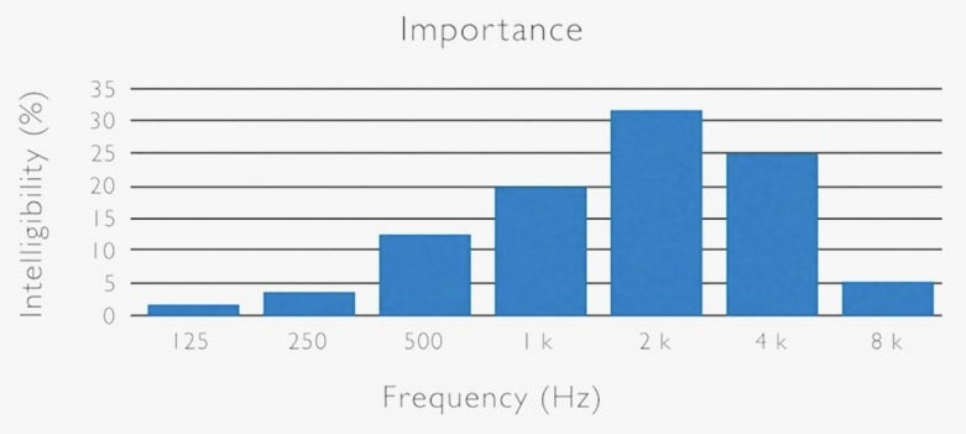


Obrázek 3.15: Srovnání grafů směrovosti mezi naměřenými daty (červeně) a vypočtenými daty (zeleně) pro DMA prvního řádu ve směru aproximovaného mikrofonu s váhou $v = \frac{1}{2}$. Vyjádřeno v decibelech.

Pohledem na grafy 3.14 a 3.15 můžeme sledovat sníženou směrovost zezadu, což odpovídá charakteristice kardioidy. Ze zpracování naměřených dat proto můžeme usoudit, že algoritmus funguje správně, a to i pro aproximované mikrofony.

3.5 Vyhodnocení použitelnosti

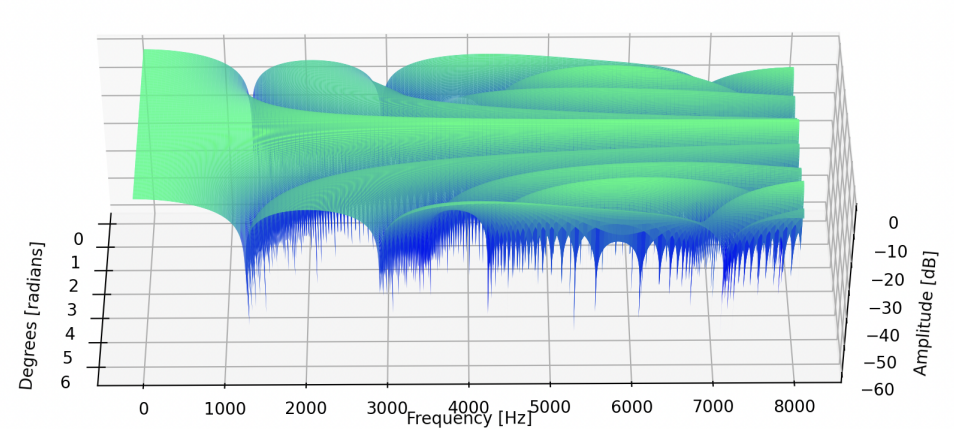
Na základě předešlé analýzy a ověření výpočtů na naměřených datech byl prototyp cirkulárního mikrofonního pole vyhodnocen jako použitelný a spolehlivý pro provádění DSB a DMA algoritmů. Největším jeho omezením je však jeho velikost. Řekněme, že bychom prototyp cirkulárního mikrofonního pole chtěli používat pro virtuální konference na zaměření jednotlivých mluvících osob v prostoru a izolaci ostatních. Pro tento případ potřebujeme největší citlivost na frekvencích mezi 2 až 4 kHz, jelikož tato spektra jsou nejdůležitější pro rozpoznání srozumitelnosti řeči, jak uvádí graf 3.16.



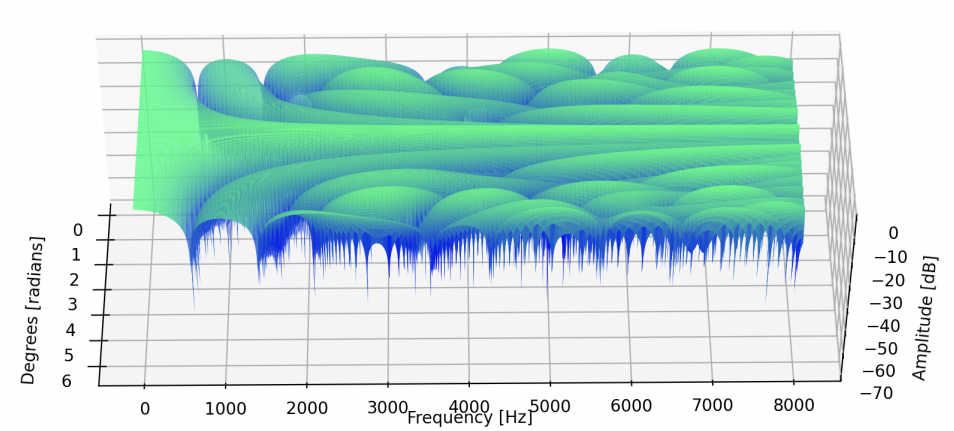
Obrázek 3.16: Graf nejpodstatnějších frekvencí pro rozeznání mluveného slova člověkem [19].

Po prozkoumání směrovosti z grafů 3.3 můžeme vidět, že v hlavní lalok v tomto frekvenčním spektru je velmi široký, což by mohlo vadit například ve chvíli, co by byl nežádoucí mluvčí v blízké vzdálenosti od toho zaměřovaného. Pokud bychom však zvětšili poloměr mikrofonního pole z 5 na 10 cm, tak získáváme užší hlavní lalok v námi požadovaném spektru. Porovnání těchto 2 poloměrů je vidět na obrázku 3.17.

Z obrázku 3.17 se tedy jeví jako vhodnější varianta verze o poloměru 10 cm. Omezenou směrovost menšího mikrofonního pole na nízkých frekvencích můžeme nicméně relativně jednoduše kompenzovat použití DMA prvního řádu, jak bylo rozebrané v kapitole 3.3. Z hlediska praktického využití je pak ovšem otázka velikosti důležitá, jelikož čím větší mikrofonní pole, tím větší vznikají nároky na skladování, přenášení, umístění na konferenční stůl, apod. Z toho plyne, že ačkoliv směrovost není ideální, tak velikost mikrofonního pole je minimálně pro experimentální účely zcela postačující.



(a) : Poloměr 5 cm



(b) : Poloměr 10 cm

Obrázek 3.17: Porovnání 3D grafů směrovosti DSB pro poloměry cirkulárního mikrofonního pole 5 a 10 cm ve frekvencích 20-8000 Hz. Směr vektoru $\hat{\kappa}$ odpovídá $\varphi = 180°$a počet mikrofonů je 8.

Kapitola 4

Implementace DSB algoritmu v reálném čase

Cílem této části je praktická implementace DSB algoritmu v reálném čase za použití jazyka Python 3 rozšířeného o knihovnu SoundDevice [13]. Veškeré vytvořené skripty jsou k nalezení v příloze C. FPGA deska komunikuje pouze se systémy Linux, tudíž je Python skript nutné spustit na počítači s operačním systémem Linux. Skript byl testován na počítači Lenovo ThinkPad x1 Carbon 3rd generation s operačním systémem Ubuntu 22.04.3 LTS (Jammy Jellyfish). S připojením prototypu cirkulárního mikrofonního pole k počítači mi pomáhal Jan Šedivý.

Struktura skriptu je vidět na diagramu tříd přiloženém v příloze A. Funkce z těchto tříd jsou pak použité ve skriptu `Fixed_direction.py`, který je aplikuje na audio stream vytvořen pomocí knihovny SoundDevice. Třídy jsou definované a implementované v souboru `Class_architecture.py`. Tento diagram tříd byl následně rozšířen o další třídy algoritmu pro lokalizci nejsilnějšího zdroje zvuku popsaném v kapitole 5.2.

Základem struktury programu je třída `MicrophoneArray`, jejíž cílem je ukládání základních informací o mikrofonním poli, jako je například počet nebo vzorkovací frekvence mikrofonů, poloměr cirkulárního mikrofonního pole, rychlost zvuku, apod. Třída `DSB` pak definuje důležité funkce pro provádění DSB algoritmu v reálném čase. Funkce `find_device_by_name` slouží k vyhledání identifikátoru mikrofonního pole mezi ostatními audio zařízeními v počítači. Tento identifikátor je pak nastaven jako vstupní zařízení ve skriptu `Fixed_direction.py`. Návod na spuštění skriptu a modifikaci inicializačních proměnných lze nalézt v příloze D.

Pro zpoždění o n vzorků je potřeba uchovávat data z předchozího bufferu, jelikož je nutné přičíst jeho koncové vzorky k začátku následujícího bufferu. Tento problém podrobněji popisuje kapitola 1.4.3. V implementaci

je řešený globální proměnnou `global previous_buffer`, do které jsou na konci běhu funkce `callback` pomocí hloubkové kopie překopírovaná data z bufferu reprezentovaného proměnnou `indata`.

Funkčnost skriptu byla ověřena poslechem, kdy byla slyšet snížená hlasitost zejména vyšších frekvencí pro směry mimo hlavní lalok. Funkce `cma_tmi` a `quantize_tmi_to_samples` byli pak odzkoušeny v kapitole 3.4 na naměřených datech z bezodrazové komory.

Kapitola 5

Algoritmus pro lokalizaci směru s největší hlasitostí

V předchozích kapitolách byly rozebrány techniky DSB a DMA prvního řádu, které byly následně aplikovány na prototyp cirkulárního mikrofonního pole vyvinutého na FEL ČVUT v rámci prací [1] [2]. To nám umožnilo zaměřit na námi určený směr hlavní lalok a úspěšně tak zeslabit zvukové vlny přicházející ze všech nežádoucích směrů. Této vlastnosti můžeme využít i v oblasti měření, jelikož schopnost izolace zvuku z nežádoucích směrů nám dává efektivní možnost měřit například hlasitost ve směru hlavního laloku. Tuto vlastnost následně můžeme využít pro porovnávání hlasitostí mezi různými směry zaměřenými hlavním lalokem. Tato schopnost má širokou škálu případů užití, jedním z nich může být například vývoj algoritmu pro hledání směru s nejsilnějším zdrojem zvuku.

Takovýto algoritmus spočívá v cyklickém otáčení hlavního laloku po celé jeho 360 stupňové ose, čimž provádí jakýsi sken svého okolí. Tímto způsobem je algoritmus na ovlivnění směrovosti aplikovaný na námi určený počet směrů, které jsou rovnoměrně distribuovány po $\varphi \in \langle 0, 2\pi)$. V průběhu tohoto procesu jsou pro každý směr vypočteny hodnoty RMS (efektivní hodnota), které poskytují informace o intenzitě zvukových signálů přicházejících z jednotlivých směrů. Ve směru, ve kterém byla naměřena nejvyšší hodnota RMS, se pak analogicky nachází nejsilnější zdroj zvuku. Je ovšem třeba mít na paměti, že tento směr může být často ovlivněn různými ozvěnami v místnosti, což může vést k špatné identifikaci reálného zdroje zvuku v prostoru.

Výstupní implementace z této kapitoly by měla sloužit jako Proof of Concept (PoC), která má ověřit funkcionalitu zkoumaných algoritmů pro budoucí vývoj. Ten může spočívat napíklad v uživatelsky přívětivé aplikace s odpovídajícím GUI. V této kapitole jsou diskutovány různé způsoby implementace, které ovlivňují jak výkon, tak přesnost nalezeného směru.

5.1 Cíle a požadavky

Cílem vývoje je nalézt algoritmus, který s dostatečnou přesností a v minimálním čase dokáže identifikovat směr nejsilnějšího zdroje zvuku. Díky tomu vznikají požadavky na dostatečnou efektivitu výpočtů, jelikož například při pohybu mluvčího po místnosti se směr zdroje zvuku může rapidně měnit. To by mohlo vést k vysoké slyšitelné latenci, kterou je třeba minimalizovat.

Při analýze prototypu cirkulárního mikrofonního pole bylo zjištěno, že aplikace DSB algoritmu na prototyp cirkulárního mikrofonního pole má velmi omezenou směrovost pro nízké frekvence (viz obrázek 3.2). Souběžně též bylo zjištěno, že DMA prvního řádu má na těchto frekvencí směrovost lepší (viz obrázek 3.10). Z tohoto důvodu byla vyvinutá druhá verze algoritmu, která vstupní signál rozděluje na nižší a vyšší pásmo pomocí butterworthova filtru s mezní frekvencí 1 kHz. Pro nízké pásmo je pak algoritmus pro lokalizaci nejsilnějšího zdroje zvuku počítán pomocí DMA prvního řádu a pro vyšší frekvence pomocí DSB. Hypotéza je taková, že zmíněné rozdělení výpočtů zlepší kvalitu určení směru na nízkých frekvencích. To by mělo vést k omezení chybovosti při určování směru nejsilnějšího zdroje zvuku s nízkou frekvencí.

Dalším požadavkem je mít dostupný nástroj k analýze, který nám umožní sledovat výsledky výpočtů v reálném čase, a to ideálně pomocí nějakého vizualizačního nástroje. Může se jednat například o aktualizující se polární graf, ve kterém budou vyobrazeny RMS hodnoty v jednotlivých směrech.

Na základě těchto cílů byli odvozeny funkční a nefunkční požadavky, které jsou popsány v následujících kapitolách 5.1.1 a 5.1.2.

5.1.1 Funkční požadavky

- F.1 - **Implementace algoritmu pro lokalizaci nejsilnějšího zdroje zvuku pomocí DSB**: Jako výchozí algoritmus pro lokalizaci směru s nejsilnějším zdrojem zvuku bude použit DSB algoritmus. Ten následně bude využit i pro zpracovávání zvuku pro frekvence vyšší než 1 kHz v kombinaci s DMA prvního řádu.
- F.2 - **Implementace algoritmu pro lokalizaci nejsilnějšího zdroje zvuku pomocí DMA prvního řádu na nízkých frekvencích**: Uživatel bude mít možnost zvolit kombinaci algoritmů DMA prvního řádu pro frekvence nižší než 1 kHz a DSB pro frekvence vyšší než 1 kHz.
- F.3 - **Parametrizace algoritmu**: Aplikace bude uživateli umožňovat nastavit klíčové parametry algoritmu, které ovlivňují způsob zpracování signálu.

- F.4 - **Vizualizace dat**: Aplikace bude poskytovat funkci vizualizace zpracovaných dat v reálném čase, která spočívá v grafickém znázornění RMS hodnot pro různé směry.

5.1.2 Nefunkční požadavky

- NF.1 - **Vysoké rozlišení směrů**: Aplikace bude poskytovat vysoké rozlišení směrů φ při detekci zvukových signálů, což znamená schopnost efektivně a přesně rozlišovat mezi různými směry zdrojů zvuku.
- NF.2 - **Nízká latence**: Aplikace bude zajišťovat minimální latenci během zpracování a vizualizace dat, aby bylo zajištěno, že obnovovací frekvence zaměření hlavního laloku na momentálně nejhlasitější zdroj v okolí, bude minimální.

5.2 Architektura aplikace

Pro větší přehlednost a zabránění kopírování kódu byla aplikace vzhledem k mnoha parametrům strukturalizovaná do tříd, jejichž diagram je dostupný v příloze A. Jeho základem je třída `MicrophoneArray`, která definuje veškeré proměnné mikrofonního pole, na kterém je prováděno zpracování signálu. Třída DSB pak provádí standardní DSB pro určitý směr, jak je popsáno v kapitole 4.

Od třídy DSB pak dědí třída `SourceFinderDSB`, která je následně rozšířena o `SourceFinderDSBCardioid`. Tyto třídy implementují 2 hlavní algoritmy pro lokalizaci nejsilnějšího zdroje zvuku, a to sice pomocí DSB a kombinace DSB a DMA prvního řádu. Tyto algoritmy lze pak různě upravovat podle dalších proměnných, které mají vliv na přesnost a délku výpočtu směru s největší hlasitostí. Konkrétní paramtery a implementace jsou rozebrány v dalších kapitolách.

5.2.1 Meziprocesová implementace

Pro naplnění nefunkčního požadavku NF.2 byla vytvořena implementace, která výpočty algoritmů paralelizuje mezi více procesů. Důvod volby procesů místo vláken je oddůvodněný v kapitole 2.2.1.

Aplikace byla proto rozdělena do 3 základních procesů, které zajišťují efektivní distribuci výpočtů:

- **Hlavní proces**: Má na starosti otevření vstupního a výstupního audio streamu, na který aplikuje DSB zpoždění v reálném čase. Přijatý buffer pak pomocí sdílené paměti předává výpočtenímu procesu, který z něho určí směr s největší hlasitostí. Zároveň slouží ke spuštění všech ostatních procesů.

- **Výpočetní proces**: Skenuje prostor pomocí aplikace zpoždění zvoleného algoritmu (DSB nebo DMA) pro předdefinované množství směrů hlavního laloku φ. Pro každý směr změří RMS hodnotu a vyhodnotí směr s největší hlasitostí. Informace o tomto směru pak předá pomocí sdílené paměti hlavnímu procesu, který podle nich upraví směr hlavního laloku. Naměřené hodnoty RMS zároveň předá pomocí sdílené paměti vizualizačnímu procesu. Na dokončení výpočtu ostatní procesy upozorňuje pomocí eventů. Ve chvíli co dopočítá data, tak čeká na přijetí dalších pomocí metody `wait()` z `multiprocessing.Event`.

- **Vizalizační proces**: Má na starosti vizualizaci dat pomocí interaktivního polárního grafu z knihovny Matplotlib [12]. V uživatelem definovaných intervalech pak aktualizuje data přijatá výpočetním procesem. Pokud data nejsou ještě přístupná, tak čeká na jejich přijetí pomocí metody `wait()` z `multiprocessing.Event`. Reálná obnovovací frekvence je ovšem často limitovaná výkonem uživatelského počítače. Funkce, běžící v rámci tohoto procesu, se jmenuje `realtime_visualization`.

Pro synchronizaci a meziprocesovou komunikaci jsou použity události `multiprocessing.Event`, zámky `multiprocessing.Lock`, sdílená paměť `multiprocessing.shared_memory` a sdílené pole `multiprocessing.Array`. Případy, kdy více procesů přistupuje ke stejným vláknem, jsou vždy ošetřeny zámky.

Optimalizace pomocí multiprocessing.Pool

Pro zvýšení efektivity algoritmu bylo použito funkce `multiprocessing.Pool` pro efektivní distribuci výpočtů mezi více procesů, jak je popsáno v kapitole 2.2.1. Toho bylo využito v rámci výpočetního procesu, který na začátu svého běhu inicializuje `Pool` procesů, mezi které distribuuje výpočty směrů φ. Počet procesů odpovídá počtu jader v počítači, které jsou získány pomocí `os.cpu_count()`. Počet inicializovaných procesů lze měnit v kódu, ovšem varianta, kde jejich počet odpovídá počtu jader, z testování vyšla jako nejvhodnější.

5.3 Lokalizace směru s největší hlasitostí pomocí DSB

V tomto případě je naším cílem aplikace DSB zpoždění na jeden přijatý audio buffer a změřit jeho RMS, nikoliv na signál v reálném čase. To nám značně usnadňuje práci narozdíl od kapitoly 4, kde pro aplikaci zpoždění byla nutná znalost dat z předchozího bufferu. V tomto případě nám pouze stačí zkrátit délku jednotlivých kanálů tak, aby odpovídala velikosti zpoždění. Toho docílíme tím, že zepředu bufferu odebereme n vzorků, které odpovídají vypočítanému zpoždění. Alternativně můžeme jednotlivé kanály zpozdit o n vzorků a data z předchozího bufferu nahradit nulami. Tato operace nám usnadňuje práci, jelikož odebírá potřebu znalosti předchozího bufferu pro aplikaci zpoždění. To vede ke zvýšení výkonu celé naší aplikace.

Tato metoda má ovšem svojí nevýhodu, a tou je sice zkrácení analyzovaného bufferu o počet vzorků, který dopovídá největšímu DSB zpoždění. V našem případě nás toto ovšem netrápí, jelikož používáme audio buffer délky 4800 vzorků a největší zpoždění odpovídá 14 vzorkům, tudíž efekt zkrácení analyzovaného signálu bude zanedbatelný. Problém by to mohl ovšem být při použití menšího bufferu a většího mikrofonního pole, pro které by největší zpoždění odpovídalo většímu množství vzorků. Zde by mohlo zpoždění odpovídat většímu počtu vzorků než je velikost bufferu. Takovýto případ nebyl v rámci této práce řešený a v případě jeho výskytu při budoucím vývoji by musel být řešen komplexněji za použití znalosti dat z předchozího bufferu.

Implementace tohoto způsobu zpožďování signálu je dostupná ve funkci `cut_dsb_delays` v rámci třídy `SourceFinderDSB`. Slovo "cut"je použité z toho důvodu, že se obrazně řečeno jedná o "ustřižení"vzorků ze začátků a konců jednotlivých kanálů bufferu pro simulaci zpoždění.

Algoritmus byl primárně implementován pro hledání nejhlasitějšího zdroje pouze v horizontální rovinně pro různé úhly φ, kde má mikrofonní pole největší směrovost. Pro správnou funkčnost algoritmu musí být proto `THETA_STEPS` nastavené na hodnotu 1. Při testování algoritmu pro `THETA_STEPS` větší než 1 algoritmus v momentální verzi nefungoval spolehlivě, jedná se tedy bod, na který se lze zaměřit v rámci budoucího vývoje.

5.4 Lokalizace směru s největší hlasitostí pomocí DMA prvního řádu

Cílem této sekce je otestovat hypotézu, zda algoritmus bude fungovat lépe za použití DMA prvního řádu na nízkých frekvencích, než DSB, jak je

popsáno v kapitole 5.1. Toho bylo dosaženo rozdělením signálu na 2 frekvenční pásma pomocí butterworthova filtru na mezní frekvenci 1 kHz. Implementace butterworthova filtru byla použitá z knihovny SciPy [15]. Po aplikaci výpočtů na DMA a DSB byli pak jejich výsledné RMS hodnoty zkombinovány pomocí vzorce

$$rms = \sqrt{rms_{low}^2 + rms_{high}^2}. \tag{5.1}$$

Překryv frekvenčních pásem na mezní frekvenci vytvořený butterworthovým filtrem při zpětném skládání RMS hodnot byl zanedbán. Pro získání většího rozlišení směrů φ bylo využito aproximace dalších mikrofonních párů, jak je tomu popsáno v kapitole 3.3.2. Algoritmus byl prvotně testován bez zohlednění rozdílnosti přenosových funkcí aproximovaných mikrofonů tím, že na nich zpoždění bylo zaokrouhleno. Tento algoritmus nicméně nefungoval, což vedlo k další analýze, která odhalila rozdílnost přenosových funkcí popsanou v zmíněné kapitole 3.3.2. Řešení tohoto problému bylo docíleno pomocí 2 metod popsaných v 5.4.1 a 5.4.2.

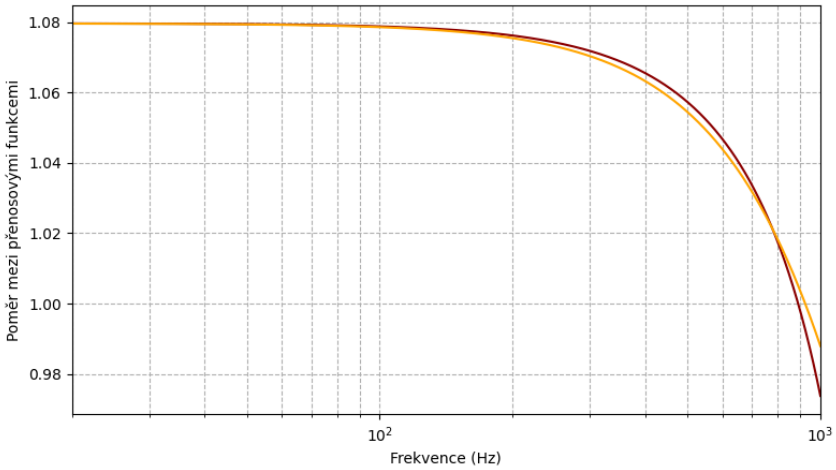
5.4.1 Kompenzace přenosových funkcí aproximovaných mikrofonů pomocí lineární interpolace

Pro kompenzaci přenosových aproximovaných mikrofonů na nich bylo použité zpoždění definované vzorcem 3.6. Implementace tohoto vzorce pro τ_{comp} je dostupná ve funkci `tau_compensation`. Zpoždění jsou ukládané ve slovníku `interpol_delays`, který je vypočítaný funkcí `prepare_interpol_delays`. Teoretický popis tohoto problému bylo již popsán v kapitole 3.3.2.

5.4.2 Kompenzace přenosových funckcí aproximovaných mikrofonů pomocí butterworthova filtru

Myšlenkou této metody je zesílení signálu z DMA prvního řádu aproximovaných mikrofonů tak, aby se jejich přenosová funkce co nejvíce podobala přenosové funkci fyzických mikrofonů. Jak ovšem můžeme vidět z grafu 3.11, tak nutnost zesílení je proměnlivá s rostoucí frekvencí, tudíž nám nestačí pouze vynásobit signál skalárem v časové doméně. Jak se potřebná míra zesílení mění s rostoucí frekvencí můžeme pozorovat v grafu na obrázku 5.1. Graf ukazuje, jakým koeficientem musíme vynásobit signál z výstupu DMA aproximovaných mikrofonů pro získání hlasitosti z DMA fyzických mikrofonů.

Z grafu 5.1 si můžeme všimnout, že se poměry podobají frekvenčnímu průběhu butterworthova lowpass filtru. Této pobnosti můžeme využít v



Obrázek 5.1: Poměr mezi hlasitostmi výstupu DMA prvního řádu z fyzických a aproximovaných mikrofonů (červeně). Aproximované mikrofony jsou počítány s váhou $v = \frac{1}{2}$. Oranžově je znázorněn butteworthův lowpass filtr prvního řádu s frekvencí 2256 Hz.

kompenzaci rozdílných hlasitostí přenosových funkcí tím, že signál z DMA aproximovaných mikrofonů nejdříve vynásobíme nevyšším rozdílovým koeficientem (například pro $v = \frac{1}{2}$ je to 1.08) a následně signál odfiltrujeme pomocí butterworthova filtru. Rozdíly v hlasitostech tam sice stále budou, ovšem budou tak malé, že je můžeme zanedbat.

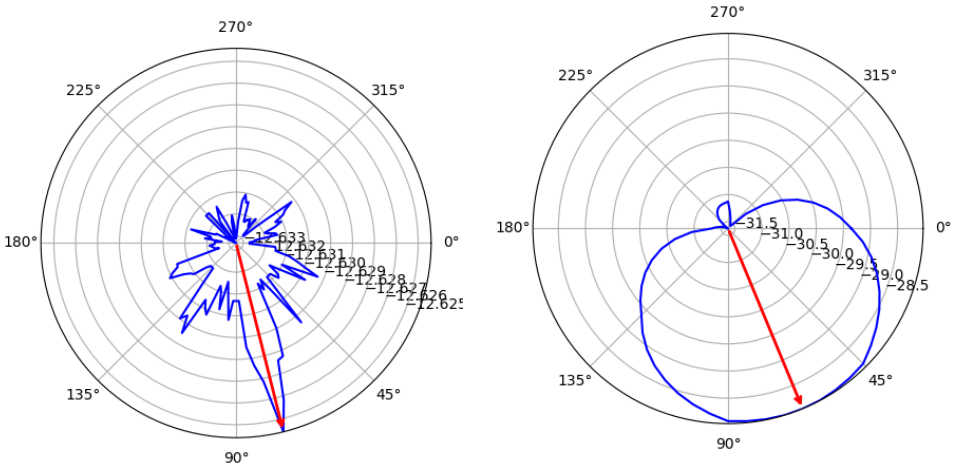
Pro nalezení vhodné mezní frekvence butterworthova filtru bylo využito porovnávání frekvenčních průběhů funkcí poměrů hlasitostí a odezvy butterworthova filtru. K jejich porovnání bylo využito jednotek MAE (Mean Absolute Error), které udávají průměrnou míru chybovosti jedné funkce k druhý. Mezní frekvence pro butterworthův filtr s nejnižší hodnotou MAE byla vyhodnocena za nejpodobnější a tedy vhodnou pro pužití v praxi. Implementaci zmíněného postupu nabízí funkce `find_most_similar_frequency`.

Vzhledem k tomu, že výpočet pro nalezení vhodné frekvence, na které je algoritmus podobný, je výpočetně náročný, bylo pro ukládání již hotových výpočtů využito JSON souboru. Vypočítané mezní frekvence jsou proto ukládány pro každou váhu v v souboru `cutoff_data.json` pro využití v dalších bězích programu. Pokud data pro danou váhu v v souboru chybí, tak je skript vypočítá před otevřením audio streamu a uloží je do souboru.

Cílem vývoje této metody bylo porovnání s metodou lineární interpolace, vůči které by mohla být rychlejší. Tato hypotéza byla otestována v kapitole 5.6.

5.5 Vizualizace v reálném čase

Pro naplnění požadavku F.4 byla vytvořena vizualizace dat pomocí interaktivního polárního grafu z knihovny Matplotlib. Jeho běh je dán vizualizačním procesem popsaným v 5.2.1, který přijímá data od výpočetního vlákna v reálném čase. V polárním grafu byla zároveň vytvořena šipka, která ukazuje směrem, ve kterém byl nalezen nejsilnější zdroj zvuku. Data jsou zobrazována v decibelech a normalizovaná k aktuálně nejvyšší a nejnižší hodnotě. To uživateli umožňuje lépe pozorovat rozdíly mezi hlasitostmi jednotlivých směrů. Na obrázku 5.2 jsou k dispozici snímky obrazovky toho, jak diagram může vypadat při svém běhu. Snímek byl pořízen v moment, kdy na prototyp mikrofonního pole byl nasměrován reproduktor s bílým šumem. Z grafů je hezky vidět úzký hlavní lalok DSB algoritmu a naopak velmi široký hlavní lalok kardioidy.



(a) : DSB pro celé frekvenční spektrum

(b) : DMA prvního řádu pro frekvence menší než 1 kHz

Obrázek 5.2: Vizualizace v reálném čase za průběhu algoritmu pro lokalizaci nejhlasitějšího zdroje v okolí. Na 5.2a je zobrazen DSB algoritmus na celém frekvenčním spektru a na 5.2b je zobrazen pouze DMA prvního řádu pro frekvenční pásmo pod 1 kHz.

5.6 Porovnání vytvořených algoritmů

Základním parametrem použitým při porovnávání algoritmů byl průměrný čas na výpočet směru z jednoho bufferu. Měření bylo prováděno vždy po dobu 20 sekund, načež byl spočítán průměrný čas výpočtů směru z jednoho bufferu v milisekundách. Měření bylo prováděno bez běžící vizualizace v reálném čase pro získání více autentických výsledků. Veškeré testování bylo prováděno na počítači Lenovo ThinkPad x1 Carbon 3rd generation s operačním systémem

Ubuntu 22.04.3 LTS (Jammy Jellyfish). Výsledky jsou dostupné v tabulce 5.1.

	8	**32**	**128**	**512**	**2048**
DSB_LOOP	4.4	14.6	58.9	226.9	921.2
DSB_POOL	11.0	22.8	57.2	177.2	610.8
DSB_CARD_LOOP interpol	9.4	27.6	107.3	394.2	1552.1
DSB_CARD_LOOP butter	8.8	28.1	109.1	422.5	1697.8
DSB_CARD_POOL interpol	36.2	79.5	108.0	303.6	1061.2
DSB_CARD_POOL butter	37.1	83.0	115.3	322.5	1182.5

Tabulka 5.1: Průměrné časy výpočtu nejhlasitějšího směru z jednoho bufferu v milisekundách. V prvním řádku jsou počty směrů φ, které algoritmus skenoval. V prvním sloupci jsou uvedené algoritmy, které byli použité pro výpočty. Podrobnější popis jejich názvosloví je uveden v příloze D.2.1.

Z tabulky 5.1 můžeme pozorovat, že pro nízký počet směrů φ je paradoxně efektivnější varianta bez distribuovaných výpočtů. To je nejspíše způsobeno tím, že samotná operace paralizace je moc dlouhá na to, aby byla na takto malém počtu výpočtů efektivní. To se obrací na zhruba 128 směrech, kde jsou oba algoritmy zhruba stejně výkonné. Na vyšších počtech směrů je pak efektivnější paralelizace směrů pomocí `multiprocessing.Pool`. Překvapující je to zejména u algoritmu na výpočet směrů pomocí DMA prvního řádu, kdy je paralelizovaný algortimus pro 8 směrů zhruba 4x pomalejší.

Zajímavé zjištění také je, metoda kompenzace přenosové funkce pomocí butterworthova filtru se při velkém množství dat jeví jako méně efektivní. Na nižším počtu dat má pak zhruba stejnou efektivitu jako lineární interpolace.

Je nicméně nutno podotknout, že takovéto výsledky byli získány měřením pouze na jednom počitači. Na počítačích s jinými procesory (především s jiným počtem jader) se proto výsledky mezi distribuovanými a nedistribuovanými algoritmy mohou lišit.

Závěrem lze říci, že časy výpočtů se pohybují v mezích vhodné pro praktické použití, což naplňuje nefunkční požadavky NF.1 a NF.2 To vyplývá z toho, že časy se pohybují často v desítkách, až stovkách milisekund, což je velmi přijatelná rychlost. V případě, že by se časy pohybovali ve vyšších sekundách, až minutách, by bylo třeba uvažovat nad větší optimalizací algoritmu, případně způsobu efektivního zredukování množství výpočtů.

5.7 Testování v audiovizuálním studiu

Testování bylo provedené v audiovizuálním studiu Katedry radioelektroniky FEL ČVUT. Cílem testování bylo ověřit funkci algoritmu pro lokalizaci směru

s nejsilnějším zdrojem zvuku.

Testování probíhalo umístěním 2 reproduktorů značky M-Audio ve stejné vzdálenosti od cirkulárního mikrofonního pole pod úhlem 90°, jak je vidět na obrázku 5.3. Signál do reproduktorů byl posílán počítačem přes zvukovou kartu Focusrite Scarlett 2i2 2. gen pomocí XLR kabelů. Signál byl generován v programu Audacity. Vzdálenost mikrofonů od sebe byla stejná a před začátkem testování byli hlasitosti reproduktorů vykalibrovány na stejnou úroveň pomocí zvukoměru.

Obrázek 5.3: Testování v audiovizuálním studiu

5.7.1 Struktura testování

Testování pak probíhalo tak, že do reproduktoru pod úhlem 90° byl posílán bílý šum a do reproduktoru pod úhlem 0° byla posílaná sinusoida s požadovanou frekvencí. Následně byl algoritmus puštěn na 10 sekund, během kterých zapisoval do souboru úhel směru s nejvyšší detekovanou hlasitostí. Následně byly postupně zvyšovány a snižovány hladiny šumu a sinusoidy, přičemž bylo sledováno, jak se algoritmus bude chovat. Tím bylo testováno, zda je algoritmus schopný efektivně rozlišit signál z příchozího směru od okolního šumu.

Testování probíhalo pro 2 frekvenční pásma, na kterých byli testovány

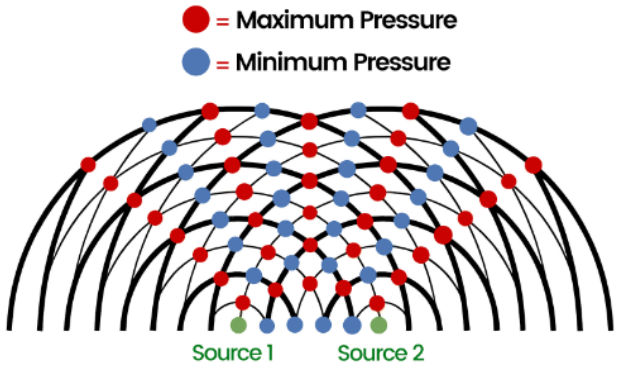
algoritmy DSB a DMA. Jako frekvence byli zvolené 600 a 2000 Hz. Testovány byli následující scénáře:

- 1) Frekvence 600 Hz
  - 1a) Lokalizace pomocí DMA prvního řádu s kompenzací přenosové funkce aproximovaných mikrofonů pomocí butterworthona filtru
  - 1b) Lokalizace pomocí DMA prvního řádu s kompenzací přenosové funkce aproximovaných mikrofonů pomocí lineární interpolace
  - 1c) Lokalizace pomocí DSB
- 2) Frekvence 2000 Hz
  - 2a) Lokalizace pomocí DSB

Testování probíhalo pouze pro odlišné úhly φ (tj. úhel θ byl nastaven vždy na 90 stupňů), jejichž množství bylo nastaveno na 32. To znamená, že rozdíl mezi kroky bylo vždy 11.25 stupňů. Úhly θ byli všechny nastaveny na 90°.

5.7.2 Vlnová interference

V průběhu měření byl vypozorován jev konstruktivní vlnové interference, která způsobila, že při vyrovnané hlasitosti obou reproduktorů začal algoritmus jako směr s nejhlasitějším zdrojem zvuku vyhodnocovat směr mezi úhly 0° a 90°. Takovýto jev nastává ve chvíli, co se 2 vlny o stejné frekvenci setkají v takovém bodě v prostoru, kde se sečtou, čímž můžou mít vyšší nebo nižší výslednou amplitudu. Z tohoto důvodu se při zvyšování hlasitosti jednoho z reproduktorů začal identifikovaný směr vždy postupně posouvat k reálně hlasitějšímu zdroji, než se tam obrátil úplně. Jedná se o základní fyzikální jev, jež je znázorněn na obrázku 5.4.



Obrázek 5.4: Příklad vlnové interference v prostoru se 2 umístěnými reproduktory. Převzato z [20].

5.7.3 Výsledky testů

Výsledky přinesly překvapivé pozitivní zjištění, jelikož algoritmus fungoval dobře i při vysoké hladině šumu přicházejícího z reproduktoru pod 90°. Z toho důvodu byly i zahrnuté testy, kdy byla snižována hlasitost sinusoidy pro nalezení hranice, na které bude algoritmus již jako nejsilnější zdroj zvuku detekovat směr šumu.

V případě scénářů 1a) a 1b) byla detekována vlnová interference mezi úhly 0° a 90° při snížení signálu sinusoidy o cca -10 dB oproti šumu. Do té doby algoritmus detekoval zdroj vždy ve směru 0°. Vlnová interference se zde vyskytovala až do snížení hladiny sinusoidy oproti šumu o -25 dB.

V případě scénářů 1c) se algoritmus choval velmi podobně jako 1a) a 1b), ovšem s tím rozdílem, že vlnová interference pokračovala pouze do hladiny -20 decibelů. Z toho plyne, že hypotéza ohledně větší efektivity algoritmu DMA prvního řádu na nižší frekvence, byla podle provedených testů spíše vyvrácena, protože algoritmy se v tomto frekvenčním pásmu chovají téměř identicky. Pro úplné vyvrácení by bylo potřeba algoritmus testovat pro více frekvencí pod 1 kHz, na což v rámci semestru již nezbyl čas.

Pro scénář 2a) algoritmus nedetekoval žádnou vlnovou interferenci, jako to bylo v předchozích případech. Zde naopak došlo ke skoku, kdy na hladině snížení šumu o -5 dB byl detekovaný jako nejhlasitější směr úhel 0° a na snížení o -10 db byl již detekovaný směr zdroje šumu.

Veškeré naměřené hodnoty jsou dostupné v odpovídajících souborech v příloze C.

5.7.4 Závěry testování

Algoritmus funguje podle očekávání, vyniká zejména ve své schopnosti odlišovat harmonický signál od okolního šumu. Tohoto efektu byl schopný dosáhnout i za předpokladu, že hladina šumu byla až o 10 decibelů vyšší než harmonická vlna. V případě, kdy byl šum o cca 10-25 decibelů slabší, docházelo k vlnové interferenci, která zapříčinila identifikaci zdroje zvuku mezi směrem zdroje šumu a harmonické vlny. Varianty použití DMA prvního řádu DSB se ukázali jako téměř stejně efektivní, tudíž separátní zpracování signálů nízkého pásma se dle provedených testů nejeví jako spolehlivější varianta.

Zároveň byla při testování algoritmu využívající kompenzaci přenosové funkce aproximovaných mikrofonů pomocí butterworthova filtru objevena drobná chyba, která byla následně opravena. Tato chyba je pozorovatelná v souborech s naměřenými daty.

Kapitola 6

Vize budoucího vývoje

V rámci této práce byly prozkoumány možnosti ovlivňování směrovosti cirkulárního mikrofonního pole v časové doméně, načež byla vytvořena aplikace pro lokalizaci směru nejhlasitějšího zdroje zvuku, která má sloužit jako Proof of Concept pro další vývoj. Při práci na tomto projektu bylo zjištěno mnoho směrů, kterými by se práce mohla ubírat, jelikož se jedná o velmi široké téma. V této kapitole jsou nastíněné další směry možného budoucího vývoje.

6.1 Frekvenční doména

Prvním směrem je přesunutí algoritmů pro zpracování signálů z časové do frekvenční domény, tj. provedení FFT při každém přijetí bufferu. Frekvenční doména by umožňovala zpoždění signálu bez použití lineární interpolace, která je výpočetně náročná. Toho samého efektu by bylo možné docílit násobením signálu ve frekvenční doméně exponenciálou, která reprezentuje zpoždění. Jako další směr by nadále mohlo být hledání jiných algoritmů na provádění beamoformingu, které pracují ve frekvenční doméně.

6.2 Mikrofonní pole

V rámci této práce bylo pracováno pouze s mikrofonním polem o cirkulární geometrii. V další práci by bylo možné pracovat i s jinými geometriemi, jako je například sférická. Dále by bylo možné experimentovat s velikostí poloměru R cirkulárního mikrofonního pole, případně i kombinovat více cirkulárních geometrí dohromady. To by spočívalo ve výrobě dalšího cirkulárního mikrofonního pole, které by bylo umístěné uvnitř nebo vně toho původního.

Takováto geometrie by pak umožňovala zpracovávat signál za využití techniky cirkulárního diferenciálního mikrofonního pole vyšších řádů, jak je například rozebíráno v této knize [3].

6.3 Algoritmus pro lokalizaci směru s největší hlasitostí

Základem pro další vývoj by bylo provedení rozsáhlejšího testování stávajícího algoritmu, na které v rámci posledních 2 semestrů již nezbyl čas. Testovací scénáře by mohli zahrnovat například testování na reálné řeči v konferenční místnosti plné lidí nebo rozlišování mezi dvěmi podobně hlasitými zdroji zvuku v prostoru. Dále by pak bylo vhodné ostestovat algoritmus pro harmonické vlny o dalších frekvencích nebo schopnost rozlišit směry s minimálním rozdílem stupňů φ. Takovéto testování by mohlo vést k určení směrů vývoje, ve kterých momentální implementované algoritmy strádají.

V případě pokračování ve vývoji algoritmu pro lokalizaci nejsilnějšího zdroje zvuku pomocí DMA prvního řádu by bylo potřeba zvážit implementaci vyvažování hlasitostí na nižších frekvencí pomocí integrátoru. V momentální verzi žádné takovéto vyvažování prováděné není, z toho důvodu může mít algoritmus tendenci identifikovat jako hlasitější zdroje takové signály, které mají sice vyšší frekvenci, ale ne nutně vyšší hlasitost. Takovýto případ by ovšem bylo vhodné otestovat předtím, než se dojde k implementaci integrátoru. Vzhledem k tomu, že DSB funguje zhruba stejně dobře jako DMA prvního řádu by bylo ovšem z praktického hlediska třeba najít jiný smysl a cíl vývoje tohoto algoritmu.

Dalším bodem může být rozvoj algoritmu pro detekování směrů i ve vertikální rovině pro θ úhly. Struktura kódu je připravena na detekování vertikálního směru, nicméně v rámci testování nefungovala, směrem dalšího vývoje by bylo proto rozebrání a vyřešení tohoto problému.

Dalším směrem by mohla být tvorba uživatelsky přívětivé aplikace, která uživateli umožní nastavovat parametry mikrofonního pole v reálném čase pomocí grafického rozhraní (GUI). V takovémto případě by stálo za to zvážit implementaci aplikace na vhodnější platformě pro zvýšení výkonu, jako je například C++ [11]. Takováto aplikace by mohla být eventuelně spuštěná na externím jednodeskovém počítači, jako je například Raspberry Pi.

Kapitola 7

Závěr

V této bakalářské práci byly zkoumány možnosti využití cirkulárního mikrofonního pole s digitálními MEMS mikrofony pro zpracování a analýzu signálů přicházejících z různých směrů. Byly prozkoumány a implementovány techniky Delay and Sum Beamforming a diferenciální mikrofonní pole prvního řádu s cílem zlepšit směrovostní charakteristiky mikrofonního pole.

Prototyp cirkulárního mikrofonního pole, vyvinutý Davidem Vagnerem a Janem Šedivým, se ukázal jako efektivní nástroj pro analýzu a implementaci algoritmů ovlivňující směrovost, což bylo potvrzeno měřeními z bezodrazové komory a audiovizuálního studia. Výsledky ukázaly, že použité techniky mohou výrazně zlepšit izolaci zvukového signálu z určitého směru, což má významné aplikace v různých akustických systémech.

Během tvorby práce bylo také zjištěno, že implementace těchto technik v jazyce Python pomocí knihoven jako NumPy a SciPy nabízí dostatečný výkon pro praktické použití, minimálně v rámci testování. Pro zvýšení výkonu v rámci praktické aplikace by bylo ovšem vhodné zvážit jejich implementaci na efektivnější platformě, jako je například jazyk C++, pro minimalizaci latence.

Pro budoucí práce se nabízí rozšíření analýzy na další typy beamformingových algoritmů a porovnání jejich efektivity v různých akustických podmínkách. Také by bylo vhodné prozkoumat možnosti integrace těchto technik do komerčních akustických systémů, což by mohlo vést k jejich širšímu praktickému využití.

Literatura

[1] Vagner, D. *Návrh HW pro mikrofonní pole s MEMS mikrofony.* Diplomová práce. České vysoké učení technické v Praze, Fakulta elektrotechnická, 2023.

[2] Šedivý, J. *Systém na bázi FPGA pro zpracování dat ze senzorových polí v reálném čase.* Diplomová práce. České vysoké učení technické v Praze, Fakulta elektrotechnická, 2024.

[3] Benesty, J. – Chen, J. – Cohen, I. *Design of Circular Differential Microphone Arrays.* Springer Cham, Leden 2015. ISBN 978-3-319-14842-7.

[4] Roig, E. T. *Eigenbeamforming array systems for sound source localization.* Disertační práce, Technical University of Denmark, Listopad 2014.

[5] Clénet, B. *Circular microphone array based beamforming and source localization on reconfigurable hardware.* Master thesis, Graz University of Technology, Signal Processing and Speech Communications Laboratory, 2010.

[6] Messner, E. *Differential Microphone Arrays.* Master thesis, Graz University of Technology, Signal Processing and Speech Communications Laboratory, 2013.

[7] Benesty, J. – Chen, J. *Study and Design of Differential Microphone Arrays.* Springer, 2013. ISBN 978-3-642-33753-6.

[8] Honzík, P. *Komunikace a měření v multimediální technice 4.* Přednáška. Praha: ČVUT v Praze, 18. října 2023.

[9] Smith, J. O. *Linear Interpolation. In: Physical Audio Signal Processing.* W3K Publishing, 2010. Dostupné z: `https://ccrma.stanford.edu/~jos/pasp/Linear_Interpolation.html`.

[10] *Stack Overflow Developer Survey 2023* Online. Stack Overflow, Květen 2023. [cit. 2024-1-1]. Dostupné z: https://survey.stackoverflow.co/2023/#most-popular-technologies-language.

[11] Wilczek, J. *Top 5 Languages For Audio Programming* Online. Říjen 2023. [cit. 2024-1-1]. Dostupné z: https://thewolfsound.com/top-5-languages-for-audio-programming/.

[12] Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*. 2007, 9, 3, s. 90–95. doi: 10.5281/zenodo.7697899. [cit. 2024-1-1].

[13] Geier, M. – others. Sounddevice, 2020. Dostupné z: https://python-sounddevice.readthedocs.io/en/0.3.15/index.html.

[14] Harris, C. R. et al. Array programming with NumPy. *Nature*. September 2020, 585, 7825, s. 357–362. doi: 10.1038/s41586-020-2649-2. Dostupné z: https://doi.org/10.1038/s41586-020-2649-2. [cit. 2024-1-1].

[15] Virtanen, P. et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*. 2020, 17, s. 261–272. doi: 10.1038/s41592-019-0686-2.

[16] AJITSARIA, A. *What Is the Python Global Interpreter Lock (GIL)?* Online. Real Python, b.d. [cit. 2024-30-4]. Dostupné z: https://realpython.com/python-gil/#author.

[17] *Python 3.12.3 documentation*. Python Software Foundation. Dostupné z: https://docs.python.org/3.10/index.html.

[18] Ringsmuth, D. *Kondenzátorový mikrofon s dělenou pevnou elektrodou.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta elektrotechnická, 2023.

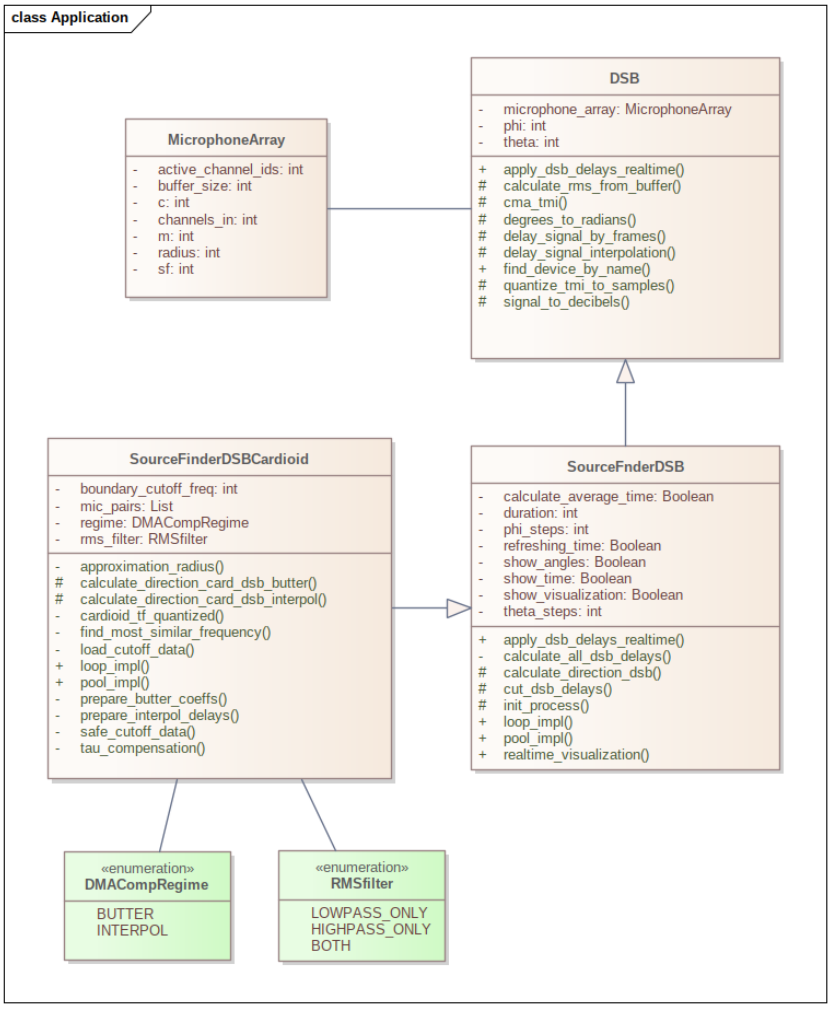
[19] *FACTS ABOUT SPEECH INTELLIGIBILITY* Online. DPA Microphones, Březen 2021. [cit. 2024-1-1]. Dostupné z: https://www.dpamicrophones.com/mic-university/facts-about-speech-intelligibility.

[20] *Interference* Online. Study mind, b. d. [cit. 2024-8-5]. Dostupné z: https://studymind.co.uk/notes/interference/.

Přílohy

Příloha A

Diagram tříd



Obrázek A.1: Diagram tříd

Příloha B

Seznam zkratek

DSB	Delay and Sum Beamforming
DMA	Differential Microphone Array
CMA	Cicular Microphone Array
WAV	Waveform Audio File Format
GIL	Global Interpreter Lock
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
MEMS	Micro-Electro-Mechanical System
USB	Universal Serial Bus
DAW	Digital audio workstation
PoC	Proof of Concept
RMS	Root Mean Square
MAE	Mean Absolute Error
JSON	JavaScript Object Notation
GUI	Graphical User Interface

Příloha C

Odkaz na zdrojový kód

Zdrojový kód je dostupný jak v příloze, tak v následujícím GitLab repozitáři:

https://gitlab.fel.cvut.cz/helmima1/bp-zpracovani-signalu-z-mikrofonnich-poli-s-digitalnimi-mems-mikrofony

Příloha D

Návod na spuštění skriptů

Skripty jsou nakonfigurované tak, aby zpracovávali signál přímo z prototypu cirkulárního mikrofonního pole vyvinutého na FEL ČVUT [1] [2]. Prototyp posílá signál v podobě 16 kanálového výstupu, kdy aktivní jsou pouze kanály uvedené v tabulce D.1. Označení mikrofonů je vidět na obrázku D.1. Skript je upravený přímo pro tuto konfiguraci, při použití jiného mikrofonního pole by bylo třeba upravit kód. Pro nalezení a přiřazení mikrofonního pole jako vstupního zařízení je pak vytovřena funkce `find_device_by_name`, která hledá vstupní zařízení s názvem "FPGA MicArrayBoard: USB Audio (hw:2,0)".

Kanál	Mikrofon
0	L1
1	L2
2	L3
3	L4
4	-
5	-
6	-
7	-
8	P1
9	P2
10	P3
11	P4
12	-
13	-
14	-
15	-

Tabulka D.1: Přiřazení kanálů k mikrofonům. Pole označené "-"značí kanály, které nejsou připojené k žádnému mikrofonu.

Obrázek D.1: Označení jednotlivých mikrofonů v prototypu mikrofonního pole L1-L4 a P1-P4 (popsáno černou fixou).

D.1 Spuštění skriptu s fixním směrem

Skript, který implementuje aplikaci DSB zpoždění na signál v reálném čase, je pojmenovaný `Fixed_direction.py` a třídy jsou definované v souboru `ClassArchitecture.py`. V souboru si lze nastavit následující proměnné definující průběh skriptu:

- **DURATION**: Čas v sekundách, po který má aplikace běžet.
- **THETA**: Úhel θ, kterým má být natočený hlavní lalok.
- **PHI**: Úhel φ, kterým má být natočený hlavní lalok.
- **PRINT_STATUS**: Pokud dojde k přetečení bufferu (buffer overflow), tak informaci vypíše do konzole.
- **OUTPUT_DEVICE_ID**: Identifikátor výstupního zařízení, do kterého má jít výstup s aplikovaným DSB. Identifikátor lze zjistit pomocí knihovny `sounddevice` zavoláním metody `print(sd.query_devices())`.

D.2 Spuštění skriptu s algoritmem pro lokalizaci nejhlasitějšího zdroje zvuku

Skript, který implementuje algoritmus pro lokalizaci nejsilnějšího zdroje v okolí, je pojmenovaný `Loudest_Source_Finder.py` a třídy jsou definované v souboru `ClassArchitecture.py`. V následujících podkapitolách jsou rozepsané proměnné, které definují průběh aplikace.

D.2.1 Režimy běhu aplikace

Veškeré inicializační proměnné jsou uvedené v hlavičce souboru v sekci "MODIFYING VARIABLES". Proměnná `APP_REGIME` má 4 stavy definované třídou `AppRegime` typu `Enum` v souboru `ClassArchitecture.py`:

- **DSB_LOOP**: Použití pouze DSB algoritmu pomocí iterace.
- **DSB_POOL**: Použití pouze DSB algoritmu s paralelizovanými výpočty na hledání směru pomocí `multiprocessing.Pool`.
- **DSB_CARD_LOOP**: Použití kombinace DMA prvního řádu a DSB - hledání směru pomocí iterace.
- **DSB_CARD_POOL**: Použití kombinace DMA prvního řádu a DSB - paralelizace výpočtů pro hledání směru pomocí `multiprocessing.Pool`.

Pro režimy lokalizaci nejsilnějšího zdroje zvuku pomocí DMA prvního řádu lze pak ještě rozlišit mezi dvěma způsoby kompenzace přenosové funkce, jak je popsáno v kapitole 5.4. Toho lze docílit nastavením proměnné `DMA_COMP_REGIME` , která je definovaná třídou `DMACompRegime` typu `Enum` a má 2 možné stavy:

- **INTERPOL**: Kompenzace přenosových funkcí pomocí lineární interpolace.
- **BUTTER**: Kompenzace přenosových funkcí pomocí butterworthova lowpass filtru a násobení skalárem.

D.2.2 Analytické proměnné

Pro analýzu algoritmů je možné definovat následující proměnné typu `boolean`:

- **CALCULATE_AVERAGE_TIME**: Počítá průměrný čas počítání směru za celý běh aplikace. Ten po doběhnutí programu vypíše do konzole.
- **PRINT_STATUS**: Pokud dojde k přetečení bufferu (buffer overflow), tak informaci vypíše do konzole.
- **SHOW_TIME**: Ukazuje čas počítání směru po provedení každého výpočtu směru výpočetním vláknem. Slouží ke kontrole konzistence času výpočtu, případně detekce anomálií.
- **SHOW_ANGLES**: V reálném čase tiskne do konzole úhel φ s nejvyšší detekovanou hlasitostí.

Pro účel analýzy algoritmu pro lokalizaci nejsilnějšího zdroje zvuku pomocí DMA prvního řádu je možné jeho výstup nastavit tak, aby byl zpracovaný signál pouze z nízkého nebo vysokého frekvenčního pásma rozděleného butterworthovým filtrem. Toto rozdělení usnadňuje testování funkčnosti DMA prvního řádu na nízkých frekvencích. Toto rozdělení je možné definovat v proměnné `RMS_FILTER`, která je definovaná třídou `RMSFilter` typu `Enum` a má 3 možné stavy:

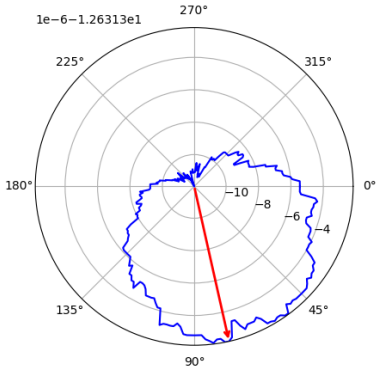
- **LOWPASS_ONLY**: Určování směru s nejsilnějším zdrojem zvuku probíhá pouze pomocí DMA prvního řádu na nízkých frekvencích.
- **HIGHPASS_ONLY**: Určování směru nejsilnějším zdrojem zvuku probíhá pouze pomocí DSB na vysokých frekvencích.
- **BOTH**: Určování směru s nejsilnějším zdrojem zvuku probíhá kombinací DSB a DMA prvního řádu na nízkých i vysokých frekvencích. Toto by mělo být základní nastavení pro praktické využití.

D.2.3 Vizualizace

Vizualizace je vytvořená za pomocí interaktivního grafu v Matplotlib, který je nastavený následujícími proměnnými:

- **SHOW_VISUALIZATION**: Proměnná typu `boolean`, určuje zda má být zobrazená vizualizace v reálném čase či nikoliv.
- **VISUALIZATION_REFRESH_TIME**: Čas, který má vizualizační proces počkat po vykreslení dat. Slouží k umělému zpomalení programu stejně jako proměnná COMPUTATION_THREAD_REFRESH_TIME.

Pokud jsou hodnoty na ose y v polárním grafu velmi podobné, tj. mají mnoho společných desetinných míst, tak se na osách polárního grafu zobrazí pouze poslední odlišné desetinné cifry. Jak moc je číslo pak oseknuté, je zobrazeno v levém horním rohu, jak je vidět na obrázku D.2. Bylo vypozorováno, že k tomuto jevu dochází nejčastěji při velmi malých hlasitostech zvuků v okolí.



Obrázek D.2: Vizualizace s velmi podobnými naměřenými hlasitostmi v okolí. V levém horním rohu je vidět společná část os y.

D.2.4 Ostatní proměnné

Další parametry běhu programu určují následující proměnné:

- **DURATION**: Čas v sekundách, po který má aplikace běžet.
- **OUTPUT_DEVICE_ID**: Identifikátor výstupního zařízení, do kterého má jít výstup s aplikovaným DSB. Identifikátor lze zjistit pomocí knihovny `sounddevice` zavoláním metody `print(sd.query_devices())`.
- **PHI_STEPS**: Množství úhlů φ v prostoru $(0, 360)$ stupňů. Pro režim lokalizace nejhlasitějšího zdroje pomocí DMA prvního řádu toto číslo musí být dělitelné 8 beze zbytku.
- **THETA_STEPS**: Množství úhlů θ v prostoru $(0, 90)$ stupňů. Tato proměnná může v momentální verzi nabývat pouze hodnoty 1, pro více hodnot zatím nebyla implementována. Její implementace může být zaměřením budoucího vývoje, viz kapitola 6.3.
- **COMPUTATION_THREAD_REFRESH_TIME**: Obnovovací čas pro výpočetní vlákno. Pokud je čas výpočtu delší než definovaný čas, tak je tato proměnná zanedbána. Slouží pro umělé zpomalení algoritmu, aby se tím šetřil výkon počítače pro další úkony, případně se předcházelo přehřívání.