Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informačné zdroje v súlade s Metodickým pokynom o dodržiavaní etických princípov pri príprave vysokoškolských záverečných prác.

V Prahe 15. mája 2018

Matej Oravec

Poďakovanie

Ďakujem pánovi Profesorovi za schopnosť poradiť v každej situácii, vytvorenie vysoko nadštandardných podmienok pre prácu nielen na tomto texte a najmä jeho nevídanú ochotu a ľudský prístup.
Vaškovi ďakujem za cenné rady, precíznu korekciu a *výbornú* kávu.

Abstrakt

Presné meranie rýchlosti je nepochybne nenahraditeľnou disciplínou, nachádzajúcou svoje uplatnenie mimo iného v doprave a navigácii. Zatiaľčo pri informatívnom charaktere výsledkov, aké zobrazuje napríklad bežný tachometer v automobile, nie je veľká presnosť nevyhnutná, nájdeme i aplikácie, kde na presnosti naopak veľmi záleží – ako príklady možno uviesť vyhodnocovanie dopravných nehôd či doplnkový senzor v systéme určovania polohy, ktorý môže tvoriť krytie družicového systému pri výpadku signálu.

Jednou z dávno známych metód merania rýchlosti je meranie dopplerovským radarom. Hoci ide o jednoduchú, dobre preskúmanú a mnohými výhodami disponujúcu myšlienku s rokmi overenou funkčnosťou, nie je bežné využiť ju pre meranie malých rýchlostí. V tejto práci sa budeme venovať spracovaniu signálu z komerčného radarového senzoru práve s dôrazom na to, aby bolo možné spoľahlivo určovať rýchlosti blízke nule. Ako výsledok mnohých pokusov uvedieme návrh softvéru, ktorý bude možné využiť pre samostanú jednotku, určujúcu rýchlosť v reálnom čase.

Abstract

In the field of transport and navigation, accurate velocity estimation acts as one of the fundamental needs. There are applications like standard personal vehicle speedometer, in which there is no need for high accuracy according to their mainly informative purpose. However, there are plenty of them in which high confidence is crucial, for example traffic accident analysis or realization of backing mechanism for satellite navigation system, which meets serious difficulties underground or indoor.

One of well-inspected velocity estimation methods known for many years is the one based on *Doppler shift*. Even though employment of frequency shift tends to be deeply explored, it is roughly uncommon to utilize this concept in measurement of *very low* velocities. In this thesis, we will focus on multiapproach processing of signal from commercial radar sensor, putting emphasis specifically on accuracy at the lowest velocity levels. As a result, we introduce software that could be utilized later on stand-alone radar-based speedometer.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam použitých nahrávok

$\mathcal{N}_1$ n1.wav
$\mathcal{N}_2$ n2.wav
$\mathcal{N}_3$ n3.wav
$\mathcal{N}_4$ n4.wav
$\mathcal{N}_5$ n5.wav
$\mathcal{N}_6$ n6.wav
$\mathcal{N}_7$ n7.wav
$\mathcal{N}_8$ n8.wav
$\mathcal{N}_9$ n9.wav
$\mathcal{N}_{10}$ n10.wav
$\mathcal{N}_{11}$ n11.wav
$\mathcal{N}_a$ . . . . . na.wav
$\mathcal{N}_b$ . . . . . nb.wav
$\mathcal{N}_c$ . . . . . nc.wav
$\mathcal{N}_d$ . . . . . nd.wav
$\mathcal{N}_e$ . . . . . ne.wav
$\mathcal{N}_{X1}$ nx1.wav
$\mathcal{N}_{X2}$ nx2.wav
$\mathcal{N}_{X3}$ nx3.wav
$\mathcal{N}_{m1}$ nm1.wav
$\mathcal{N}_{m3}$ nm3.wav

Nahrávky sú súčasťou archívu `nahravky.tar.gz`.

KAPITOLA 1

Princíp dopplerovského radaru

1.1 Pojmy a história

Pojem *radar* značí vo všeobecnosti zariadenie, určujúce skúmanú veličinu pomocou elektromagnetického vlnenia, šíriaceho sa prostredím. Z etymologického hľadiska pochádza toto slovo z anglickej skratky, znamenajúcej *radio detection and ranging*. Z nej iste nevyplýva žiadna nutnosť jeho použitia na meranie rýchlosti či využitia Dopplerovho efektu.

Prvý radarový systém uviedol do prevádzky *Sir Robert Wattson-Watt* v roku 1935 [17], no v tomto prípade nešlo o radar *dopplerovský* – tento vznikol až o niekoľko rokov neskôr.

1.2 Fyzikálny princíp a obmedzenia

V prvej kapitole si priblížime vlastný mechanizmus, stojaci za možnosťou určenia rýchlosti zo zmeny frekvencie navrátenej elektromagetickej vlny, odrazenej od pohybujúceho sa objektu. Hoci v názve práce hovoríme iba o *dopplerovskom meraní rýchlosti*, je nutné podotknúť, že dopplerovské meranie rýchlosti sa uskutočňuje v bežnej praxi dvoma rôznymi typmi radaru, ktorými sú

Radar s kontinuálnou vlnou, označovaný bežne ako *CW-radar*[i], ktorý využíva súčasné vysielanie vlny o istej nominálnej frekvencii a príjem vlny odrazenej, ktorej frekvencia je pozmenená[ii]. Senzor IVS-167 je práve prípadom takéhoto radaru a v ďalšom texte budeme označením *dopplerovský radar*, prípadne *radar* myslieť vždy radar s kontinuálnou vlnou.

Pulzný radar, ktorý vyšle pulz vlny o istej dobe trvania a čaká, kým je tento pulz odrazený a navrátený do antény. Radar na takomto princípe je hojne používaný v oblasti riadenia leteckých spojov alebo military aplikáciach.

[i] Z anglického *continous wave*.

[ii] V prípade nulovej rýchlosti odrazného objektu bude pozmenená o 0 Hz.

Označme nominálnu frekvenciu vlny, vysielanej vysielacou anténou radaru ako f_0. Pokiaľ sa niekde v akčnom rádiuse[iii] tejto vlny nachádza objekt, pohybujúci sa rýchlosťou v_x po spojnici s osou hlavného laloku vysielacej antény[iv] a anténa vysiela signál

$$s_T(t) = A_T \cos(2\,\pi\, f_0\, t)\,, \tag{1.1}$$

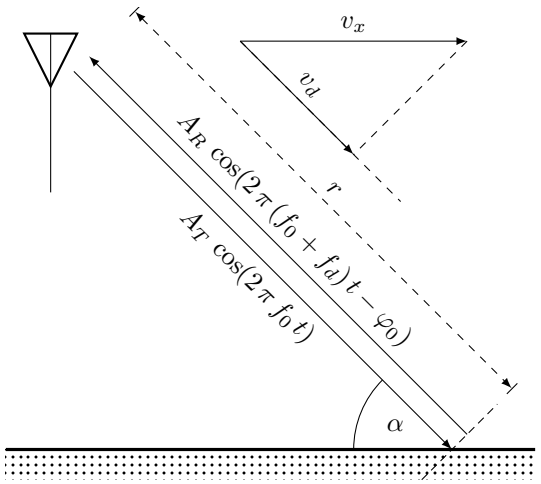
bude frekvencia odrazenej vlny pozmenená a prijímacia anténa bude prijímať signál

$$s_R(t) = A_R \cos(2\,\pi\,(f_0 + f_d)\, t - \varphi_0)\,, \tag{1.2}$$

kde

f_d je tzv. *dopplerovská frekvencia*, teda frekvenčný posun, spôsobený vzájomným pohybom vysielača a plochy, od ktorej je signál odrazený,

φ_0 je zmena fáze vysielanej vlny, spôsobená trajektóriou jej šírenia, ktorej dĺžka nie je celočíselným násobkom vlnovej dĺžky. Toto je sčasti ilustrované na obrázku 1.1



Obr. 1.1: Usporiadanie radaru pri meraní rýchlosti. Pokiaľ má vyslaná vlna s_T vlnovú dĺžku λ_T a platí $r \neq k\,\lambda_T$ pre $k \in \mathbb{N}_0$, odrazená vlna bude od odraznej plochy „vyslaná“ s nenulovou počiatočnou fázou φ_0 i v stave bez pohybu ($v_d = v_x = 0$). Rýchlosť v_x je rýchlosťou v priamom smere, ktorú skúmame, rýchlosť v_d potom jej priemetom na spojnicu zdroja vlny a pomyselného bodu, v ktorom nastane jej odraz.

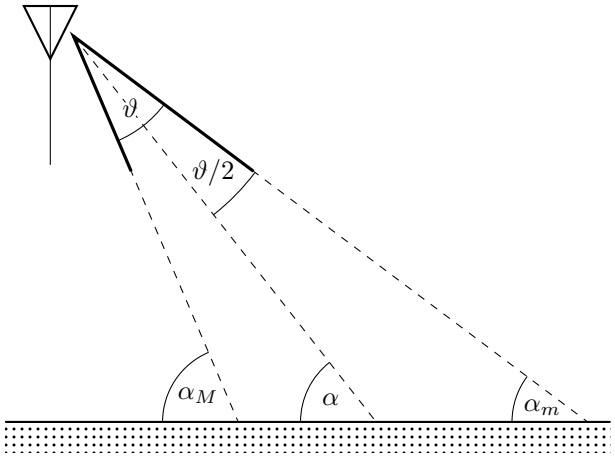
Pre dopplerovskú frekvenciu f_d platí

[iii] Danom šírkou zväzku vysielacej antény, vysielaným výkonom a elektromechanickými parametrami prostredia a odraznej plochy.

[iv] Pohyb samozrejme nemusí byť pre platnosť uvedených vzťahov konaný po spojnici s radarom; V takom prípade možno hovoriť o všeobecnej rýchlosti $\vec{v}$ a za rýchlosť v_x prehlásime *kolmý priemet* vektoru $\vec{v}$ na spojnicu pohybujúceho sa objektu a radarového senzoru

$$f_d = 2\,\frac{v_d}{\lambda_T} = 2\,\frac{v_x}{\lambda_T}\cos(\alpha)\,, \tag{1.3}$$

Podrobné odvodenie tohoto vzťahu je možné nájsť napríklad v [1]. Uhol α je uhol, ktorý zviera spojnica radaru a pohybujúceho sa objektu s vektorom rýchlosti tohoto objektu. Pri prevedení radaru pre meranie rýchlosti vozidiel, ktorým sa zaoberáme, je pohybujúcim sa objektom *povrch zeme*, čo napomáha odôvodniť doteraz možno trochu zavádzajúci náčrt na obrázku 1.1. Vzhľadom na to, že povrch zeme nie je dokonale rovný a vyžarovacia charakteristika antén nie je dokonale úzka, je elevačný uhol α v čase v istých medziach premenný. Vplyv nenulovej šírky anténneho zväzku na elevačný uhol je ilustrovaný na obrázku 1.2.



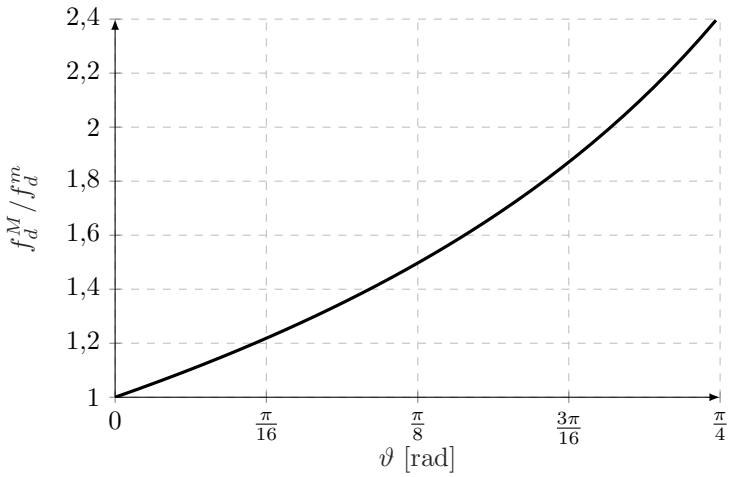
Obr. 1.2: Vplyv nenulovej šírky vyžarovacej charakteristiky antén na pracovný elevačný uhol.

Frekvencia odrazeného signálu potom môže byť oproti frekvencii signálu vyslaného posunutá v rozmedzí f_d^m až f_d^M, čo vyplýva z rovnice (1.3). Pokiaľ označíme šírku zväzku antény ϑ, bude pomer maximálneho a minimálneho dopplerovského posunu určený vzťahom

$$\frac{f_d^M}{f_d^m} = \frac{\frac{2\,v_x}{\lambda_T}\cos(\alpha - \frac{\vartheta}{2})}{\frac{2\,v_x}{\lambda_T}\cos(\alpha + \frac{\vartheta}{2})} = \frac{\cos(\alpha - \frac{\vartheta}{2})}{\cos(\alpha + \frac{\vartheta}{2})}\,. \tag{1.4}$$

Závislosť z rovnice (1.4) je vynesená na obrázku 1.3. Vidíme, že pri šírke hlavného laloku $\frac{\pi}{4}$ je pomer maximálnej a minimálnej dopplerovskej frekvencie väčší než 2, čo je samozrejme pre presné určenie rýchlosti neakceptovateľné. Pre čo najväčšie „uľahčenie práce“ algoritmom spracovania je preto nutné zabezpečiť veľmi úzky hlavný lalok vyžarovacej charakteristiky antén a veľké potlačenie lalokov postranných. Veríme, že nakoľko ide o požiadavky celkom štandardné, výrobca senzoru potrebné parametre antén zabezpečil. Správnosť tohoto predpokladu dokazujú okrem datasheetu i výsledky meraní rýchlosti.

Analogický vplyv na jednoznačnosť určenia dopplerovského posunu má, celkom prirodzene, i nenulová šírka zväzku v rovine *horizontálnej*. Výsledky jej



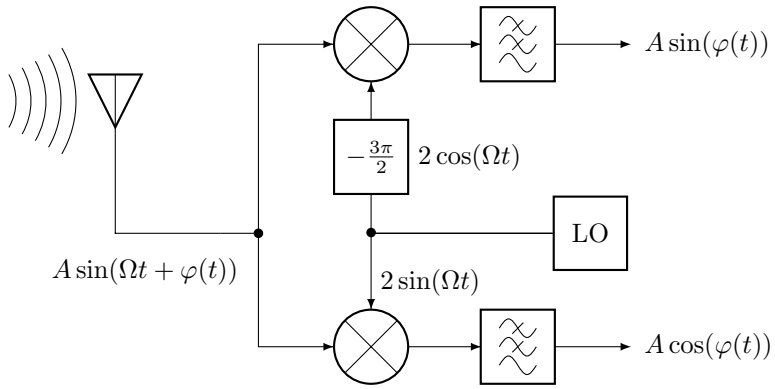
Obr. 1.3: Závislosť pomeru maximálnej a minimálnej dopplerovskej frekvencie na šírke anténneho zväzku vo vertikálnej rovine. Obrázok odpovedá uhlu náklonu radaru $\alpha = \pi/4$.

vplyvu na rozsah dopplerovských frekvencií na prijímacej strane sú podrobne rozoberané v [1] a pre túto prácu nepovažujeme za prínosné zaoberať sa nimi.

1.3 Rozlíšenie smeru pohybu

Z merania rozdielu frekvencií vyslaného a prijatého signálu je možné vo všeobecnosti určiť absolútnu hodnotu i smer pohybu pohybujúceho sa objektu – frekvencia, odrazená od vzďaľujúceho sa objektu je nižšia, než frekvencia vysielaná a od približujúceho sa objektu sa naopak vráti frekvencia vyššia. Problém však nastáva s realizáciou takéhoto merania pre náš účel. V radarovom senzore sa totiž nenachádza prístroj na meranie absolútnej hodnoty prijatej frekvencie – je tam iba zmiešavač, ktorý dokáže určiť *rozdielovú fekvenciu*. Pre frekvencie navrátené od objektov, ktoré sa pohybujú v smere od radaru rýchlosťou $v_{\rightarrow}$ a frekvencie, odrazené od objektov, pohybujúcich sa v smere k radaru rýchlosťou $v_{\leftarrow} = -v_{\rightarrow}$ teda rozdielové frekvencie splynú, čím nastáva strata informácie.[v] Možnosťou, ako je tento problém možné riešiť, je použiť *kvadratúrne zmiešavanie*. Pre vysvetlenie začnime obrázkom 1.4.

[v] Ide o situáciu analogickú s problémom zrkadlových frekvencií pri návrhu rádiových prijímačov.



Obr. 1.4: Schéma dvojkanálového superheterodynného prijímača.

Radar vyšle všeobecný harmonický signál $s_T(t) = A\,\sin(\Omega t)$. Tento sa odrazí od objektu a v prípade, že sa tento pohybuje, fáza signálu s_T sa posunie v zmysle Dopplerovho javu. Prijímacia anténa teda príjme signál[vi]

$$s_R(t) = A\,\sin(\Omega t + \varphi(t))\,. \tag{1.5}$$

Na konverziu tohoto signálu do základného pásma je využitý lokálny oscilátor, kmitajúci uhlovou frekvenciou, rovnou uhlovej frekvencii signálu $s_T(t)$, pričom jeho výstup je rozdelený do dvoch vetiev, vzájomne posunutých o $\frac{\pi}{2}$ rad[vii]. Výstupy zmiešavačov budú vyzerať takto:

$$\begin{aligned}
I(t) &= A\,\sin(\Omega t+\varphi(t))\cdot 2\,\sin(\Omega t) = 2\,A\,\sin(\Omega t+\varphi(t))\,\sin(\Omega t) &=\\
&= 2\,A\,\cos(\varphi(t))\ \underbrace{\sin^2(\Omega t)}_{\frac{1}{2}(1-\cos(2\Omega t))}\ +\underbrace{2\,\cos(\Omega t)\,\sin(\Omega t)}_{\sin(2\Omega t)}\cdot A\,\sin(\varphi(t)) &=\\
&= A\,\cos(\varphi(t)) - \underbrace{A\cos(\varphi(t))\,\cos(2\Omega t) + 2\,A\,\sin(2\Omega t)\,\sin(\varphi(t))}_{\text{Odstránime pomocou DP}} &=\\
&= A\,\cos(\varphi(t))
\end{aligned} \tag{1.6}$$

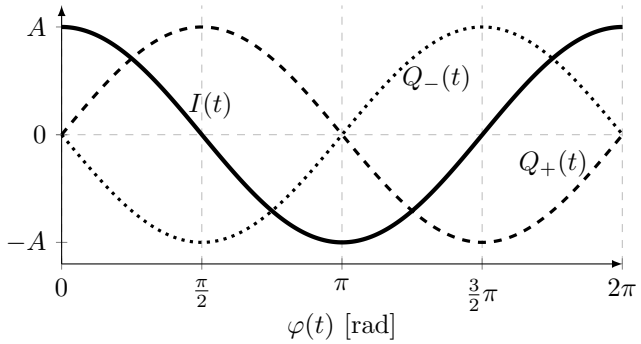
$$\begin{aligned}
Q(t) &= A\,\sin(\Omega t+\varphi(t))\cdot 2\,\cos(\Omega t) = 2\,A\,\sin(\Omega t+\varphi(t))\,\cos(\Omega t) &=\\
&= 2\,A\,\sin(\varphi(t))\ \underbrace{\cos^2(\Omega t)}_{\frac{1}{2}(1+\cos(2\Omega t))}\ +\underbrace{2\,\cos(\Omega t)\,\sin(\Omega t)}_{\sin(2\Omega t)}\cdot A\,\cos(\varphi(t)) &=\\
&= A\,\sin(\varphi(t)) + \underbrace{A\sin(\varphi(t))\,\cos(2\Omega t) + 2\,A\,\sin(2\Omega t)\,\cos(\varphi(t))}_{\text{Odstránime pomocou DP}} &=\\
&= A\,\sin(\varphi(t))
\end{aligned} \tag{1.7}$$

Pretože kosínus je funkciou párnou, platí $\cos(-\varphi(t)) = \cos(\varphi(t))$ a súfázová zložka nebude znamienkom $\varphi(t) = 2\,\pi\,f_d\,t$ ovplyvnená. Naproti tomu ale vďaka nepárnosti sínu platí $\sin(-\varphi(t)) = -\sin(\varphi(t))$ a fáza kvadratúrnej zložky $Q(t)$

[vi] Predpokladajme teraz dokonalú situáciu, kedy nastane dokonalý odraz a signál nebude pri prechode prostredím tlmený.

[vii] Hoci na obrázku 1.4 je znázornený posun o $\frac{3\,\pi}{2}$ rad, je jednoduché si uvedomiť, že táto hodnota je závislá iba na voľbe signálu lokálneho oscilátoru medzi sínom a kosínom.

sa so znamienkom dopplerovského posunu f_d mení o π rad. Toto je ilustrované na priebehoch jednotlivých signálových zložiek na obrázku 1.5, kde $Q_-(t)$ značí kvadratúrnu zložku pre $f_d < 0$ a $Q_+(t)$ túto pre $f_d > 0$. Zo spektra komplexných vzoriek takéhoto signálu, kde reálna časť vzorky je tvorená vzorkou súfázovej zložky a imaginárna časť vzorkou kvadratúrnej zložky teda dokážeme rozlíšiť smer pohybu.



Obr. 1.5: Priebehy súfázovej a kvadratúrnej zložky prijatého signálu

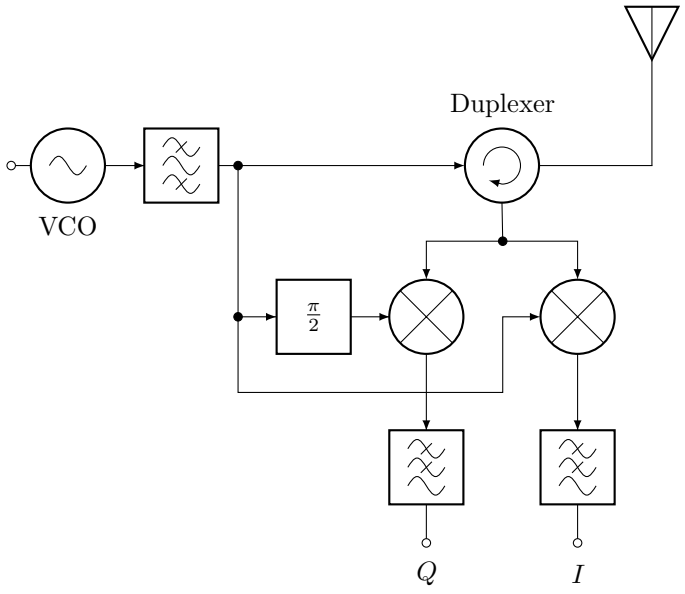
KAPITOLA 2

Prototyp radaru

Pri testovaní navrhnutých algoritmov sa nezaobídeme bez reálneho radaru. V tomto prípade pracujeme s prototypom jednoduchého zapojenia, ktorého jadro tvorí komerčne dostupný radarový senzor *IVS-167* od firmy *InnoSenT*. Kompletný popis tohoto zapojenia uvedieme v nasledujúcej kapitole.

2.1 Radarový senzor IVS-167

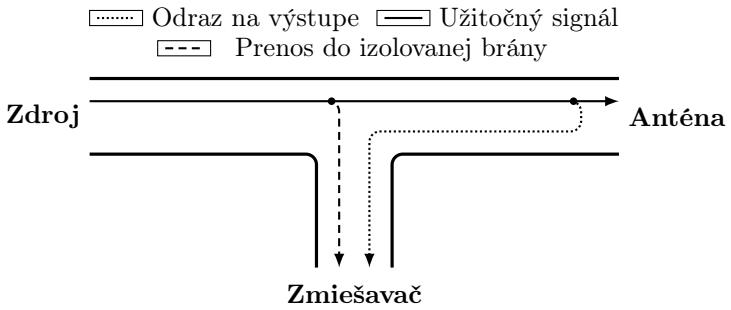
Schéma samotného senzoru je uvedená na obrázku 2.1.



Obr. 2.1: Schéma radarového senzoru IVS-167 [10].

Namiesto samostatnej vysielacej a prijímacej antény je tu použité zapojenie s *duplexerom*, čo je vo všeobecnosti obvod, umožňujúci obojsmernú komuniká-

ciu, využívajúc jedinú cestu [14]. V datasheete [10] nie je vlastné prevedenie duplexeru popísané, vzhľadom na rozmery a cenu senzoru však pravdepodobne pôjde o jedno z mikropáskových prevedení. Jednoznačnou výhodou takéhoto zapojenia je ušetrenie jednej antény. Nevýhodou je možné vzájomné ovplyvnenie prijímaného signálu signálom vysielaným[i], čo je spôsobené nedokonalosťami duplexeru. Tieto sú znázornené na obrázku 2.2. Pri testovaní algoritmov spracovania signálu budeme predpokladať dokonalý duplexer, nakoľko zmeny senzoru nie sú predmetom tejto práce. V kapitole 3 uvidíme, že tieto nedokonalosti nemusia byť zanedbateľné.



Obr. 2.2: Znázornenie možných problémov pri použití duplexeru a jedinej antény ako náhrady dvoch samostatných antén.

Okrem zdroja primárnej vlny, pásmovej priepusti, duplexeru a antény obsahuje radarový senzor štandardné zapojenie kvadratúrneho zmiešavača, ktorého výstupy I a Q sú doplnené príslušnými filtrami typu dolná priepusť.

Primárna vysielaná vlna má typickú frekvenciu $24,125$ GHz, ktorá je platná pri teplote $\theta_0 = 25$ °C. Frekvencia pri teplote θ je na základe údajov v [10] daná vzťahom

$$f(\theta) = 24,125 - 10^{-3} \cdot (\theta - \theta_0) \text{ GHz}\,. \tag{2.1}$$

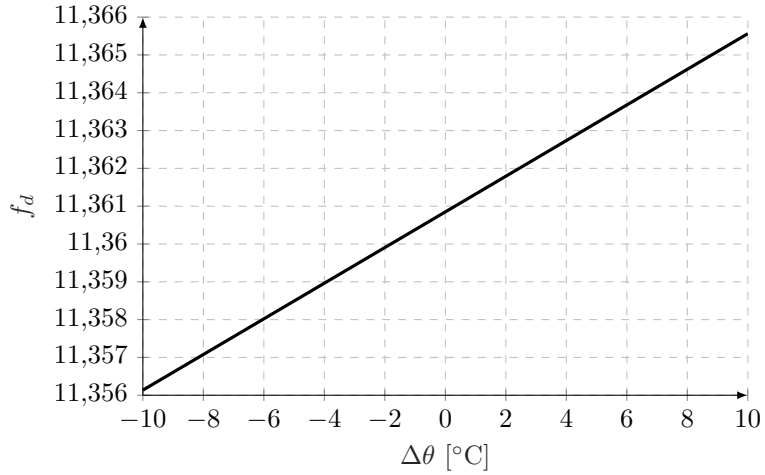
Predpokladajme, že merania prebiehajú pri teplote o niečo nižšej[ii] než je *normálna* teplota θ_0 a skúmajme možnosť ovplyvnenia výsledkov týmto faktom. Použitím vzťahov (1.3) a (2.1) dostávame lineárnu závislosť

$$f_d(\Delta\theta) = \overbrace{\frac{2\,v_x\,\cos(\alpha)}{c}}^{\text{ozn. } \zeta} f(\theta_0 - \Delta\theta) = \zeta\,(24,125 - 10^{-3} \cdot (25 - \Delta\theta)) \cdot 10^9\,, \tag{2.2}$$

ktorá je vynesená na obrázku 2.3. Vidíme, že zmena o približne 8 °C spôsobí zmenu dopplerovskej frekvencie f_d o približne $0,004$ Hz, čo odpovedá zmene rýchlosti v_x o cca. 35 μm/s. Táto zmena je tak malá, že teplotný drift výstupnej frekvencie radarového senzoru možno bez problémov zanedbať.

[i] V opačnom smere je toto ovplyvnenie iste možné tiež; Vzhľadom na nízku výkonovú úroveň signálu prijímaného ho však možno zanedbať.

[ii] Väčšina meraní prebiehala pri odhadovanej teplote $\theta = 22$ °C.



Obr. 2.3: Závislosť dopplerovskej frekvencie f_d na zmene teploty $\Delta\theta$ pri priamej rýchlosti pohybujúceho sa objektu $v_x = 0,1$ m/s.

Pre celistvosť informácií uvádzame dôležité parametre radarového senzoru, prevzaté z datasheetu [10], v tabuľke 2.1.

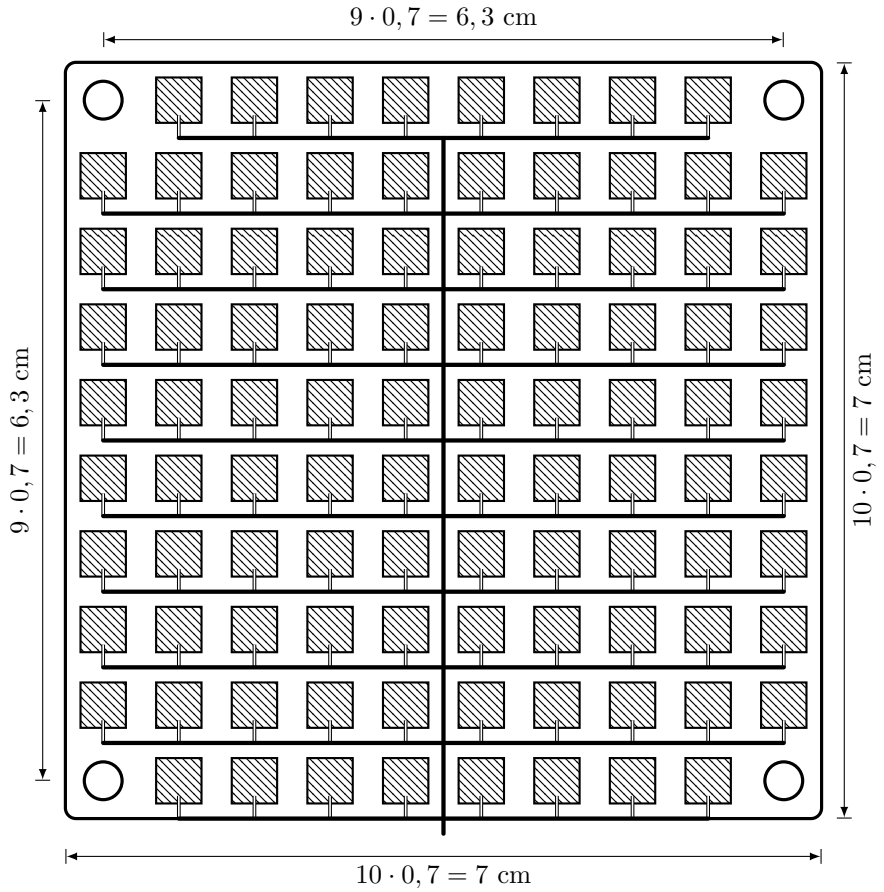
Parameter	**Min.**	**Typ.**	**Max.**
Frekvencia vysielaného signálu[iii] [GHz]	24,1	24,125	24,15
Frekvenčný drift [MHz/°C]		-1	
EIRP [dBm]		18	20
Pomer amplitúd výstupných zložiek [dB]			6
Fázový posun výstupných zložiek [°]	60	90	120
Šírka zväzku pre pokles o 3 dB[iv] [°]		11	
Potlačenie postranných lalokov[v] [dB]		25	
Napájacie napätie [V]	4,75	5	5,25
Napájací prúd [mA]		33	40

Tab. 2.1: Parametre senzoru IVS-167 [10].

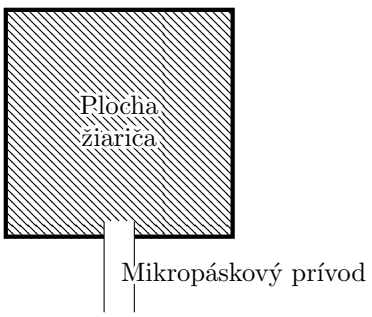
2.1.1 Prevedenie antény

Anténa senzoru je riešená ako anténne pole 10×10 obdĺžnikových elementárnych žiaričov, pričom anténky v rohoch sú vynechané. Schématické náčrty elementárneho žiariča a celého poľa spolu s naznačením mikropáskových vodičov a rozmerov sa nachádzajú na obrázkoch 2.4 a 2.5.

[iii] Pre ladiace napätie 2,5 V a pri teplote 25 °C je udávaná hodnota $f_T = 24,125$ GHz.
[iv] Platí v oboch na seba kolmých rovinách.
[v] Platí v oboch na seba kolmých rovinách.



Obr. 2.4: Schéma anténneho poľa senzoru IVS-167.

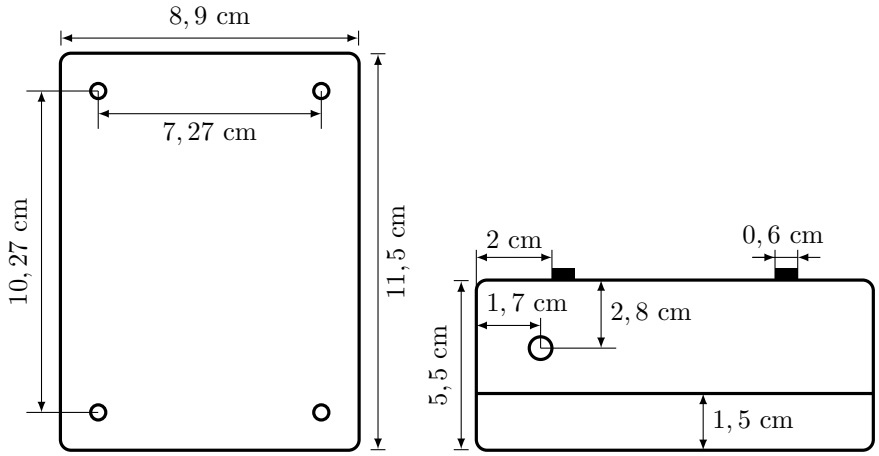


Obr. 2.5: Schéma elementárnej anténky senzoru IVS-167.

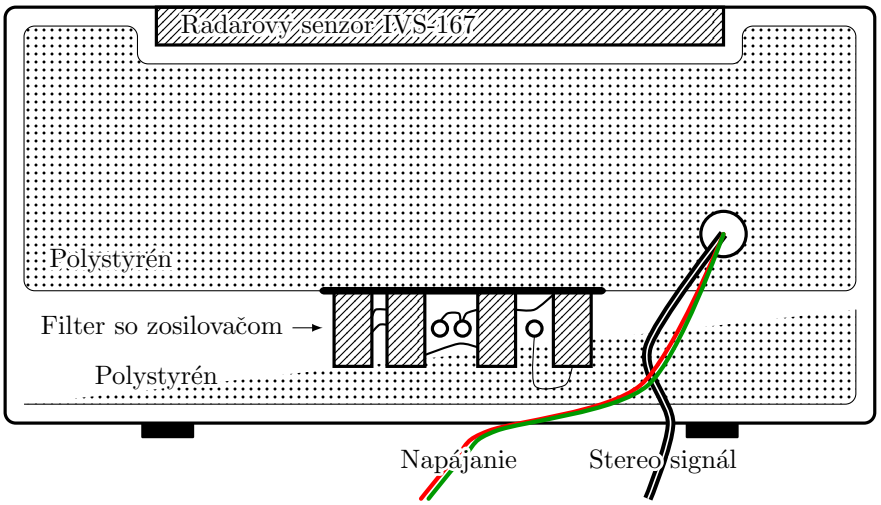
Vyžarovacie vlastnosti tohto poľa sú zhrnuté v práci [1] a v tomto texte sa nimi nebudeme ďalej zaoberať.

2.2 Spracovanie radaru

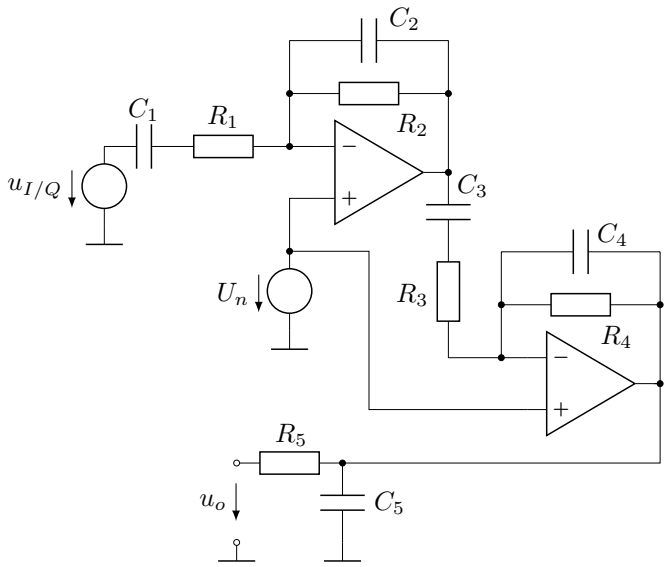
V tejto časti bude zhrnuté kompletné radarové zapojenie, použité na merania, ktoré slúžia ako podklady pre vyhodnocovacie algoritmy. Senzor IVS-167 z časti 2.1 je umiestnený vo vnútri plastovej krabičky, ktorej rozmery a vnútorné usporiadanie sú zhrnuté na obrázkoch 2.6, resp. 2.7.



Obr. 2.6: Rozmery krabičky radarového zapojenia.



Obr. 2.7: Prierez krabičky radarového zapojenia.

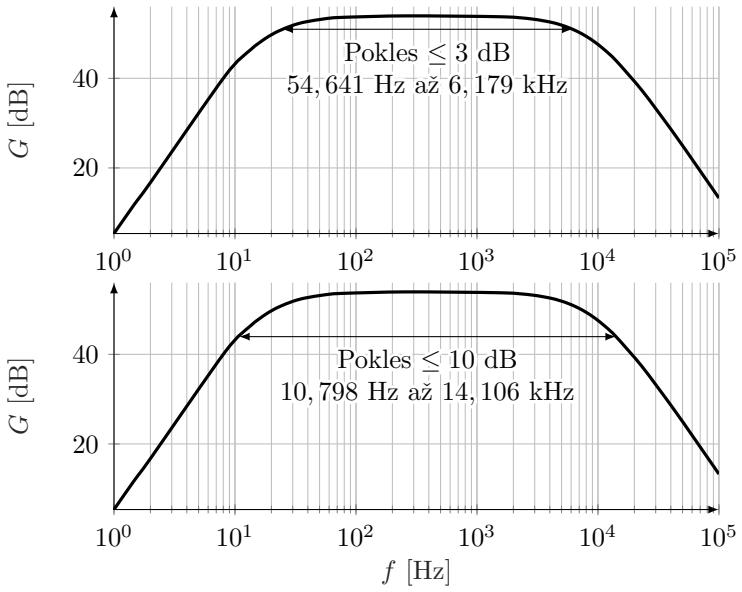


Obr. 2.8: Schéma zapojenia pásmovej priepusti so zosilnením, použitej v radarovom zapojení.

Senzor je prikrytý polystyrénovou ochrannou vrstvou, nad ktorú sú vyvedené signálové vodiče I a Q zložiek a napájacia stočená dvojlinka. Signálové vodiče vstupujú ďalej do zosilovača, filtra a odporového deliča. Použité je zapojenie pásmovej priepusti s istým nenulovým ziskom, ktorého schéma sa nachádza na obrázku 2.8. Použité sú tu *rail to rail* operačné zosilovače *MCP602* od firmy *Microchip*. Hodnoty prvkov tohoto zapojenia sú uvedené v tabuľke 2.2. Vzhľadom na to, že k tomuto filtru bola dokumentácia nedostatočná, obvod sme simulovali pomocou prostredia *Microcap*. Simulovaná modulová charakteristika je zobrazená na obrázku 2.9. Možno z nej odčítať pásmo filtru pre pokles maximálnej výkonovej úrovne o 3 dB, ktoré siaha od 24, 641 Hz do 6, 179 kHz a pásmo pre pokles maximálnej úrovne o 10 dB, ktoré je 10, 798 Hz až 14, 106 kHz.

i	R_i [kΩ]	C_i [nF]	U_n [V]
1	1	10000	
2	10	1, 5	
3	1	10000	5
4	10	1, 5	
5	100	1	

Tab. 2.2: Hodnoty prvkov zapojenia z obrázku 2.8. Vstup $u_{I/Q}$ je signálovým vstupom, napätie U_n je dodávané po zbernici USB.



Obr. 2.9: Modulová frekvenčná charakteristika filtru so ziskom z obrázku 2.8.

Výskumná správa [1] uvádza pásmo tohoto filtru pre pokles maxima o 3 dB ako 22 Hz až 8,2 kHz, čo približne odpovedá našej simulácii. Stanovme si požiadavku na užitočné rozmedzie rýchlostí v_x v priamom smere ako

$$v_x \in \langle 0,05;\ 18 \rangle \text{ m/s}\,. \tag{2.3}$$

Na základe rovnice (1.3) tomu odpovedá rozmedzie dopplerovských frekvencií

$$f_d \in \langle 5,6863;\ 2047,07 \rangle \text{ Hz}\,. \tag{2.4}$$

Zatiaľčo z pravej strany naša požiadavka filtrom obmedzená nie je (a to s rezervou[vi]), z ľavej strany nastáva istý konflikt medzi požiadavkou na dobrú filtráciu nízkofrekvenčného rušenia a dobrým rozlíšením malých frekvencií. Hodnote $f_d = 5,6863$ Hz odpovedá na charakteristike z obrázku 2.9 hodnota útlmu maxima o 18,17 dB, čo je dosť vysoká hodnota. Skutočné správanie pri meraní však ťažko odhadovať.

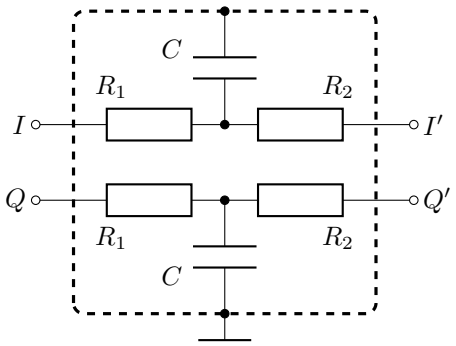
Vzhľadom na to, že najvýznamnejšia zložka nízkofrekvenčného rušenia, teda sieťových 50 Hz, leží i v súčasnom stave v priepustnom pásme tohoto filtru,[vii] je pravdepodobné, že vhodnejším filtrom by bolo možné výsledky celého zapojenia zlepšiť.

2.3 Primárne spracovanie výstupu

Signály z vodičov I a Q, teda súfázová a kvadratúrna zložka o dopplerovskej frekvencii f_d, sú vedené do jednoduchej dolnej priepusti s odporovým deličom

[vi] Dopplerovskej frekvencii 6,179 kHz odpovedá rýchlosť $v_x = 56,28$ m/s.
[vii] Za priepustné pásmo tu pokladáme pásmo útlmu maxima o menej než 3 dB.

ako na obrázku 2.10 a následne na stereo 3, 5 mm jack konektor a tento ďalej do externej zvukovej karty.



Obr. 2.10: Schéma dvojvodičovej dolnej priepusti s odporovým deličom, použitej v radarovom zapojení pred vstupom signálu do konektoru. Hodnoty prvkov nám nie sú z [1] známe.

Ako externá zvuková karta bola použitá karta *Axagon ADA-HP*, ktorej parametre sú uvedené v tabuľke 2.3. Hoci táto karta má možnosť nastavenia niektorých parametrov, z týchto uvádzame iba tie nami použité.

Počet bitov A/D prevodníku	16
Vzorkovacia frekvencia	44, 1 kHz
Odstup signál – šum	≥ 90 dB
Harmonické skreslenie	≤ -80 dB
Výstupná zbernica	USB 2.0

Tab. 2.3: Vybrané parametre použitej zvukovej karty.

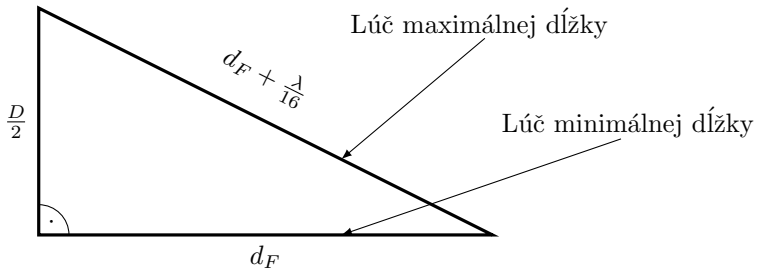
Nahrávanie dát pre testovanie algoritmov spracovania prebiehalo pomocou počítača, do ktorého bol pomocou USB zavedený výstup zo zvukovej karty. Z ďalšieho USB portu počítača bolo potom brané napájanie radarového senzoru a operačných zosilovačov. Po mechanickej inštalácii bol volaný z Matlabu skript E.1, ktorý ukladá WAVE súbor užívateľom zadanej *dĺžky* a *názvu* do preddefinovanej zložky. Všetko ďalšie spracovanie už prebiehalo práve s takto nahratými WAVE súbormi.

2.4 Získavanie testovacích nahrávok

2.4.1 Relevantné umiestnenie radaru

Pretože konkrétne zameranie radarového zapojenia, ktoré by mohlo byť logickým pokračovaním, využívajúcim výsledky tejto práce, nie je dané z hľadiska výšky senzoru nad povrchom zeme ani typu odraznej plochy, snažili sme sa algoritmy vytvárať pokiaľ možno v týchto ohľadoch univerzálne. Jedným z obmedzujúcich faktorov pre výšku nad povrchom je iste polomer *vzdialenej*

zóny[viii] anténneho poľa d_F. Iba vo vzdialenosti od antény väčšej či rovnej d_F sú všeobecne platné vzťahy, na základe ktorých boli pravdepodobne počítané vyžarovacie vlastnosti antény. Podľa [15] je polomer vzdialenej zóny definovaný ako taká vzdialenosť od antény, v ktorej pri aproximácii reálnej vlnoplochy *rovinnou* vznikne na ploche, ktorej maximálny rozmer je D, fázová chyba maximálne $\pi/8$[ix]. Možno sa na to pozerať i z opačnej strany – a je to dokonca prípad názornejší – kedy požadujeme, aby fázový rozdiel medzi dvoma rôznymi lúčmi, vyžiarenými anténou o maximálnej apertúre D bol maximálne $\pi/8$. Potom pre d_F dostávame podľa obrázku 2.11



Obr. 2.11: K odvodeniu vzťahu pre polomer vzdialenej zóny antény. Odvesna vľavo predstavuje polovicu maximálnej apertúry, odvesna vpravo a prepona potom línie k bodu vo vzdialenosti d_F, ktoré majú najväčší vzájomný rozdiel dĺžok. Obrázok je schémou prierezu systému v akejkoľvek rovine, obsahujúcej najdlhšiu úsečku, ktorú možno zostrojiť na ploche, danej okrajom antény.

$$\begin{aligned} \left(d_F + \frac{\lambda}{16}\right)^2 &= d_F^2 + \frac{D^2}{4} \\ d_F &= \frac{2\,D^2}{\lambda}\,, \end{aligned} \tag{2.5}$$

kde sme na ľavej strane rovnice zanedbali člen $\left(\frac{\lambda}{16}\right)^2$. To je možné urobiť, pokiaľ platí $D \gg \lambda$. Pri frekvencii signálu 24,125 GHz a rozmeroch antény podľa obrázku 2.4 je však táto podmienka splnená len veľmi okrajovo a nemožno teda očakávať príliš presný výsledok. Snažíme sa ale získať hlavne výsledok približný, ktorý by nám ukázal rozmedzie vzdialeností antény radaru od odraznej plochy, v ktorých má zmysel merať. Na tento účel je vzťah (2.5) dostatočný.

Po dosadení konkrétnych hodnôt dostávame

$$d_F = \frac{2\,D^2}{\lambda} = \frac{2\,(\sqrt{0,07^2 + 0,07^2})^2}{\frac{3\cdot 10^8}{24,125\cdot 10^9}} \approx 1,58\ \mathrm{m}\,, \tag{2.6}$$

čo je vzdialenosť *príliš veľká* na jednoduché uchytenie aparatúry napríklad pri meraní rýchlosti automobilu. Podľa [1] však možno za prípustnú maximálnu hodnotu fázovej chyby pokladať i $\Delta\varphi_M = \frac{\pi}{4}$, čo polomer vzdialenej zóny zmenší na približne 32 cm [1], čo je vzdialenosť na merania *vhodná*.

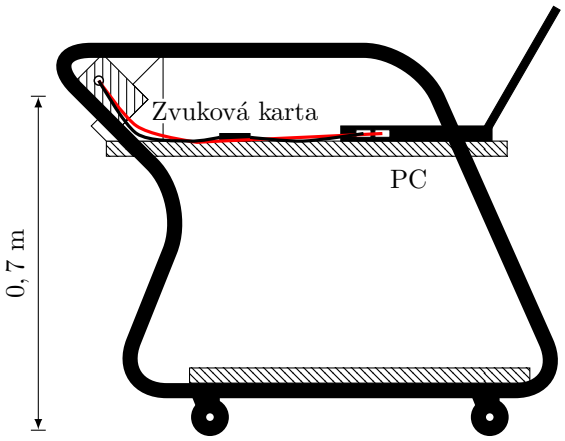
[viii] Používa sa tiež termín *Fraunhoferova vzdialenosť*.

[ix] Väčšina zdrojov uvádza hodnotu požadovanej maximálnej fázovej chyby $\pi/8$, no táto hodnota môže byť samozrejme zvolená rôzne.

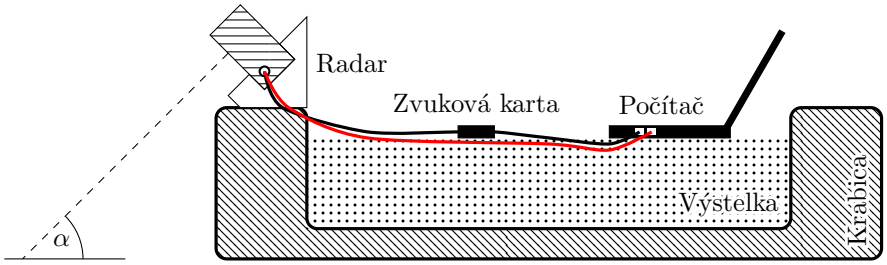
2.4.2 Meracie usporiadania

Pre účely záznamu dát vo forme WAVE súborov boli zvolené tri rôzne usporiadania. Ich opis uvádzame spolu s ich schématickým znázornením.

Prvým z nich bol kolieskový vozík s kovovou klietkou a dvomi policami. Výška tej využitej nad zemou bola $0,7$ m. Vozíkom boli merané nahrávky indoor a to nad dvojitou podlahou, pokrytou linoleom. Využívali sme ho na meranie dát pre rýchlosti od 0 do približne 2 m/s.



Obr. 2.12: Usporiadanie aparatúry pre indoor meranie malých rýchlostí.

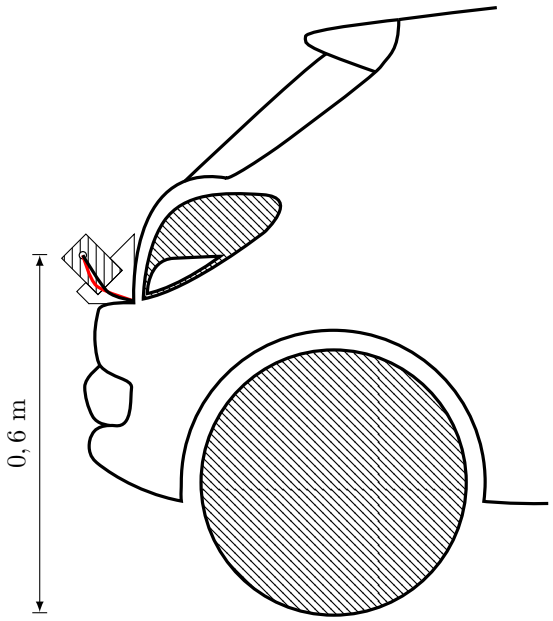


Obr. 2.13: Aparatúra na meranie v blízkej zóne.

Druhým prevedením, slúžiacim na meranie rýchlostí celého skúmaného rozsahu[x] bolo umiestnenie radaru z vonkajšej strany osobného automobilu. Výška umiestnenia nad zemou bola v pokojovom stave približne $0,6$ m. Merania prebiehali nad rôznymi povrchmi ako napríklad suchý a mokrý asfalt, mačacie hlavy, kalužvody[xi].

[x] Teda $0,05$ až 18 m/s.

[xi] Ako sa ukázalo, tieto štandardné povrchy boli pri meraní prakticky indiferentné.



Obr. 2.14: Usporiadanie pre meranie outdoor. Zvuková karta a počítač sú umiestnené vo vnútri vozidla.

Posledné z testovacích usporiadaní slúžilo na pokusy s meraniami s odraznou plochou veľmi blízko antény – výška antény nad zemou bola približne 10 cm. Aparatúra pozostávala z krabice s výstelkou a kusu lana, ktorým sme ju rozpohybovávali[xii]. Použitým povrchom boli keramické kachličky s častými nehomogenitami. Takéto merania slúžili na skúšanie algoritmov v extrémnych situáciách. Zaujímavou vlastnosťou meraní z tohoto usporiadania je, že ako jediné vykazovali úspešné výsledky pre použitie algoritmu z časti 3.4.2.1.

[xii] Na obrázku nie je naznačené.

KAPITOLA 3

Algoritmy vyhodnocovania dát

Prvou úlohou, ktorú bolo na ceste ku komplexnému vyhodnocovaciemu softvéru potrebné vyriešiť, bolo nájsť vhodnú parametrizáciu nameraných dát, určiť spôsoby, ktorými budú jednotlivé parametre vyhodnotené a navrhnúť model, akým budú ich výsledky syntetizované. V nasledujúcej kapitole budú podrobne rozpísané testované postupy. Okrem tých, ktoré sú vo finálnom programe použité, spomenieme i tie, ktorým sme sa venovali, no ich výsledky neboli uspokojivé alebo boli horšie, než výsledky iných algoritmov, určujúcich rovnaký parameter.

Dvomi hlavnými parametrami, ktoré je potrebné z nahrávok získať, sú *absolútna hodnota rýchlosti* a *smer pohybu*. Ich hodnoty sme získavali buď priamym návrhom parametrizácie signálu či jeho spektra, alebo s pomocou jednoduchej *neurónovej siete*, ktorej základné znaky si priblížime v nasledujúcej časti.

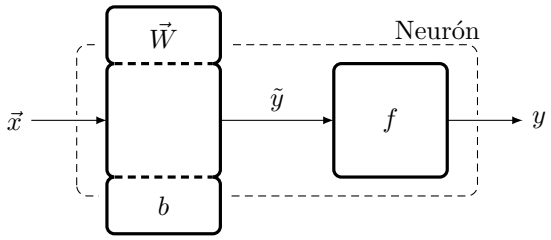
3.1 Neurónová sieť

Neurónová sieť je jedným zo základných nástrojov dnes veľmi populárnej disciplíny, nazývanej *strojové učenie*[i]. Ako názov disciplíny napovedá, základnou ideou je vytvoriť matematický model štruktúry, ktorá je schopná klasifikovať spojité dáta na základe znalosti výsledkov iba v konečnom počte diskrétnych hodnôt[ii]. Využiteľnosť tatýchto štruktúr je celkom zrejmá – je pomocou nich možné predpokladať hodnoty funkcie medzi zmeranými vzorkami či klasifikovať rôzne signály, pre ktoré je inak náročné nájsť vhodnú parametrizáciu.

Neurónová sieť je jedným zo zástupcov takejto štruktúry. Jej elementárnou bunkou je *neurón*, ktorý je modelom biologického neurónu ako bunky *mozgu*. Štandardný neurón je vyobrazený na obrázku 3.1

[i] Bežné je používať skôr anglický názov *machine learning*.

[ii] Vo všeobecnosti môže takáto štruktúra byť schopná určiť výstup (*naučiť sa* rozoznávať vstup) i bez znalosti výstupov pre konkrétny, konečný počet vstupov. Tento spôsob učenia sa nazýva *učenie bez dohľadu* (*unsupervised learning*). V rámci tejto práce však budeme vždy pracovať s učením s dohľadom (*supervised learning*), pre ktoré je tento odstavec pravdivý.



Obr. 3.1: Schéma neurónu.

Prvá časť neurónu je v podstate pamäťou – je v nej uchovaný vektor *váh* (*weights*) $\vec{W}$ a *posunutie*[iii] (*bias*) b. Vektor $\vec{W}$ má počet prvkov zhodný s očakávaným počtom prvkov vstupného vektoru $\vec{x}$.

Pri prechode prvou časťou neurónu je transponovaný[iv] vektor $\vec{x}$ skalárne násobený vektorom $\vec{W}$ a následne je pripočítaná hodnota posunutia b. Pre jednorozmerný prípad ($\vec{x} = x_1 = x$) je teda $\tilde{y}$ obrazom bodu x na priamke so smernicou $k = W$ a posunutím b. Pre n-rozmerný prípad potom pôjde o súčet obrazov i-tych súradníc vektoru x na priamkach so smernicami $k_i = W_i$, ku ktorému je pripočítaná hodnota b.

Druhou časťou neurónu je časť výstupná, ktorá priraďuje vstupnej hodnote $\tilde{y}$ obraz v zobrazení f. Zobrazenie f sa nazýva *prechodová funkcia* neurónu a slúži na vyhodnotenie výsledku prvej časti. Rozoberme tri základné prechodové funkcie.

Identita. Identická prechodová funkcia odpovedá neurónu z obrázku 3.1, ktorý obsahuje iba prvú časť. hodnoty $\tilde{y}$ a y splynú. Výstup siete, pozostávajúcej z akokoľvek poprepájaných neurónov s identickou prechodovou funkciou je potom vždy iba *lineárnou kombináciou* prvkov vstupných vektorov a sieť možno zjednodušovať prepočtom váh a posunutí a redukovať tak počet vrstiev. Takúto prechodovú funkciu nebudeme priamo používať, no budeme používať funkciu podobnú, ktorá bude vstupnú hodnotu iba *normovať* do istého rozmedzia.

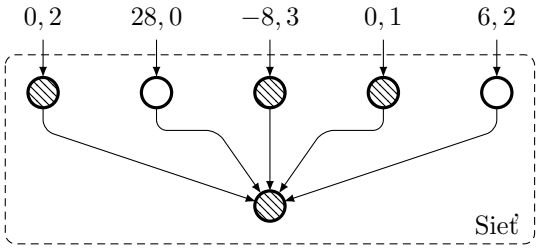
Jednotkový skok. Funkcia

$$1(t) = \begin{cases} 1 & \text{pre } t \geq 0 \\ 0 & \text{pre } t < 0 \end{cases} \quad (3.1)$$

poskytuje neurónu možnosť iba dvojstavového výstupu. Vhodná môže byť typicky vtedy, keď za vrstvou neurónov s takouto prechodovou funkciou očakávame binárne kódovaný výstup. Príkladom na použitie iba takejto prechodovej funkcie môže byť nasledujúca sieť:

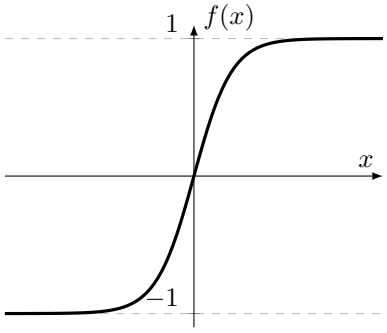
[iii] I keď tento termín nie je bežne zaužívaný a používa sa skôr termín *bias* alebo neformálne *béčko*, považujeme za vhodné použiť tento slovenský ekvivalent. Dobrým dôvodom je analógia s posunutím v analytickom tvare rovnice priamky v rovine, ktorá vyplynie z ďalšieho textu.

[iv] Vektorom o n prvkoch v tejto práci označujeme maticu o n riadkoch a jednom stĺpci.



Obr. 3.2: Príklad siete s neurónmi s prechodovou funkciou $f(t) = 1(t)$. Každý neurón predstavuje kruh, ktorý nadobúda dva stavy – *vyšráfovaný* (1) a *nevyšráfovaný* (0) podľa hodnoty jeho výstupu. Neuróny hornej vrstvy chceme naučiť vyšráfovať sa v prípade, že na ich vstupe je číslo menšie než 1. Neurón v spodnej vrstve by sme zas chceli naučiť vyšráfovať sa v prípade, že aspoň tri neuróny v hornej vrstve sú vyšráfované.

Nelineárna spojitá prechodová funkcia s oborom hodnôt $(-1; 1)$[v]. Pri použití takejto prechodovej funkcie nadobúdajú význam viacvrstvé siete, pri ktorých môže byť počet neurónov redukovaný vhodne zvolenou topológiou. Jednou z najbežnejších nelineárnych prechodových funkcií je *hyperbolický tangens*[vi], zobrazený na obrázku 3.3.



Obr. 3.3: Priebeh funkcie $f(x) = \tanh(x)$.

Vyhodnocovacie prechodové funkcie, ktoré sú typickou súčasťou neurónov poslednej vrstvy siete a majú za úlohu priradiť na výstup hodnotu z intervalu $\langle 0; 1\rangle$, ktorá by mala rozhodovať o tom, *ako veľmi má mať daný neurón „jednotkový“ výstup*. Ide o akúsi analógiu spojitej logiky – namiesto priameho binárneho výstupu z neurónu dostaneme iba číslo, na základe ktorého dokážeme spoznať i to, ako moc je naša sieť o svojom výstupe „presvedčená“. Pokiaľ máme napríklad dva výstupné neuróny s takouto prechodovou funkciou a na výstupe dostaneme z jedného z nich hodnotu $0,51$ a z druhého $0,49$, môžeme ľahko povedať, že na takýto

[v]Predpokladáme vstup do neurónu, ktorého prvky nadobúdajú taktiež hodnoty z intervalu $(-1; 1)$. Pokiaľ vstup nadobúda hodnoty mimo tento interval, rieši sa to jeho normovaním na tento interval – príkladom môže byť funkcia `mapminmax()` v časti 4.3.

[vi]V angličtine *tangent sigmoid.*

výsledok je v prípade rozhodovacej úrovne uprostred nespoľahlivý. Ak ale necháme neuróny bez takejto prechodovej funkcie, dostaneme výstupy 1 a 0, o ktorých toho toľko povedať nevieme.

V našich sieťach, trénovaných pomocou *NNT*, sme ako vyhodnocovaciu funkciu používali MATLAB funkciu `softmax`, definovanú ako

$$\texttt{softmax}(\vec{v}) = \left(\frac{e^{v_1}}{\sum_{i=1}^{\|\vec{v}\|} e^i}, \frac{e^{v_2}}{\sum_{i=1}^{\|\vec{v}\|} e^i}, \ldots, \frac{e^{v_{\|\vec{v}\|}}}{\sum_{i=1}^{\|\vec{v}\|} e^i} \right). \tag{3.2}$$

Neurónovú sieť potom tvoria neuróny, usporiadané do vrstiev, ktoré sú vzájomne istým spôsobom poprepájané – tak, ako pri sieťach iných druhov (napríklad telekomunikačných) sa toto usporiadanie nazýva *topológiou*. Učenie siete využíva znalosť hodnôt výstupu pre konkrétne vstupy a spätnú väzbu. Bez ujmy na všeobecnosti predpokladajme sieť s jediným výstupným neurónom. Zjednodušene je možné priebeh jej učenia vyjadriť takto:

1. Všetkým neurónom sú priradené počiatočné vektory $\vec{W}$ a hodnoty b[vii].
2. V cykle sú počítané hodnoty výstupu siete pre vstupy, ktorých výstupné hodnoty poznáme a pomocou ich hodnôt sú upravované váhy a posunutia všetkých neurónov v sieti.
3. Učenie je ukončené, pokiaľ po definovanom počte za sebou idúcich iterácií budú mať výsledky siete zhoršujúci sa trend, pokiaľ sa vyčerpajú trénovacie dáta alebo jednoducho po istom definovanom čase.

Presnejší popis algoritmu *SGDM*[viii], ktorý bude použitý i pri učení našich jednoduchých sietí, je uvedený v dodatku A.

Neural network toolbox

Mysliac na to, že neurónová sieť je v tejto práci našim *nástrojom* a nie *cieľom*, na vytvorenie siete a jej tréning sme použili čo najmohutnejšie zjednodušenia. I keď matematický model neurónu a synapsií[ix] nie je zložitý, balíček *Neural network toolbox* v MATLABe priniesol možnosť testovať výsledky sietí pre náš účel bez toho, aby sme sa museli zdržiavať implementáciou jednotlivých častí siete od základu. Pre všetky vyskúšané siete bol tento balíček použitý a v ďalšom texte tento fakt nebudeme pripomínať.

3.2 Algoritmy pre komplexné vyhodnotenie

3.2.1 Maximá spektrogramu

Tak trochu paradoxne, najjednoduchším z algoritmov pre určenie rýchlosti ako absolútnej hodnoty a ako smeru je jeden taký, ktorý tieto dve základné zložky určuje *spoločne*. Jedná sa o hľadanie *maxím spektrogramu*.

[vii] Prechodová funkcia je pre každý z neurónov fixná a počas učenia sa nemení.

[viii] Ide o skratku z anglického *stochastic gradient descent with momentum*. Hoci nie je problémom tento výraz preložiť, budeme sa pridŕžať označenia prostého označenia skratkou *SGDM*.

[ix] Prepojenia medzi neurónmi.

Spektrogram je maticou, ktorá obsahuje v každom svojom stĺpci *spektrum*[x] niekoľkých po sebe nasledujúcich vzoriek signálu. Parametre, ovplyvňujúce kvalitu spektrogramu, sú predovšetkým:

Šírka výrezového okna, udávajúca, z koľkých po sebe nasledujúcich vzoriek je počítaný Fourierov obraz pre každý zo stĺpcov spektrogramu. Čím väčšia je šírka okna, tým presnejšiu frekvenčnú informáciu o signále v okolí vzorky v strede výrezového okna dostaneme. Horší sa tým však informácia o časovej polohe takto vypočítaného spektra.

Záporný prekryv, ktorého hodnota označuje, koľko vzoriek od konca okna pre i-ty stĺpec spektrogramu sa podieľa na i plus prvom stĺpci. Nulový záporný prekryv znamená, že v signále neexistuje vzorka, ktorá by sa pričinila na hodnote viac než jedného stĺpca spektrogramu. Zväčšovaním záporného prekryvu dokážeme zlepšovať časové rozlíšenie spektrogramu pri použití dlhého okna. Narastá, samozrejme, ale i výpočtová náročnosť.

Okienková funkcia, ktorá *tvaruje* výsledné spektrá (stĺpce) a stará sa o potlačenie tzv. *presiakavania spektra*, teda javu, kedy sa vytvorením nekonečne veľkej zmeny v nekonečne malom čase pri „vyseknutí“ niekoľkých vzoriek v spektre objavia parazitné vyššie frekvencie. Okienkové funkcie[xi] majú preto na svojich krajoch hodnoty blízke nule a sú *symetrické*. Okienkovou funkciou sa bod po bode násobia vzorky z výrezového okna predtým, než vstúpia do Fourierovej transformácie[xii].

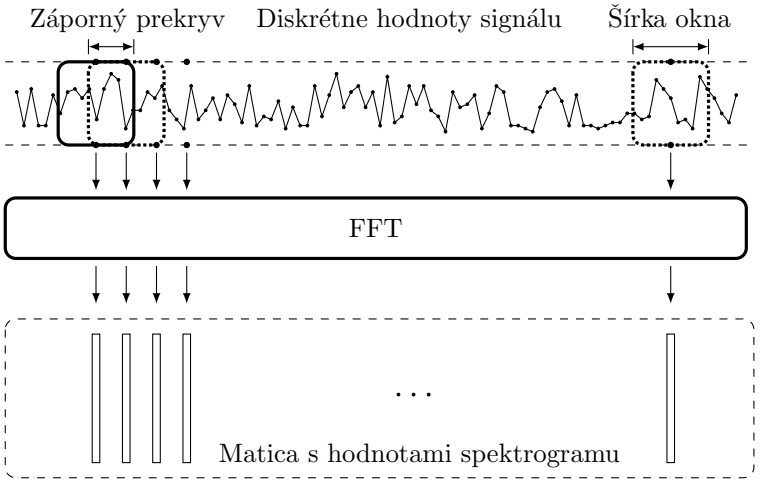
Hladina nasýtenia, ktorou možno v prvom náhľade odfiltrovať nezaujímavé výsledky – všetky hodnoty spektra vo všetkých stĺpcoch, nižšie než táto hladina, položíme rovné jej.

Algoritmus, určujúci rýchlosť na základe hľadania maxím spektrogramu, vychádza z faktu, že limitným prípadom pre nekonečne veľkú vzorkovaciu frekvenciu, nekonečne úzke výrezové okno a dokonalé podmienky je spektrogram, majúci v každom z nekonečného počtu svojich stĺpcov jedinú nenulovú hodnotu a to na pozícii, odpovedajúcej hľadanej dopplerovskej frekvencii, ktorá priamo odpovedá rýchlosti. Pre reálny prípad, nižšiu vzorkovaciu frekvenciu a širšie okno je teda opodstatnené očakávať, že i napriek tomu, že nenulové hodnoty sa budú nachádzať po celej dĺžke stĺpcov spektrogramu, hľadanej dopplerovskej frekvencii bude pre každý stĺpec odpovedať aspoň pozícia *maxima* spomedzi jeho hodnôt. Tento algoritmus je formálne zhrnutý v algoritme 1.

[x] Môže ísť o výkonové spektrum, spektrálnu výkonovú hustotu, spektrálnu výkonovú hustotu v decibeloch, ...

[xi] Hoci hovoríme o *funkciách*, máme v tomto prípade na mysli postupnosť o dĺžke rovnej dĺžke výrezového okna, ktorú tvoria obrazy hodnôt 1 až *dĺžka výrezového okna* v zobrazení, ktoré možno nazvať už skutočne *okienkovou funkciou*.

[xii] Pre lepšiu analógiu s programovou implementáciou sme zvolili postup, kedy najskôr použijeme *obdĺžnikové* okno na „vyseknutie“ potrebnej časti radu vzoriek (n_w bodov) a následne tento úsek násobíme *Hammingovym* oknom o plnej šírke tohoto úseku, teda taktiež n_w bodov.



Obr. 3.4: K vysvetleniu pojmu *spektrogram.*

Algoritmus 1 Hľadanie maxím spektrogramu

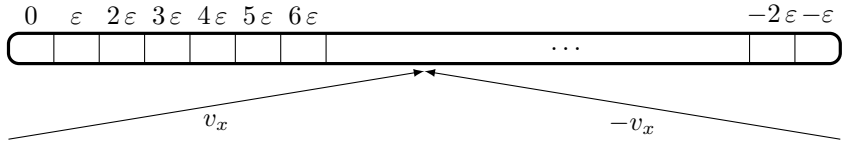
1: **procedure** H.M.S($\vec{s}$, f_s, n_w, o_n, p_{sat}, okienková funkcia) ▷ Vektor $\vec{s}$ je vektor komplexných vzoriek, získaných vzorkovaním radarových signálov vzorkovacou frekvenciou f_s. n_w predstavuje šírku okna, o_n záporný prekryv, p_{sat} hladinu nasýtenia pre uvažovaný druh spektra.
2: Načítaj n_w vzoriek z $\vec{s}$ do poľa `buffer`.
3: **for** $i \leftarrow$1 až n_w **do**
4: $\texttt{buffer}_i \leftarrow \texttt{buffer}_i \cdot w(i)$ ▷ $w(i)$ je hodnota okienkovej funkcie v bode i
5: **end for**
6: FFT(`buffer`) ▷ Výpočet FFT. Predpokladajme, že funkcia FFT pracuje *in-place* a výsledok bude teda opäť uložený v poli `buffer`.
7: **for** $i \leftarrow$1 až n_w **do**
8: **if** $\texttt{buffer}_i < p_{\text{sat}}$ **then**
9: $\texttt{buffer}_i \leftarrow p_{\text{sat}}$
10: **end if**
11: **end for**
12: $m \leftarrow FindMaxInd(\texttt{buffer})$ ▷ Funkcia $FindMaxInd$ hľadá index maxima zadaného poľa
13: **return** $VX(m)$ ▷ Funkcia VX mapuje hodnotu indexu maxima na hodnotu rýchlosti v_x.
14: **end procedure**

Pred implementáciou algoritmu 1 je ešte vhodné spektrogram preusporiadať tak, aby bol lepšie čitateľný. Pôvodné spektrum, ktoré dostaneme Fourierovou transformáciou okienkovou funkciou preškálovaného výrezu, je nesymetrické – ako vyplýva časti 1.3, hodnoty kladnej dopplerovskej frekvencie narastajú od prvej hodnoty (0) s krokom ε až do polovice dĺžky spektra. Od poledného indexu smerom k tým nižším zas narastajú absolútne hodnoty zápornej frekvencie a to od hodnoty ε s krokom ε. Krok ε je najmenšou rozlíšiteľnou zmenou

dopplerovskej frekvencie a je rovný

$$\varepsilon = \frac{f_s}{n_w}\,, \tag{3.3}$$

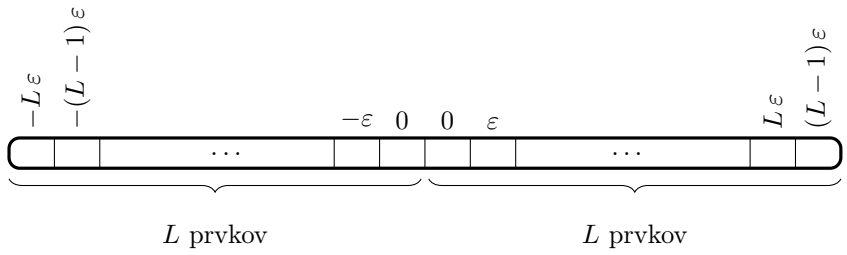
kde f_s je vzorkovacia frekvencia. Takto indexované spektrum je znázornené na obrázku 3.5.



Obr. 3.5: Dopplerovské frekvencie, odpovedajúce bodom spektra, teda jedného stĺpca spektrogramu.

My by sme však chceli usporiadať spektrá tak, aby f_d plynulo nadväzovala od jej najzápornejších hodnôt cez hodnotu nulovú až k hodnotám najväčším kladným. Za *spektrogram* teda ďalej nebudeme hneď považovať priamo rad výsledkov FFT, aplikovanej na škálované výrezy – toto pomenovanie tento rad dostane až po preskupení a odstránení nadbytočných hodnôt.

Zvolené preskupenie indexov stĺpcov je na obrázku 3.6.



Obr. 3.6: Hodnoty dopplerovských frekvencií, zodpovedajúce bodom novozavedeného spektrogramu. Voľba parametra L a jeho význam sú ozrejmené v časti 3.2.1.1

3.2.1.1 Voľba užitočných indexov

Na základe výsledkov práce [1] a nášho overenia sme hneď od začiatku zvolili parametre spektrogramu, ktoré odpovedali kompromisu medzi frekvenčným a časovým rozlíšením. Zmenou oproti [1] bolo použitie *Hammingovho* okna namiesto *Hannovho*[xiii], pretože sme s ním dosahovali menšie hladiny PSD v oblasti najnižších frekvencií. Parametre sú zhrnuté v tabuľke 3.1.

Šírka výrezového okna	8192
Záporný prekryv	7169
Okienková funkcia	Hammingova

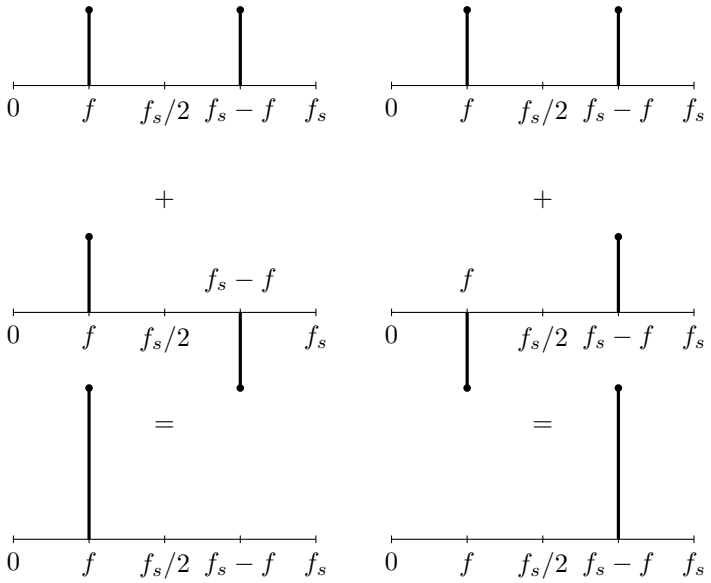
Tab. 3.1: Parametre spektrogramu.

[xiii] Možno sa stretnúť i s označením *Hanning window*.

Pri tejto šírke okna a našej vzorkovacej frekvencii dostávame spektrum vo frekvenčných bodoch, vzdialených od seba na základe rovnice (3.3)

$$\varepsilon = \frac{44100}{8192} \approx 5,3833 \text{ Hz}\,. \tag{3.4}$$

Ich rozmedzie prirodzene siaha od 0 Hz do $f_s - \varepsilon$ Hz. Používame komplexný signál, ktorého reálnu časť tvorí kosínusový signál a imaginárnu časť signál sínusový o rovnakej frekvencii (označme ju f)[xiv], pričom znamienko imaginárnej časti môže byť v závislosti od smeru pohybu kladné alebo záporné. V závislosti od smeru pohybu sa teda mení i poloha nenulovej hodnoty výsledného spektra – pre kladný smer pohybu je umiestnená na frekvencii f, pre smer záporný zas na frekvencii $f_s - f$. Ilustrované je to na obrázku 3.7, kde v prvom riadku je spektrum reálnej časti, v druhom riadku spektrum imaginárnej časti s kladným a záporným znamienkom a v riadku poslednom spektrum ich súčtu, čo vyplýva z linearity Fourierovej transformácie.



Obr. 3.7: K vysvetleniu závislosti tvaru spektra komplexných vzoriek od smeru pohybu.

Stanovme si rozmedzie absolútnych hodnôt rýchlosti, ktoré chceme vedieť zachytiť. V rámci tejto práce sme zvolili maximálnu detekovateľnú rýchlosť $v_x^M = 63,2$ km/h, čo korešponduje so zameraním práce na rozpoznávanie *nízkych* rýchlostí.

Frekvenčný krok ε, prepočítaný na krok rýchlosti ϵ podľa vzťahu (1.3) je

$$\epsilon = \frac{\varepsilon\,\lambda_T}{2\,\cos(\alpha)} = \frac{5,3833 \cdot \frac{3}{241,25}}{2 \cdot \frac{1}{\sqrt{2}}} = 0,0473 \text{ m/s}\,. \tag{3.5}$$

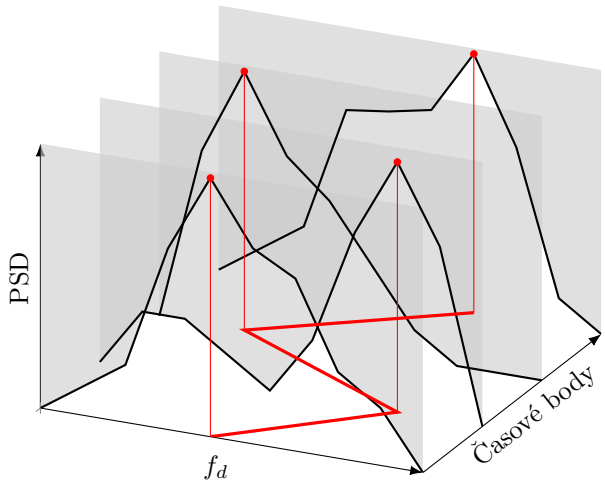
[xiv] Isteže to nie je splnené presne; V ideálnom prípade však sú a v tejto vete sa snažíme o popis princípu, nie reálneho prípadu.

Potom maximálny index pri indexovaní od 1 z každej „strany“ spektra, ktorý chceme ešte vyhodnocovať, je

$$\mathcal{I} = \left\lceil \frac{v_x^M}{\epsilon} \right\rceil = \left\lceil \frac{\frac{63,2}{3,6}}{0,0473} \right\rceil = 372\,. \tag{3.6}$$

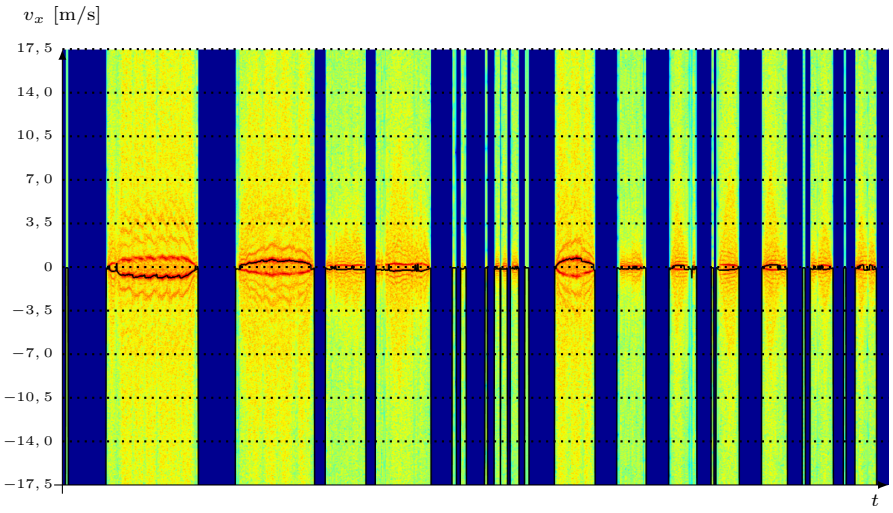
Na obrázku 3.6 je parameter L práve maximálnym uvažovaným indexom. V tejto práci preto budeme používať spektrogram, pozostávajúci zo stĺpcov, znázornených na obrázku 3.6, kde $L = \mathcal{I} = 372$.

Vyhodnotenie rýchlosti potom prebehne tak, ako je ilustrované na obrázku 3.8. V prostriedku stĺpcov spektrogramu sa bude vďaka vhodnému preusporiadaniu nachádzať nulová rýchlosť, v hornej polovici bude s krokom ϵ narastať rýchlosť vpred a rýchlosť vzad bude narastať s týmto istým krokom smerom od stredu spektrogramu nadol.

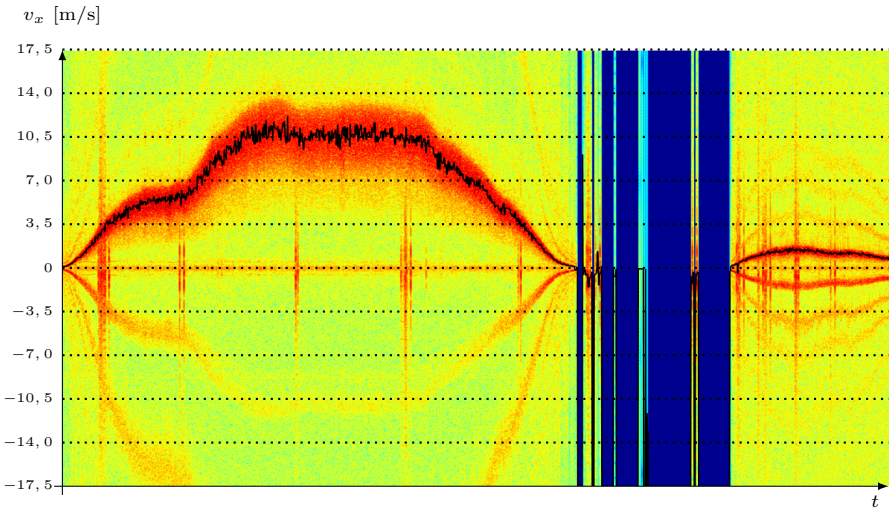


Obr. 3.8: Spôsob vyhodnotenia rýchlosti vrátane jej smeru z maxím spektrogramu. Jednotlivé spektrá sú znázornené čiernymi priebehmi, priebeh rýchlosti je čiarou červenou. Medzi časovými bodmi, v ktorých je počítaný spektrogram, je jej priebeh lineárne interpolovaný.

Implementácia takéhoto prístupu sa nachádza v kóde E.2 a výsledok jej použitia na obrázkoch 3.18 a 3.10.



Obr. 3.9: Vyhodnotenie rýchlosti na nahrávke $\mathcal{N}_1$ pomocou hľadania maxím spektrogramu. Čiernou čiarou je znázornený vyhodnotený priebeh. Nahrávka bola vytvorená pri použití usporiadania z obrázku 2.12.



Obr. 3.10: Vyhodnotenie rýchlosti na nahrávke $\mathcal{N}_2$ pomocou hľadania maxím spektrogramu. Nahrávka bola vytvorená pri použití usporiadania z obrázku 2.14.

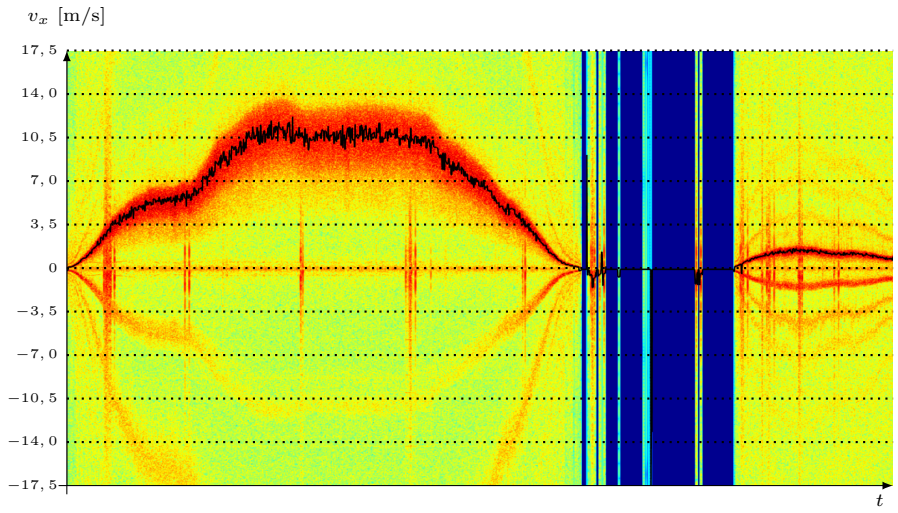
Na prvý pohľad možno o týchto výsledkoch prehlásiť nasledujúce tvrdenia:

1. Pre stredné rýchlosti, teda od približne 1 do 7 m/s funguje toto vyhodnotenie spoľahlivo. Pri vyšších rýchlostiach je frekvenčné rozlíšenie relatívne malé a výsledok výrazne osciluje. Toto je možné odstrániť vhodne zvoleným filtrom.

2. Pre rýchlosti nižšie než 1 m/s rozlíšenie smeru pohybu nie je dostatočné. Prítomnosť vyšších harmonických napohľad maximálnej dopplerovskej frekvencie však ponúka istú obmenu v spôsobe zisťovania smeru pohybu.

3. Pri prudkých zmenách rýchlosti, viditeľných na obrázku 3.10, ktoré vznikli pravdepodobne nábehmi vozidla na terénne nerovnosti pri vysokej rýchlosti, pracuje tento algoritmus relatívne rozumne.

4. Vhodnou obmenou je pri zistení maximálneho indexu na *okraji* spektrogramu položiť rýchlosť na zodpovedajúcom indexe rovnú *nulovej*. Obrázok 3.10 po tejto úprave je uvedený na obrázku 3.11.



Obr. 3.11: Vyhodnotenie rýchlosti na nahrávke $\mathcal{N}_2$ pomocou hľadania maxím spektrogramu po úprave, kedy sme hodnoty, ktoré boli vyhodnotené ako index 0, položili rovné indexu 372, teda nulovej rýchlosti a teda prostriedku spektrogramu vo frekvenčnom smere. Úprava skriptu E.2 sa nachádza v kóde E.3.

3.2.2 Vyhodnotenie neurónovou sieťou

Alternatívou k algoritmickému prístupu z predchádzajúcej časti je využiť neurónovú sieť. Základná myšlienka neurónových sietí je popísaná v časti 3.1. Pri pohľade na spektrogram, ako je napríklad ten z obrázku 3.18, je v mnohých prípadoch zrejmé, aká by mala byť správne určená hodnota rýchlosti i bez väčšej kvalifikácie či veľmi dobrého zraku. Nabáda sa preto vytvoriť a natrénovať sieť, ktorá by bola sama schopná hodnotu rýchlosti určiť.

3.2.2.1 Sieť s využitím obrázkov

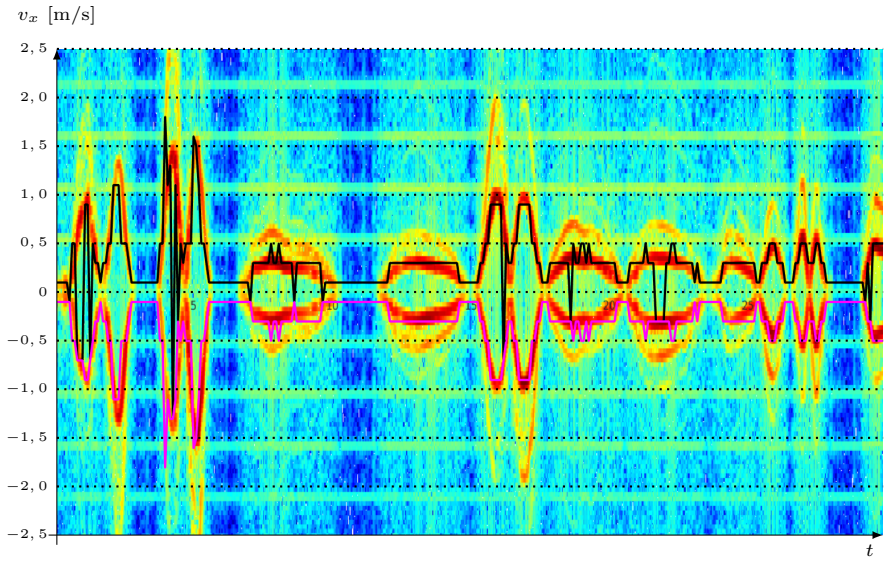
Prvou vyskúšanou sieťou bola taká, ktorá ako vstup brala časti spektrogramu o šírke väčšej než 1 stĺpec, ktoré sme kvôli metodike spracovania nazvali *obrázky* či *podspektrogramy*. Vstupy siete budú tvoriť definované časti „plochy“ takýchto obrázkov a výstupom bude rýchlosť, ktorá je na nich zobrazená.

Získavanie trénovacích dát Prvým predpokladom na to, aby bolo možné vytvoriť fungujúcu sieť, je dostupnosť trénovacích dát, teda množiny vstupov, $\mathbb{T}$, ktorá obsahuje dáta rovnakého typu, aký bude vstup do natrénovanej siete a množiny cieľov, $\odot$, ktorá obsahuje obrazy množiny $\mathbb{T}$ v zobrazení, ktoré by sme chceli, aby sieť simulovala.

Problémom pri získavaní trénovacích dát je, že množinu $\odot$ je častokrát (aspoň v prípadoch, kedy nepoznáme algoritmus priradenia vstupu siete k výstupu a má teda skutočne zmysel sieť využiť) nutné tvoriť ručne, subjektívnym hodnotením množiny $\mathbb{T}$. Alternatívou pre vytvorenie dvojice $\mathbb{T}$, $\odot$ je tiež prístup, kedy máme známu množinu $\odot$ a predstavu o tom, ako by mal pre tú-ktorú hodnotu z $\odot$ vyzerať vstup, ktorý na základe toho syntetizujeme. Pre siete, vyskúšané v tejto práci sme použili oba tieto prístupy.

Postup získavania trénovacích dát pre obrázkovú sieť sa nachádza v dodatku B.

Tréning na dátach z patchwork generátoru Jedným z prvých pokusov o generovanie trénovacích dát bol *patchwork generátor*, popísaný bližšie v časti B.3. Jeho použitie narážalo na niekoľko veľmi závažných problémov a to konkrétne



Obr. 3.12: Vyhodnotenie nahrávky $\mathcal{N}_a$ pomocou skriptu E.7. Ružovou čiarou je znázornené vyhodnotenie absolútnej hodnoty výsledku (so záporným znamienkom kvôli zabráneniu prekryvu čiar na jednej strane obrázku), čiernou vyhodnotenie bez ovplyvnenia absolútnou hodnotou.

1. Veľká zložitosť takéhoto riešenia oproti iným prístupom, dosahujúcim omnoho lepšie výsledky.

2. Práca s RGB hodnotami podspektrogramov – takáto sieť prijíma obrázky vo formáte bitovej mapy, čo prináša ďalšie výpočty naviac s radarovým signálom.

3. Celková nevhodnosť takejto siete pre určovanie v reálnom čase.

Najlepšie dosiahnuté výsledky sa nachádzajú na obrázku 3.12. Je potrebné tiež poznamenať, že ako spôsob znázornenia spektrogramu, tak ani rozsah pozorovaných rýchlostí sa nezhodujú s informáciami, deklarovanými v predchádzajúcich častiach. Je to spôsobené tým, že tento generátor bol vytvorený pred ustálením našich konvencií. Vzhľadom na jeho zlé výsledky a hlavne ťažkopádnosť aktuálneho prevedenia sme v jeho úpravách ani zlepšovaní nepokračovali; Považujeme však za vhodné sa o ňom zmieniť, nakoľko po dôkladnej rekonštrukcii a zjemnení kroku môže byť v budúcnosti táto idea užitočná.

Tréning na dátach z priemerovacieho generátoru Druhý z prístupov ku generovaniu trénovacích dát kombinuje oba prístupy ich zisku, ktoré boli spomínané v časti 3.2.2.1. Jeho myšlienkou je z malého množstva ručne ohodnotených podspektrogramov získať množstvo dostatočné na dobrý tréning siete tak, že:

1. Ručne – napríklad spôsobom z časti B.1 – ohodnotíme rozumné množstvo dát.

2. Jednotlivé podspektrogramy hodnotených dát poukladáme do zložiek, z ktorých každá bude obsahovať práve jednu hladinu rýchlosti na základe subjektívneho vyhodnotenia. Je zrejmé, že tieto zložky nebudú obsahovať rovnaký počet podspektrogramov; Zabezpečme však, aby v každej z nich, teda pre každú hladinu rýchlosti, ktorú chceme rozoznávať, boli prítomné podspektrogramy *aspoň dva*.

3. V každej zo zložiek v cykle robme tieto úkony:

   i. Kontrola počtu podspektrogramov v zložke – pokiaľ je väčšia, než žiadaný počet, cyklus ukončíme.

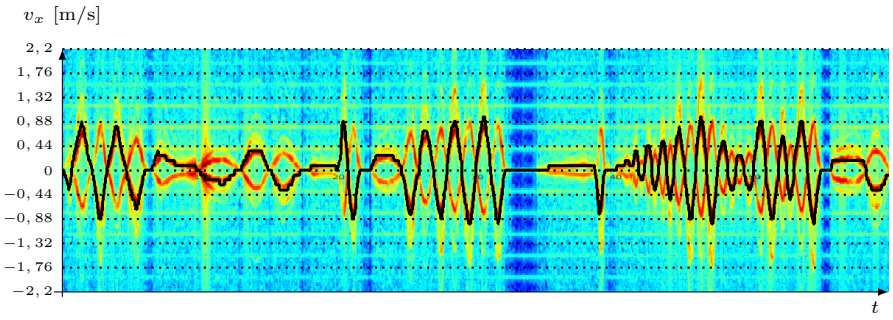
   ii. Výber dvoch náhodných podspektrogramov z tejto zložky.

   iii. Uloženie aritmetického RGB priemeru týchto dvoch podspektorgramov ako *nový* podspektrogram.

I v tomto prípade bol generátor navrhovaný pred ustálením finálnych konvencií. Vzhľadom na to, že sme sa rozhodli ho neskôr pri návrhu softvéru pre samostatné radarové zapojenie nepoužiť, nemenili sme rozsah rýchlostí na trénovacích (a teda ani testovacích) obrázkoch. Veríme, že rozsah, použitý v tejto časti nebude pre čitateľa rušivý.

Ohodnotenie prebehlo na spektrogramoch nahrávok $\mathcal{N}_b$, $\mathcal{N}_c$ a polovice spektrogramu nahrávky $\mathcal{N}_d$, nahraných pri neštandardnom usporiadaní, kedy bola k radaru približne pod uhlom $\alpha = 45°$ približovaná a vzďaľovaná ľudská dlaň. Po vykreslení spektrogramov sme ich MATLAB prevedenie uložili, upravili sme rozmery (výška bola zvolená 756 px) a tak, ako v dodatku B.1 rozrezali pomocou IMAGEMAGICK na 2 px široké „pásy“ a ku každému z nich pridali stupnicu.

Príklad ručného ohodnotenia podspektrogramov spektrogramu nahrávky $\mathcal{N}_b$ je uvedený na obrázku 3.13.



Obr. 3.13: Ručne vyhodnotený spektrogram nahrávky $\mathcal{N}_b$.

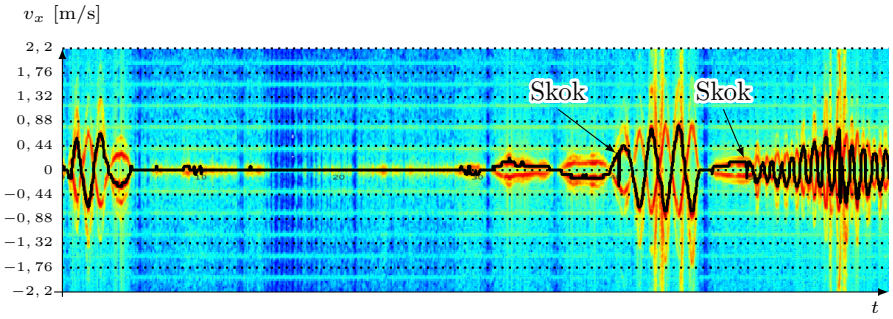
Rozsah vykreslenia spektrogramov pre tento generátor pokrýval rozmedzie $-1,5$ m/s až $1,5$ m/s s krokom $0,1$ m/s. Implementácia generátoru je priblížená v dodatku B.4 a postup prípravy trénovacích množín s jeho použitím by sa dal zhrnúť takto:

1. Ručné ohodnotenie trénovacích dát a vytvorenie textového súboru `classification_subjective.txt`, obsahujúceho v k-tom riadku hodnotu rýchlosti na k-tom podspektrograme.

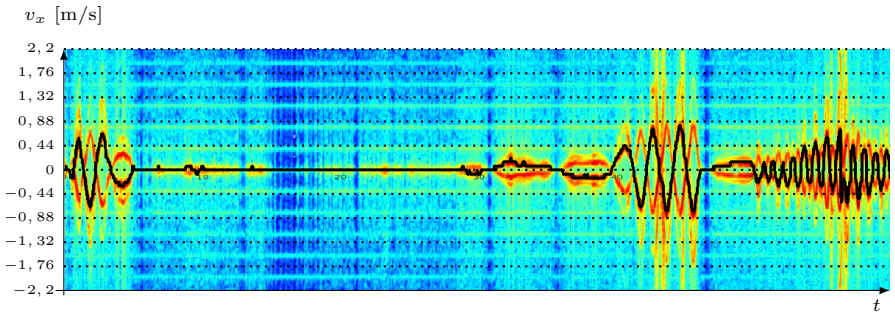
2. Rozdriedenie ohodnotených podspektrogramov do zložiek na základe `classification_subjective.txt` a ich premenovanie tak, aby každá z nich sa nazývala desaťnásobkom hodnoty rýchlosti, ktorá je zobrazená na podspektrogramoch v nej.

3. Spustenie skriptu `clone_global.sh` (viď dodatok B.4).

S použitím obrázkov z tohoto generátoru bola trénovaná sieť, špecifikovaná skriptom E.8. Každá z trénovacích zložiek obsahovala 400 podspektrogramov veľkosti $756 \times 2 \times 3$ bodov, pričom posledná dimenzia je tvorená hodnotami RGB koeficientov. Testovacie dáta tvorila nahrávka $\mathcal{N}_e$. Výsledky sú uvedené na obrázku 3.14.



Obr. 3.14: Vyhodnotenie nahrávky $\mathcal{N}_e$ sieťou z kódu E.8. Skoky sa podarilo účinne potlačiť kĺzavým priemerovaním o malej dĺžke okna.

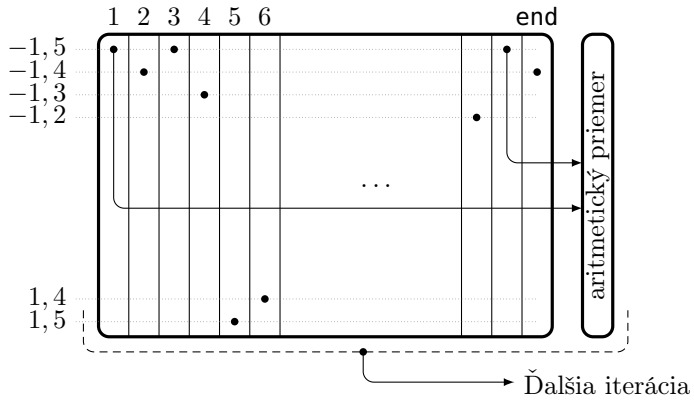


Obr. 3.15: Ukážka možnosti jednoduchého potlačenia preskokov v smere pohybu pomocou kĺzavého priemeru o dĺžke okna 5 bodov. Obrázok je výsledkom kódu E.9.

Takto volené tréningové dáta dosahujú výrazne lepšie výsledky, než tie z časti 3.2.2.1. Dalo by sa povedať, že výsledky sú uspokojivé pri dostatočne veľkom kroku určovaných rýchlostí. Pri kroku menšom sieť začne byť nespoľahlivou – priemerovacím generátorom nedokážeme uspokojivo plniť zložky, odpovedajúce rýchlostiam s krokom menším než približne $0,07$ m/s. Vzhľadom na krok o veľkosti rádu 10^{-1} m/s sme sa rozhodli tento variant vo finálnom programe nevyužiť – cieľom je predsa rozoznávať rýchlosti s presnosťou väčšou.

3.2.2.2 Bezobrázková sieť

Z praktického hľadiska je využitie obrázkov na vyhodnotenie rýchlosti nevhodné. Hoci pre predstavu a návrh generátoru trénovacích dát boli obrázky nenahraditeľným pomocníkom, uvedomme si, že RGB koeficienty ich bodov sú len obrazmi hodnôt spektrálnej výkonovej hustoty v istom jednojednoznačnom zobrazení. Obrázky teda možno bez straty informácie nahradiť vektormi či maticami číselných hodnôt a na tréning i vyhodnocovanie použiť tieto.



Obr. 3.16: Ku generovaniu trénovacích dát pre bezobrázkovú sieť.

Generovanie tréningových množín prevzalo a upravilo algoritmus 6. Veľkosť vstupov a teda celková topológia siete bola taktiež upravená. Ako sme totiž

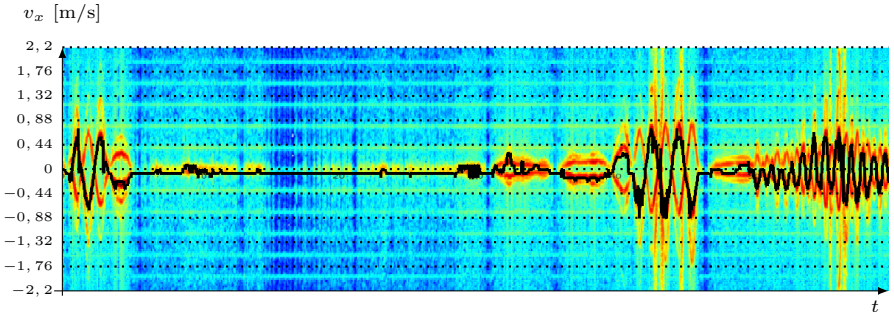
spomínali v závere časti 3.2.2.1, ani najúspešnejšia z vyskúšaných topológií pre 2 a viac časových bodov široké vstupy sa nejavila ako vhodná na použitie v našom programe. Keďže prechod z obrázkovej siete na sieť bez použitia obrázkov by výsledky pri správnej implementácii zmeniť nemal, okrem prerobenia siete na bezobrázkovú zmeníme i jej topológiu.

Nová sieť bude pracovať so samostatným podspektrogramom pre každý z časových bodov. Základ generátoru bude tvoriť, rovnako ako v prípade obrázkového priemerovacieho generátoru, set subjektívne vyhodnotených podspektrogramov z nahrávok $\mathcal{N}_b, \mathcal{N}_c$ a polovice nahrávky $\mathcal{N}_d$. Časť postupu načrtneme na obrázku 3.16.

Na začiatku je matica $\mathbb{T}$, ktorej stĺpce sú tvorené hodnotami spektrogramov trénovacích nahrávok. Na základe ich ručného vyhodnotenia vytvoríme štruktúru s, v ktorej bude každá bunka odpovedať jednej hodnote rýchlosti medzi $-1,5$ a $1,5$ m/s s krokom $0,1$ m/s a bude obsahovať indexy stĺpcov, v ktorých bola daná rýchlosť rozpoznaná. Vzhľadom na použitie MATLABu na implementáciu, indexujeme od jednotky. Iterujeme dva indexy:

- Prvý index prebieha všetky bunky štruktúry s, obsahujúce viac než nula indexov.
- Druhý index zväčšujeme dovtedy, kým v danej bunke nie je požadovaný počet indexov.

Tak, ako pri pôvodnom prevedení priemerovacieho generátoru pre obrázky, zoberieme v každej iterácii dve náhodné čísla v rozmedzí 1 až počet indexov v aktuálnej bunke. Stĺpce matice $\mathbb{T}$ na indexoch, určených týmito číslami spriemerujeme a tento priemer pripojíme k nej ako jej nový posledný stĺpec. Následne vygenerujeme náhodnú permutáciu čísel 1 až *nový počet stĺpcov* $\mathbb{T}$ a na jej základe zameníme poradie jej stĺpcov. Rovnakým spôsobom zameníme i poradie prvkov vektoru cieľových hodnôt, na ktorého koniec bola predtým pripojená hodnota rýchlosti z aktuálnej bunky štruktúry s. Po každej iterácii musíme tiež obnoviť hodnoty v bunkách štruktúry s, pretože indexy stĺpcov sa po ich permutácii zmenia. Po tom, ako aktuálna bunka nadobudne minimálny požadovaný počet indexov, presunieme sa na bunku nasledujúcu. Kompletný skript vrátane tréningu siete je uvedený v kóde E.11. Výsledok vyhodnotenia nahrávky $\mathcal{N}_e$ je uvedený na obrázku 3.17.



Obr. 3.17: Výsledky siete z časti 3.2.2.2 po vyhodnotení nahrávky $\mathcal{N}_e$.

Pretože oproti riešeniu z časti 3.2.2.1 bol zmenený ako prístup ku kódovaniu podspektrogramov, tak topológia siete, vo vyhodnotení považujeme za vhodné pristúpiť osve ku každému z týchto aspektov.

Bezobrázkový prístup znamenal podstatné zjednodušenie implementácie generátora trénovacích množín. Pretože ale koncový program nemá za úlohu sieť trénovať, ale využiť jednu z už natrénovaných, nie je táto výhoda prakticky využiteľná.

Zmena topológie siete tak, aby bol vstupom jediný vektor spektrálnych výkonových hustôt priniesla podľa očakávania mierne zhoršenie výsledkov. Počet neurónov sa však podstatne redukuje a tým sa značne znižuje i počet koeficientov, ktoré by náš program musel uchovávať, ak by sme takúto sieť využívali.

3.3 Algoritmy pre vyhodnotenie absolútnej hodnoty rýchlosti

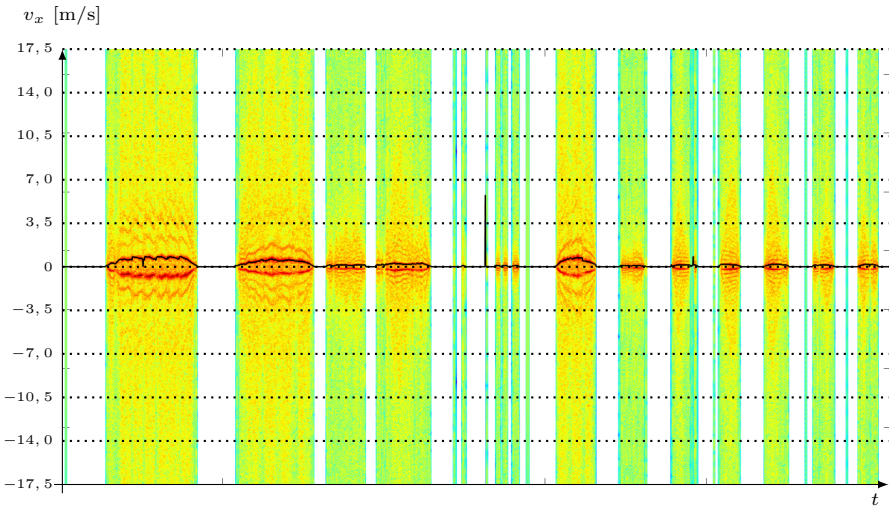
3.3.1 Maximá spektrogramu

Tak, ako v časti 3.2.1, hľadajme maximá spektrogramu. V tomto prípade ale neuvažujme celý spektrogram, ale iba jeho polovicu. Pozrieme sa na maximum *súčtu* „kladnej“ a „zápornej“ časti a maximum samotnej časti kladnej. Priamočiary algoritmus je uvedený ako algoritmus 2.

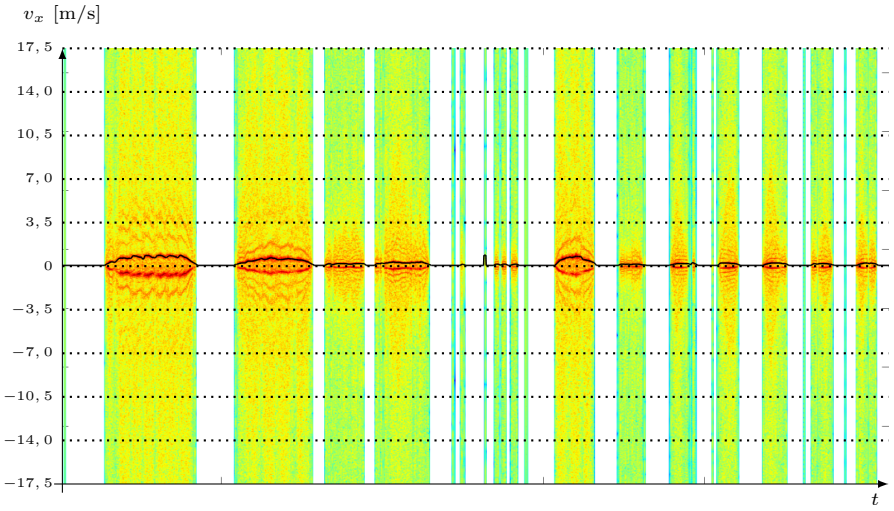
Algoritmus 2 Hľadanie absolútnej hodnoty rýchlosti z maxím spektrogramu

```
1: procedure AbsMax(𝕊)                          ▷ Matica 𝕊 je spektrogramom
2:     for i ← 1, 2, ..., počet stĺpcov 𝕊 do
3:         for j ← 1, 2, ..., 372 do
4:             𝕊_ji ← 𝕊_ji + 𝕊_(2·372−j)i
5:         end for
6:     end for
7:     for k ← 1, 2, ..., počet stĺpcov 𝕊 do
8:         M(k) = index maxima hornej polky k-teho stĺpca matice 𝕊 + 372
9:     end for
10:    return VX(M)  ▷ Vracia hodnoty rýchlostí, odpovedajúce indexom M
11: end procedure
```

Implementácia z veľkej časti kopírovala kód E.2 a jej samostatná časť je uvedená v kóde E.4. Výsledky z nahrávky $\mathcal{N}_1$ možno vidieť na obrázku 3.18. Vidíme na ňom, že presnosť takéhoto vyhodnotenia malých rýchlostí je skutočne dobrá. Po vyhladení kĺzavým priemerovaním sa podarilo jednoducho odstrániť nežiadané skoky a navyše ním boli pozitívne ovplyvnené i nábežné časti úsekov pohybu, ktoré sa vďaka nemu správali prirodzenejšie – ako bolo písané už v časti 2.2, pásmové priepusti, cez ktoré prechádzajú signálové zložky, spôsobujú výrazný útlm dopplerovských frekvencií pod 10 Hz. Výsledok, dosiahnutý s použitím kĺzavého priemeru, ktorý priemeroval pre každú vzorku hodnotu jej samotnej a 6 nasledujúcich hodnôt, je na obrázku 3.19.



Obr. 3.18: Vyhodnotenie absolútnej hodnoty rýchlosti na nahrávke $\mathcal{N}_1$ pomocou hľadania maxím súčtu kladnej a zápornej časti spektrogramu.



Obr. 3.19: Priebeh z obrázku 3.18, vyhladený pomocou kĺzavého priemeru. Dôležitá časť kódu je v kóde E.5.

Zistenou nevýhodou je nutnosť použitia rôznych dĺžok okna kĺzavého priemeru pre rôzne hodnoty rýchlostí. Pre vyššie rýchlosti totiž maximá spektrogramu v tesne po sebe nasledujúcich časových bodoch obsahujú často veľmi prudké zmeny, ktoré nemožno vyhladiť krátkym filtrom. Na druhej strane dlhé okno filtru neúnosne skreslí hodnoty malých rýchlostí, ktoré častokrát pokrývajú iba krátky časový úsek. Pri použití takéhoto vyhodnotenia by bolo nutné implementovať systém prispôsobenia dĺžky filtru približne určenej absolútnej hodnote rýchlosti, teda urobiť filter *lokálne adaptabilný*.

3.3.2 Prvý spektrálny moment

Jedným z veľkých nedostatkov hľadania absolútnej hodnoty rýchlosti pomocou maxím spektrogramu je veľký rozptyl výsledkov pre vyššie rýchlosti. Hlavným dôvodom na to je pravdepodobne to, že pri vyšších rýchlostiach (ktoré boli nahrávané jedine pri zapojení meracej aparatúry z obrázku 2.14) je nestála vertikálna zložka rýchlosti v. Rôzne nerovnosti a reakcia tlmenia automobilu na ne potom spôsobujú zmeny v dopplerovskej frekvencii.

Chceli by sme dosiahnuť, aby naše vyhodnotenie nebolo príliš citlivé na takéto fluktuácie. Iste – je možné výsledky filtrovať v čase. Tento prístup má ale prekážku, ktorá je popísaná na konci časti 3.3.1.

Riešením môže byť využitie *prvého spektrálneho momentu*[xv]. Definujme prvý spektrálny moment ako

$$C = \frac{\sum_{i=1}^{N} \mathcal{S}_i \cdot i}{\sum_{i=1}^{N} \mathcal{S}_i} \,. \tag{3.7}$$

Definícia (3.7) odpovedá hľadaniu *ťažiska* spektra – pokiaľ máme N-prvkové spektrum $\mathcal{S}$ a použijeme naň tento vzorec, dopočítame sa k *indexu* jeho ťažiska. Pokiaľ navyše upravíme výraz na pravej strane tak, že namiesto váhovania indexom i budeme váhovať hodnoty spektra nejakou inou veličinou, s vektorom indexov lineárne závislou a v menovateli zameníme znenie analogicky, môžeme na výstupe dostávať priamo hodnotu rýchlosti či dopplerovskej frekvencie v ťažisku spektra.

Vyhodnotenie teda prebehne tak, že po výpočte spektrogramu priradíme každému časovému bodu absolútnu hodnotu rýchlosti, odpovedajúcu prvému spektrálnemu momentu, váhovanému rýchlosťami v jednotlivých bodoch spektier[xvi].

Prvý spektrálny moment by mal oproti metóde maxím priniesť najmä spresnenie; Výsledky článku [12] hovoria, že výhodou môže byť i *zrýchlenie* celého procesu určovania absolútnej hodnoty rýchlosti. V tejto práci nebude takýto prístup implementovaný, v budúcnosti však môže byť vhodným zlepšením.

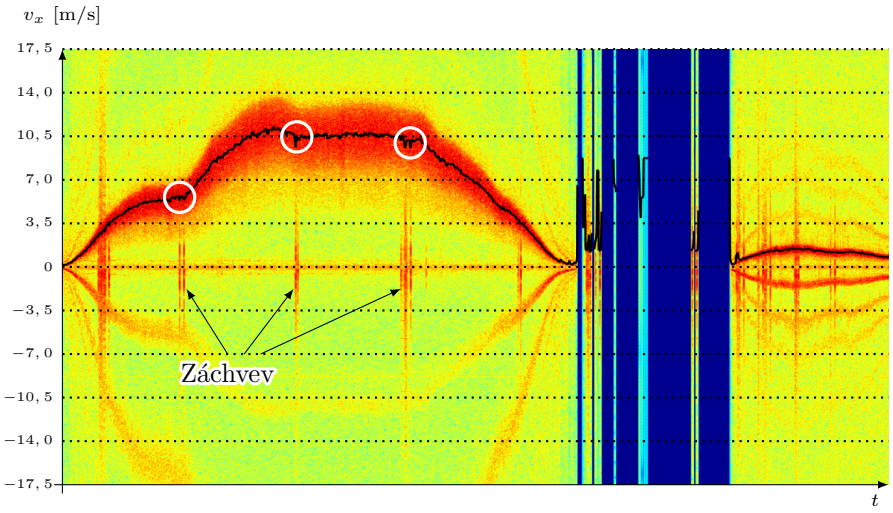
3.3.2.1 Priama aplikácia

Pred akýmikoľvek inováciami, pozrime sa na výsledky vyhodnotenia našich nahrávok priamo pomocou vzťahu (3.7). Skript na vytvorenie nasledujúcich obrázkov, ktorý obsahuje jeho implementáciu, možno vidieť v kóde E.10.

Na výsledkoch z nahrávky $\mathcal{N}_2$ (obrázok 3.20) je po porovnaní s obrázkom 3.10 zreteľné, že rýchle zmeny hodnôt pri vyšších rýchlostiach sú účinne potlačené. Problémom je rušenie na nízkych frekvenciách, ktoré je prítomné na nahrávkach, získaných pri meraní na automobile (obrázok 2.14) a ktoré budeme v rámci tejto práce nazývať *záchvevami*. Prítomnosť záchvevov je spojená výhradne s týmto meracím usporiadaním a možnou príčinou je pohyb automobilu vo *vertikálnom* smere. Tieto prirodzene tlačia hodnotu prvého spektrálneho momentu nižšie, no hodnoty spektrálnej výkonovej hustoty na týchto pozíciách nepredstavujú „hrozbu“ pre maximum príslušného spektra.

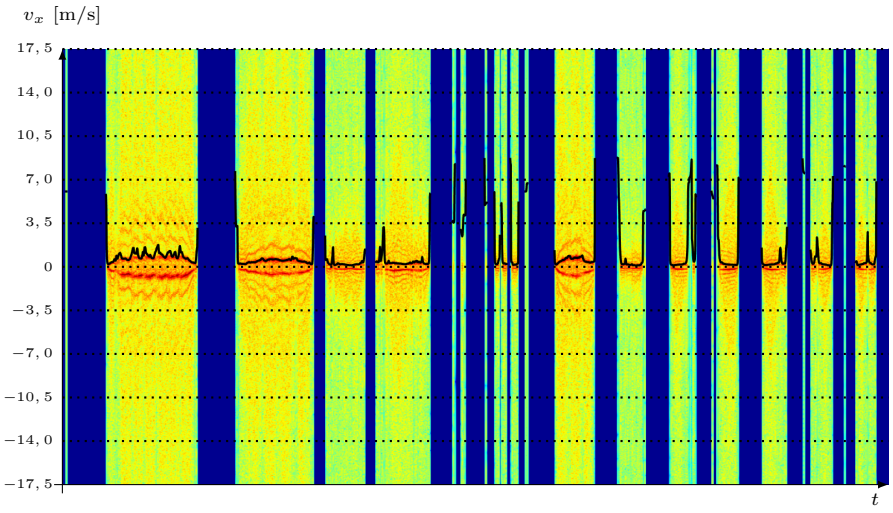
[xv] Po anglicky *spectral centroid*, z čoho vychádza i v tejto práci časté označenie C.

[xvi] Pre zavedenú konvenciu pôjde o body 0 až $17,56$ m/s s krokom $0,0473$ m/s.



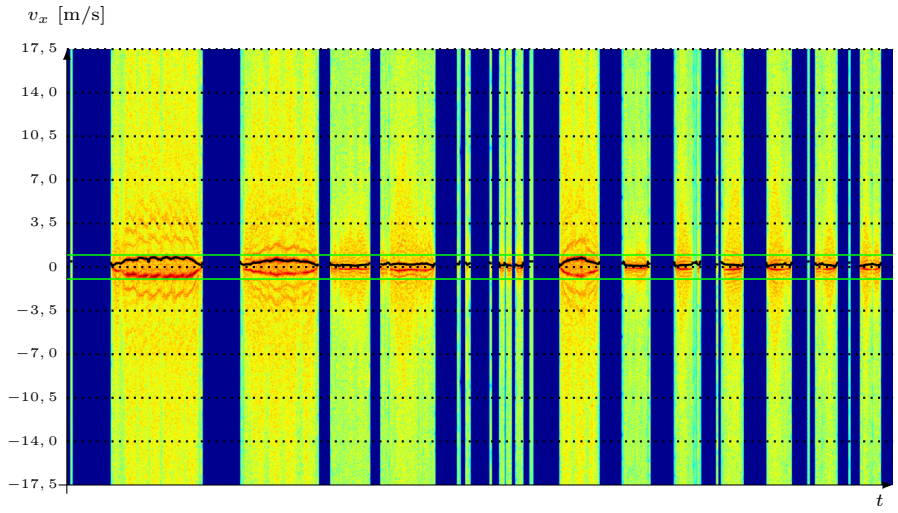
Obr. 3.20: Vyhodnotenie nahrávky $\mathcal{N}_2$ prvým spetrálnym momentom.

Výsledky prvého spektrálneho momentu pri nízkych rýchlostiach sú slabé z dôvodu ovplyvnenia pozadím a parazitnými vyššími hodnotami spektrálnej výkonovej hustoty na pozíciach vyšších harmonických rozdielového signálu. Toto možno sledovať na obrázku 3.21.



Obr. 3.21: Vyhodnotenie nahrávky $\mathcal{N}_1$ prvým spetrálnym momentom.

Pokiaľ však obmedzíme rozsah, z ktorého je C počítaný, výsledky sa – čo je nemálo intuitívne – výrazne zlepšia. Obmedzenie rozsahu výpočtu C na prvých 20 bodov, čo je rozsah, do ktorého sa zmestia všetky rýchlosti z nahrávky $\mathcal{N}_1$, prináša výsledok, zobrazený na obrázku 3.22.



Obr. 3.22: Vyhodnotenie nahrávky $\mathcal{N}_1$ prvým spetrálnym momentom z obmedzeným rozsahom indexov. Zelenými čiarami je vyznačená oblasť, ktorá odpovedá použitému rozsahu.

Tieto výsledky jasne naznačujú potrebu lokálnej adaptability vyhodnocovacieho mechanizmu. Rozhodnutie o výbere spôsobu vyhodnotenia absolútnej hodnoty rýchlosti vo finálnom programe medzi lokálne adaptabilným hľadaním maxím a lokálne adaptabilným prvým spektrálnym momentom sa u nás priklonilo na stranu prvého spektrálneho momentu, čo bolo podporené hlavne jeho odolnosťou voči širokému „pásmu“ okolo maximálnej hodnoty spektrogramu pri vyšších rýchlostiach.

3.3.2.2 Pásma pre lokálnu adaptabilitu

Hlavnou myšlienkou je využiť istý mechanizmus[xvii] na hrubý odhad rýchlosti na vyhodnocovanom podspektrograme. Na základe výsledkov tohoto mechanizmu zvolíme rozsah indexov (rýchlostí), z ktorých bude počítaný prvý spektrálny moment.

Rozsahy hrubých odhadov boli zvolené tak, aby

- Ich bol optimálny počet. Nie príliš málo, aby lokálna adaptabilita mala zmysel a nie príliš veľa, pretože každý z nich robí vyhodnotenie výpočtovo náročnejším.
- Aby prvé spektrálne momenty z nich na seba nadväzovali a v hraničných bodoch bolo možné medzi nimi „prepínať“ bez výrazného skreslenia.

Po vyskúšaní rôznych konštelácií sme za najlepšiu uznali tú, ktorá je daná tabuľkou 4.1 a obrázkom 4.6. Zatiaľčo nízke rýchlosti je potrebné opatriť jemnejším rozdelením indexov, pri rýchlostiach vyšších, než približne 1 m/s nemala

[xvii] Vo finálnom programe sú použité jednoduché neurónové siete, v tomto momente však na voľbe tohoto mechanizmu nezáleží.

lokálna adaptabilita výraznejší pozitívny vplyv, snáď len s výnimkou istého potlačenia vplyvu záchvevov. Z dôvodu lepšej náväznosti lokálnych prvých spektrálnych momentov bol zvolený nesúhlas medzi rozsahom indexov, z ktorých bol počítaný C a rozsahom rýchlostí, do ktorého nám aktuálny podspektrogram priradzuje spomínaný mechanizmus hrubého odhadu – rozsahy C sa čiastočne prekrývali, hoci rozsahy odhadov boli volené bez prekryvu.

3.3.2.3 Implementácia lokálnej adaptability

Ako najvhodnejší nástroj na priradenie podspektrogramu do rozmedzia rýchlostí sa hneď od začiatku zdala byť neurónová sieť. Podmienky pre jej využitie sú tu priam ideálne – značné rozdiely medzi polohami maxím PSD v jednotlivých pásmach, úzke hranice zón evokujúce možnosť použitia pomerne úzkeho filtra na vyhladenie ambiguitného priebehu a i to, že pre zariadenie, do ktorého by mohol byť náš softvér použitý, by nemalo byť problém zapamätať si koeficienty siete, no naopak ocení jednoduchosť výpočtu.

Zvolili sme prístup s dvoma samostatnými neurónovými sieťami, pričom výstup tej prvej udáva príslušnosť (0) či neprísluš nosť (1) vstupného podspektrogramu do dolného pásma rýchlostí a výstup tej druhej jeho príslušnosť alebo neprísluš nosť do zjednotenia dolného a stredného pásma. Pokiaľ obe siete ohlásia nesúlad vstupného spektrogramu s príslušnosťou v ich pásmach, tento podspektrogram bude vyhodnotený pomocou globálneho prvého spektrálneho momentu. Tréning oboch použitých sietí je zhrnutý v dodatku D.

Postup priradenia absolútnej hodnoty rýchlosti aktuálnemu spektru je priamočiary:

1. Pre aktuálne spektrum vypočítame tri rôzne prvé spektrálne momenty – každý z nich sa týka jedného z rozmedzí absolútnych hodnôt rýchlosti. Ten, pre najnižšie rýchlosti, označíme $C_{<0,24}$, ten, pre rozmedzie $|v_x| \in \langle 0,2367;\ 1,0832\rangle$ m/s $C_{<1,08}$ a *globálny* prvý spektrálny moment C_g.

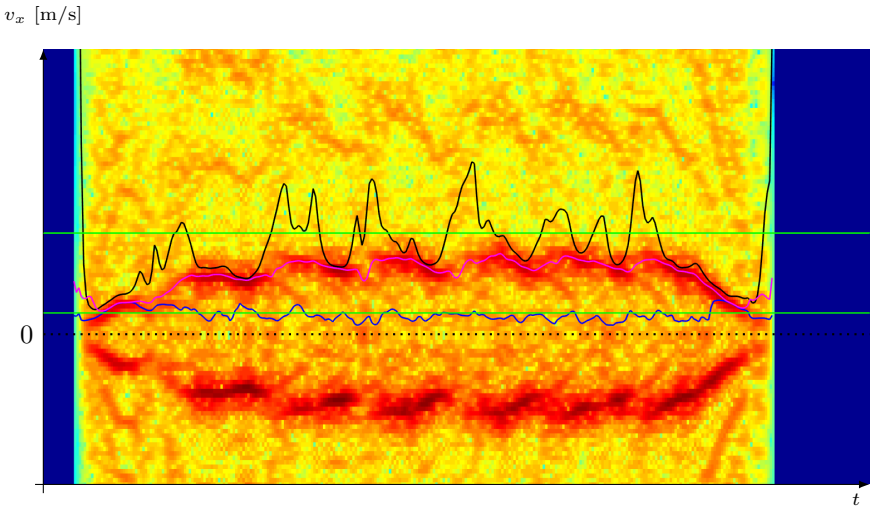
2. Necháme sieť vyhodnotiť príslušnosť spektra k jednotlivým rozmedziam. Na základe jej výsledkov priradíme na výstup hodnotu, danú tabuľkou 3.2.

Sieť pre najnižšie rýchlosti	0	0	1	1
Sieť pre stredné pásmo	0	1	0	1
Výstup	$C_{<0,24}$	$C_{<1,08}$	!	C_g

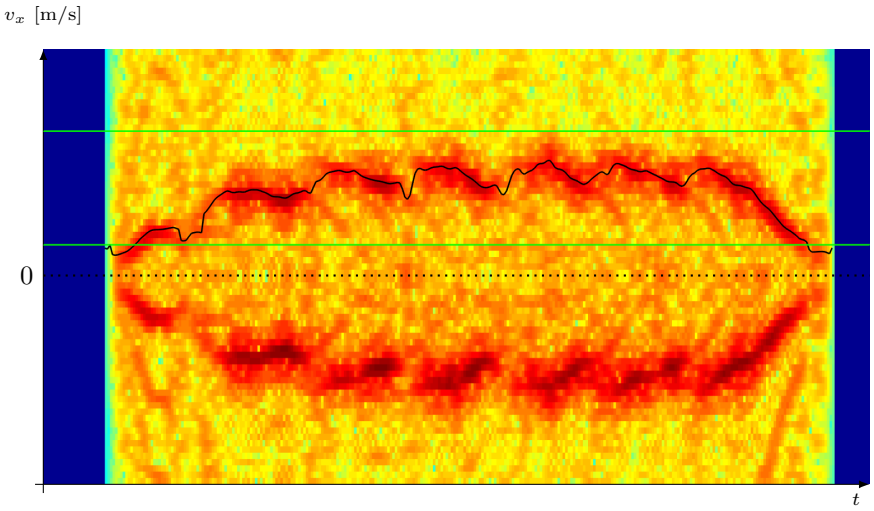
Tab. 3.2: Voľba prvého spektrálneho momentu na základe výsledkov neurónových sietí. Stav, označený výkričníkom, v ideálnom prípade nemôže nastať – reálne však nastať môže a ak sa tak stane, priradíme mu hodnotu *globálneho* prvého spektrálneho momentu.

Na obrázkoch 3.23 a 3.24 môžeme vidieť výsledok – Na prvom obrázku jednotlivé farebné čiary predstavujú priebeh spektrálnych momentov z našich troch rozmedzí[xviii]. Na tom druhom je potom čiara jediná, ktorá vznikla obmedzením sa na jednu hodnotu v jednom časovom bode, pričom výber zdrojového priebehu momentu je daný sieťami a tabuľkou 3.2.

[xviii] To, ktorá čiara prislúcha ktorému rozmedziu, je, veríme, zrejmé.



Obr. 3.23: Tri priebehy spektrálnych momentov z troch rozmedzí, zobrazené samostatne. Smer pohybu je zo spektorgramu v pozadí očividne opačný, než ten, ktorý odpovedá polovici s vyznačenými priebehmi – v tejto časti sa však zaoberáme rýdzo absolútnou hodnotou rýchlosti.



Obr. 3.24: Tri priebehy spektrálnych momentov z troch rozmedzí, zlúčené do jediného výsledku.

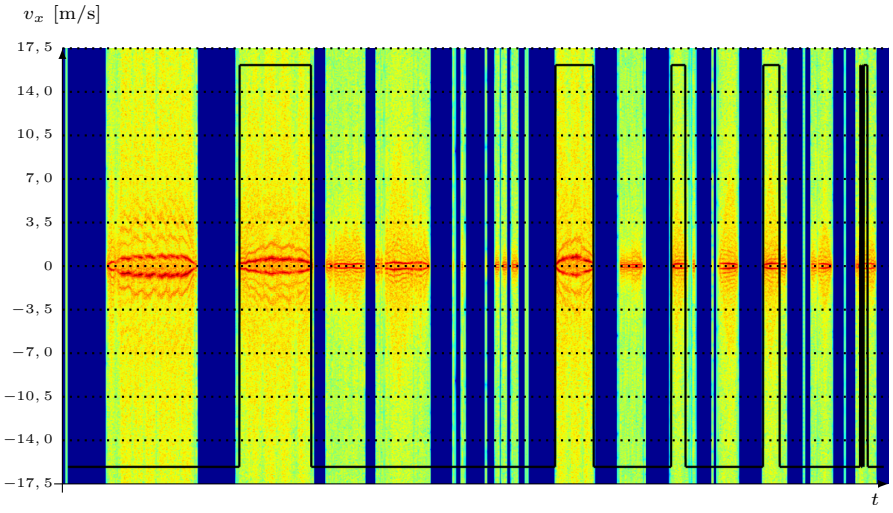
3.4 Algoritmy pre určenie smeru pohybu

Po vyskúšaní viacerých možností pre komplexné vyhodnotenie použitím jediného typu mechanizmu sa ukázalo, že žiaden z nich nedosahoval uspokojivé výsledky. Pristúpili sme preto k rozdeleniu tejto úlohy na podúlohy. Riešenie

tej druhej z dvoch hlavných, určenia smeru pohybu, si priblížime v tejto podkapitole.

3.4.1 Súčty spektrálnej výkonovej hustoty

Ako bolo ukázané v časti 3.2.1, prostým hľadaním maxím spektrogramu nedostaneme vždy správne určenie smeru pohybu, pričom problematické sú najmä (a takmer výhradne) úseky veľmi nízkych rýchlostí. Názornejšia ilustrácia sa nachádza na obrázku 3.25, kde je čiernou čiarou vyznačený smer pohybu, vyhodnotený na základe hľadania maxím spektrogramu. Výsledok bol ešte filtrovaný modusovým filtrom o 35-prvkovom okne, o ktorom je písané viac v dodatku C.



Obr. 3.25: Spektrogram z nahrávky $\mathcal{N}_1$ a vyznačenie smeru pohybu na ňom, pričom smer vpred je indikovaný čiernou čiarou na kladnej pozícii, smer vzad ňou na zápornej pozícii a úsek bez pohybu čiarou na nulových hodnotách.

Na spektrogramoch, počítaných spôsobom, uvedeným v časti 3.2, sú prítomné vyššie harmonické signálov na dopplerovskej frekvencii. Tieto sú, prirodzene, silnejšie, na tej strane spektra, ktorá odpovedá smeru pohybu v časovom úseku, kedy je toto spektrum počítané. Hoci sú vďaka nedokonalému fázovému posunu signálových zložiek[xix] vyššie harmonické prítomné na *oboch* stranách, na tej „nesprávnej“ sú zväčša na veľmi nízkych úrovniach. Poskytujú nám teda dodatočnú informáciu o smere pohybu, ktorú pri prostom hľadaní maxím strácame. To nás viedlo k rozoznávaniu smeru pomocou súčtov spektrálnej výkonovej hustoty na jednotlivých stranách spektra. V každom časovom bode sme nechali vypočítať súčet PSD na indexoch, odpovedajúcich kladnému smeru pohybu, Σ_+, súčet PSD na indexoch, odpovedajúcich zápornému smeru pohybu, Σ_- a smer pohybu bol potom daný hodnotou

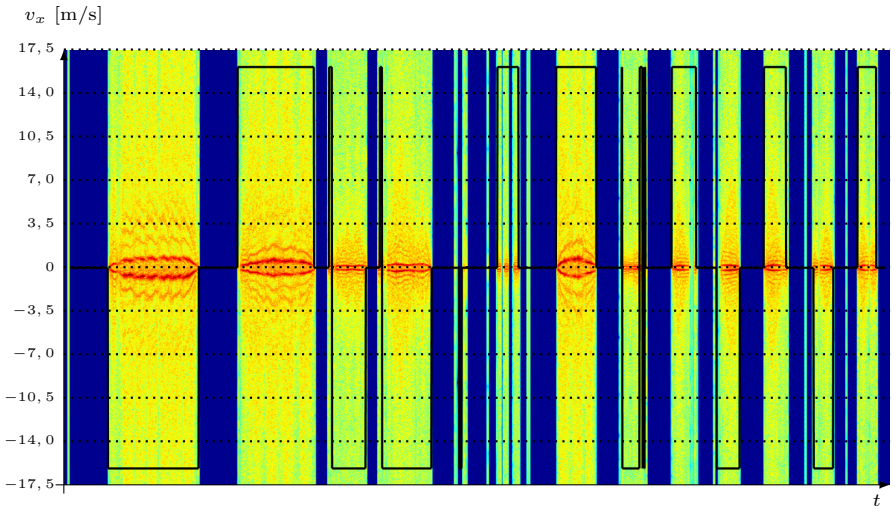
$$\operatorname{sgn}(\Sigma_+ - \Sigma_-)\,, \tag{3.8}$$

[xix] Ich fázový posun sa podľa [10] môže hýbať až v rozmedzí $90° \pm 30°$.

kde sgn(·) značí znamienkovú funkciu, definovanú ako

$$\mathrm{sgn}(x) = \begin{cases} 1 & \text{pre } x > 0 \\ 0 & \text{pre } x = 0 \\ -1 & \text{pre } x < 0 \end{cases} . \tag{3.9}$$

Výsledok tohoto postupu na nahrávke $\mathcal{N}_1$ po filtrácii 35-bodovým modusovým filtrom je na nasledujúcom obrázku.



Obr. 3.26: Spektrogram z nahrávky $\mathcal{N}_1$ a vyznačenie smeru pohybu na ňom, získaného pomocou porovnávania súčtov spektrálnej výkonovej hustoty medzi stranmi spektra.

Je zrejmé, že výsledok z obrázku 3.26 je napriek nedokonalostiam lepší, než výsledok po použití jediného maxima na indikáciu pohybu (obrázok 3.25).

Po tomto výsledku sme experimentovali i s výberom niektorých rozmedzí indexov[xx], z ktorých by boli počítané súčty PSD, z ktorých uvádzame

- Maximum a dvojnásobok maxima
- Okolie maxima o rôznej šírke
- Samostatne n-násobok maxima
- Náhodný výber indexov pre každý z časových bodov.

Žiadne z nich však nedosiahlo výsledky lepšie, než globálny súčet všetkých hodnôt na každej strane. Toto vyhodnotenie smeru pohybu sa vďaka svojej jednoduchosti a dobrým výsledkom javí ako veľmi vhodné na použitie v našom programe.

[xx]Tieto rozmedzia sú uvedené iba pre jednu stranu spektra – princípom je predsa porovnávať súčty PSD v týchto rozmedziach práve medzi dvomi stranami.

3.4.2 Využitie nevyváženia signálových zložiek

S odvolaním sa na časť 1.3 by v ideálnom prípade mali byť súfázová (I) a kvadratúrna (Q) zložka fázovo posunuté *presne* o $\pi/2$ rad, mali by mať rovnakú amplitúdu A a mali by mať *nulovú* jednosmernú zložku. V reálnom prípade to tak ale nie je. Zatiaľ čo pre ideálny prípad možno v duchu obrázku 1.5 písať

$$\begin{aligned} I(k) &= A\,\sin(\varphi(k)) \\ Q(k) &= A\,\cos(\varphi(k))\,, \end{aligned} \tag{3.10}$$

v reálnom prípade je situácia trochu iná a vo všeobecnosti by sa dalo napísať

$$\begin{aligned} I_e(k) &= c_1\,A\,\sin(\varphi(k)+\xi_1)+d_1 \\ Q_e(k) &= c_2\,A\,\cos(\varphi(k)+\xi_2)+d_2\,, \end{aligned} \tag{3.11}$$

pričom index $_e$ je takto volený s ohľadom na anglické slovo *error*. Vzhľadom na to, že amplitúda signálu nie je pre nás podstatnou informáciou[xxi] a tiež to, že jednu zo zložiek môžeme považovať za *fázovo bezhybnú* a chybu priradiť iba tej druhej, môžeme niektoré z parametrov v rovniciach (3.11). Bez ujmy na všeobecnosti vylúčime parametre c_2 a ξ_2, čím rovnice (3.11) prejdú na tvar

$$\begin{aligned} I_e(k) &= c_1\,A\,\sin(\varphi(k)+\xi_1)+d_1 \\ Q_e(k) &= A\,\cos(\varphi(k))+d_2\,. \end{aligned} \tag{3.12}$$

Spôsobom, popísaným v [11] môžeme získať hodnoty neznámych z rovnice (3.12). Zaveďme operátor *strednej hodnoty* $\mathrm{Av}\{\cdot\}$, aplikovaný na diskrétny signál $s(k)$, vzniknutý zo spojitého *periodického* signálu $s(t)$ o frekvencii f vzorkovaním vzorkovacou frekvenciou f_s ako

$$\mathrm{Av}\{s(k)\} = \frac{f}{N_p\,f_s}\sum_{i=1}^{N_p f_s/f} s(i)\,, \tag{3.13}$$

kde N_p je ľubovoľný počet periód, na ktorých počítame strednú hodnotu. Predpokladáme pritom, že perióda signálu $s(t)$ je celočíselným násobkom vzorkovacej periódy[xxii]. Dôležité je tiež zdôrazniť, že signály I a Q by mali byť pre výpočet všetkých parametrov, spomínaných v tejto časti, frekvenčne konštantné.

Hľadajme strednú hodnotu reálnych signálov I_e a Q_e z rovníc (3.12)[xxiii].

$$\begin{gathered} \mathrm{Av}\{\overbrace{c_1\,A\,\sin(\varphi)\,\cos(\xi_1)+c_1\,A\,\cos(\varphi)\,\sin(\xi_1)}^{\text{Stredná hodnota nulová}}+d_1\} = d_1 \\ \mathrm{Av}\{Q_e\} = \cdots = d_2\,. \end{gathered} \tag{3.14}$$

Jednosmerný posun zložiek sme teda schopní korigovať pomocou merania strednej hodnoty radarových signálov I_e a Q_e. Problémom je, že i táto úvaha predpokladá konštantnú frekvenciu prijímaného komplexného signálu. Predpokladajme relatívne ťažko dosiahnutý prípad, kedy sa v rámci jednej sekundy zmení rýchlosť o polovicu skúmaného rozsahu, teda $\Delta v_x = 17$ m/s. Po prepočítaní pomocou vzťahu (1.3) dostaneme odpovedajúcu frekvenčnú zmenu

[xxi] Predpokladáme, samozrejme, že je dostatočná na úspešný príjem odrazeného signálu.

[xxii] V reálnom prípade sa toto podarí len zriedka; Definíciu strednej hodnoty však možno poupraviť napríklad pomocou operátoru dolnej celej časti.

[xxiii] Napriek zjavnému výsledku uvádzame výpočet.

o $\Delta f \approx 2$ kHz. Potom za jednu vzorkovaciu periódu sa frekvencia zmení o $\Delta f_1 = \frac{2000}{44100} \approx 0,045$ Hz. Ak dokážeme strednú hodnotu vyhodnotiť z desiatich signálových periód, frekvencia sa počas doby jej vyhodnocovania zmení o $10 \cdot \Delta f_1 = 0,45$ Hz. Prehlásme teda, že v tomto veľmi zjednodušenom prípade by bolo možné jednosmernú zložku pomocou merania strednej hodnoty skutočne korigovať.

Označme zložky z rovníc (3.12) po tejto úprave ako I_{ec} a Q_{ec}:

$$\begin{pmatrix} I_{ec} \\ Q_{ec} \end{pmatrix} = \begin{pmatrix} c_1 \cos(\xi_1) & c_1 \sin(\xi_1) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} I \\ Q \end{pmatrix}. \tag{3.15}$$

Uvedomme si, že jediným záchytným bodom pre korekciu nevyváženia I a Q signálov je priebeh ich samotných. Ďalšou štatistickou veličinou – strednou hodnotou *kvadrátu* súfázovej zložky – sme schopní určiť parameter c_1 z rovníc (3.15).

$$\begin{aligned} \mathrm{Av}\{I_{ec}^2\} = c_1^2 A^2 \sin^2(\varphi(k) + \xi_1) = \frac{c_1^2 A^2}{2} \mathrm{Av}\{1 - \cos(2(\varphi(k) + \xi_1))\} = \frac{c_1^2 A^2}{2} \\ c_1 = \frac{\sqrt{2}}{A} \sqrt{\mathrm{Av}\{I_{ec}^2\}} \end{aligned} \tag{3.16}$$

S pomocou strednej hodnoty súčinu súfázovej a kvadratúrnej zložky je ďalej možné zistiť hodnotu parametra $\xi_1 \in (0, \frac{\pi}{2})$:

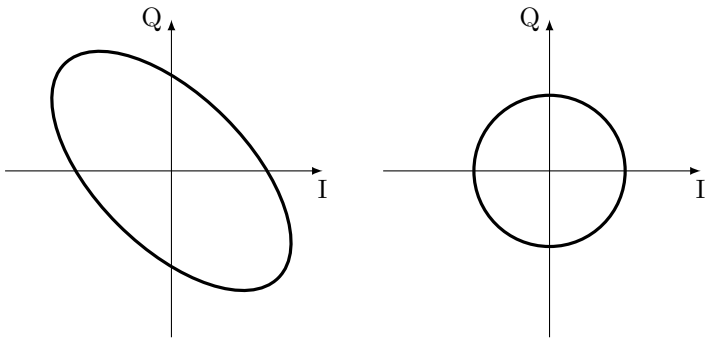
$$\begin{aligned} \mathrm{Av}\{I_{ec}\, Q_{ec}\} = c_1 A^2 \mathrm{Av}\{(\sin(\varphi(k)) \cos(\xi_1) + \sin(\xi_1) \cos(\varphi(k)) \cos(\varphi(k)))\} = \\ = c_1 A^2 \sin(\xi_1) \frac{1}{2} \mathrm{Av}\{(1 + \cos(2\varphi(k)))\} = \frac{c_1 A^2 \sin(\xi_1)}{2} \\ \sin(\xi_1) = \frac{2\,\mathrm{Av}\{I_{ec}\, Q_{ec}\}}{c_1 A^2}. \end{aligned} \tag{3.17}$$

Hodnotu $\cos(\xi_1)$ možno potom jednoducho dopočítať ako

$$\cos(\xi_1) = \sqrt{1 - \sin^2(\xi_1)}. \tag{3.18}$$

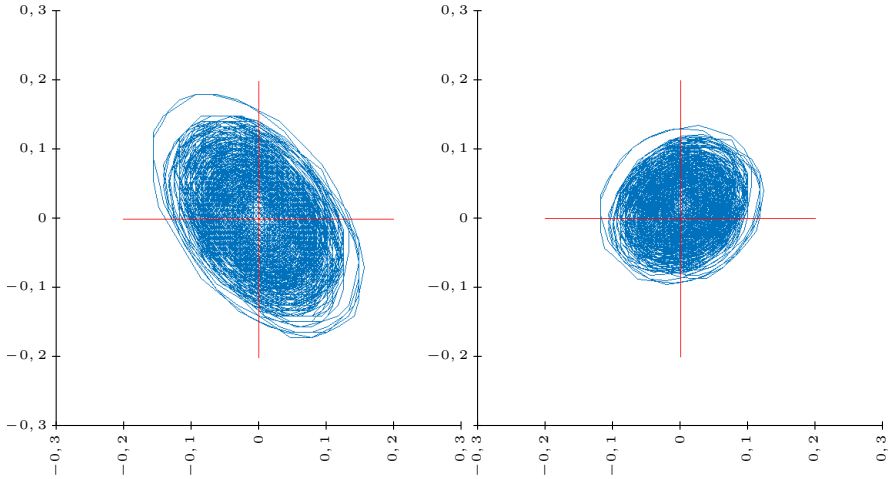
Pretože pri ideálne „zladených“ zložkách I a Q nás zaujíma iba frekvencia a fázový posun, daný binárne ako *kladný* alebo *záporný*, môžeme položiť $A \stackrel{\downarrow}{=} 1$ a zjednodušiť tak predchádzajúce vzťahy.

Korekcia bola vyskúšaná na nahrávke $\mathcal{N}_{X1}$, získanej pri usporiadaní z obrázku 2.13 a nahrávke $\mathcal{N}_1$ z usporiadania z obrázku 2.12. Po načítaní nahrávky do MATLABu pomocou skriptu E.12 boli vypočítané približné hodnoty indexov *maxím* signálových zložiek (skript E.13), aby bolo možné spočítať počet signálových periód v jednom výrezovom okne. Pomocou kódu E.14 (vrátane zakomentovanej časti), spusteného po načítaní pracovných dát, možno vidieť, ako vyzeral graf zložiek I a Q v komplexnej rovine *pred* a *po* korekcii.



Obr. 3.27: Približný náčrt signálov z nahrávky $\mathcal{N}_{C1}$ v komplexnej rovine pred a po korekcii, ktorej koeficienty boli dopočítavané z tej istej nahrávky. Pre presné vykreslenie je možné využiť zakomentované riadky na konci kódu E.14.

Po vyskúšaní korekčných koeficientov na signále o relatívne malej šírke pásma sme pristúpili k ich testovaniu na nahrávke s širším frekvenčným rozsahom. Ide o nahrávku $\mathcal{N}_{X2}$. Pred a po použití predchádzajúcich korekčných koeficientov na ňu vyzerali grafy komplexných signálov tak, ako je zobrazené na obrázku 3.28

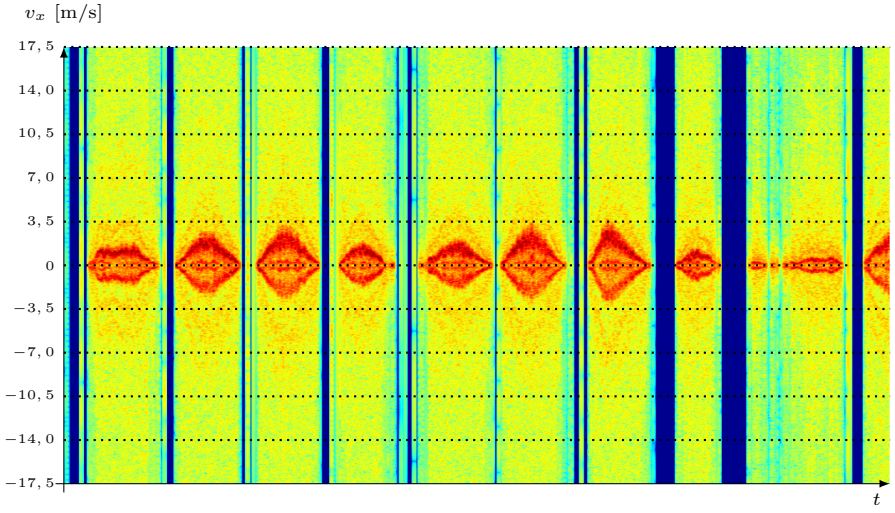


Obr. 3.28: Výsledky použitia korekčných koeficientov, vypočítaných na základe úzkopásmovej nahrávky $\mathcal{N}_{X1}$ na signál zo „širokopásmovej“ nahrávky $\mathcal{N}_{X2}$. Vľavo je zobrazený nekorigovaný signál, vpravo signál korigovaný.

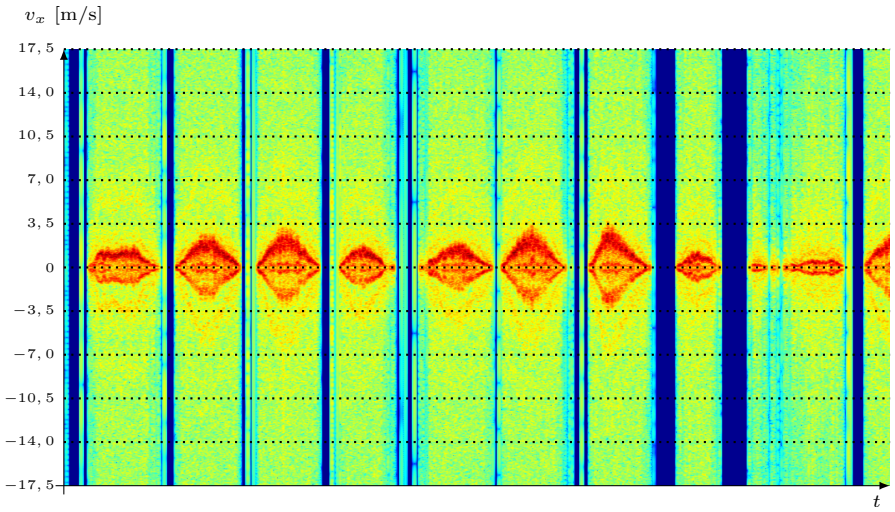
Je vidieť, že hoci sme aplikáciou takejto korekcie signálovú elipsu vyrovnali, pribudla nezanedbateľná jednosmerná zložka Q signálu. Takáto korekcia je vzhľadom na podstatu dopplerovského merania, kedy je výsledok založený na zmene frekvencie, nevhodná – ideálnym signálom pre ňu je totiž signál frekvenčne nemenný.

Pretože z obrázku 3.28 je vidieť, že jednosmerná zložka oboch signálov je minimálna a väčším problémom je fázový posun, núka sa využiť oneskorenie

jedného zo signálov na vyrovnanie elipsy. Pre nahrávku $\mathcal{N}_{X2}$ skutočne oneskorenie kvadratúrnej zložky o 22 vzoriek spôsobilo vyrovnanie signálovej elipsy a zlepšenie rozlíšiteľnosti rýchlosti na spektrograme – spektrogram pred korekciou a po nej je na obrázkoch 3.29 a 3.30.



Obr. 3.29: Spektrogram nahrávky $\mathcal{N}_{X2}$ pred korekciou oneskorením jednej zo zložiek.



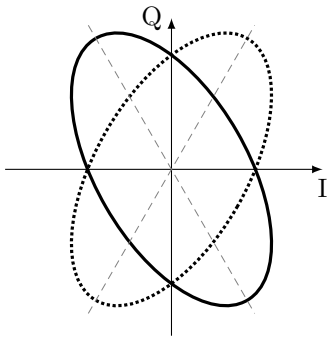
Obr. 3.30: Spektrogram nahrávky $\mathcal{N}_{X2}$ po korekcii oneskorením jednej zo zložiek.

Problémom opäť je, že jedna hodnota oneskorenia jednej zo zložiek ja platná iba pre jednu frekvenciu[xxiv] a pre jej použitie by bolo potrebné opäť imple-

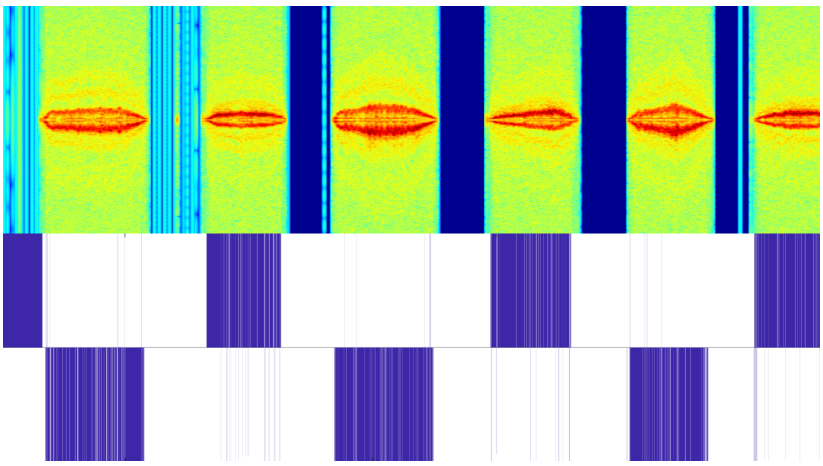
[xxiv] V skutočnosti je ich nekonečne veľa, nakoľko sa periodicky opakujú.

mentovať mechanizmus lokálnej adaptability. Vzhľadom na iba malé zlepšenie oproti situácii bez korekcie sme sa rozhodli sa touto myšlienkou vo finálnom programe nezaoberať.

Naopak môžeme fázové neideality nekorigovať a využiť ich v náš prospech. Základnou ideou je fakt, že pri zmene rýchlosti z kladnej na zápornú sa fáza kvadratúrnej vlny zmení o π radiánov. Nevyváženie kanálov navyše pridá fázový posun, veľkostne zhodný s posunom pri kladnej rýchlosti, no s *opačným znamienkom*. Signálová elipsa sa tak bude pri nekonečne rýchlych zmenách rýchlosti rovnakej absolútnej hodnoty z kladnej na zápornú, resp. naopak *preklápať* cez reálnu, resp. imaginárnu os[xxv]. Ilustrácia je uvedená na obrázku 3.31.



Obr. 3.31: Znázornenie zmeny signálovej elipsy pri zmene smeru rýchlosti pri konštantnej *absolútnej hodnote* rýchlosti.



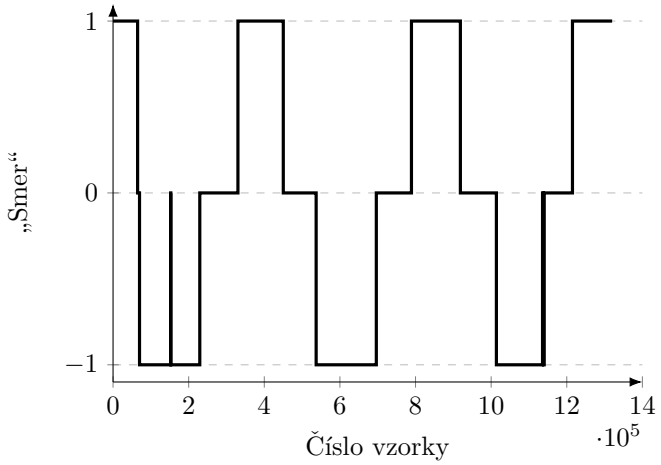
Obr. 3.32: Výsledok určenia smeru na nahrávke $\mathcal{N}_{X3}$ pomocou znamienok I signálu v maximách Q signálu. Spektrogram a výsledok sú uložené z Matlabu a položené nad seba.

[xxv] Preklopenie podľa imaginárnej či reálnej osi je analogické, v signále totiž nepredpokladáme jednosmernú zložku.

Situácia sa teda javí tak, že smer pohybu je možné rozlíšiť na základe znamienka reálnej časti komplexného radarového signálu v čase, kedy jeho imaginárna zložka nadobúda maximum.

Prvá implementácia prebehla v MATLABe a je uvedená v kóde E.15. Výsledky predčili očakávania a napríklad vyhodnotenie nahrávky $\mathcal{N}_{X3}$, ktorá pochádza z blízkozónneho usporiadania merania z obrázku 2.13, je uvedené na obrázku 3.32.

Z obrázku 3.32 je zrejmé, že tento spôsob určenia smeru má pomerne veľké množstvo výpadkov, ktoré dosť silno závisí i na presnosti určenia maxím signálových zložiek. Z hľadiska času sa však tieto výpadky javia výrazne kratšie, než úseky správnych hodnôt, čo nás vedie k použitiu kĺzavej filtrácie. Po vyskúšaní niekoľkých typov filtru sa naša pozornosť zamerala na filter *modusový*. Jeho priblíženie a poňatie, používané v tejto práci, je zhrnuté v dodatku C. Doplnenie programu na vyhodnotenie smeru pohybu o modusový filter sa nachádza v kóde E.16, výsledok potom na obrázku 3.33.



Obr. 3.33: Výsledok určenia smeru na nahrávke $\mathcal{N}_{X3}$ pomocou znamienok I zložky v maximách Q zložky po filtrácii modusovým filtrom o šírke okna 3000 bodov. Obrázok je výsledkom kódov E.15 a E.16.

3.4.2.1 Pobitový algoritmus na určenie smeru

Implementácia vyhodnotenia smeru pohybu z prechádzajúcich riadkov, využívajúca nevyváženie I a Q zložiek, bola vhodná na vyskúšanie fungovania takejto idey. Pre vyhodnotenie v reálnom čase však doterajší postup nie je vhodný. Pristúpili sme preto k návrhu algoritmu, ktorý bude schopný pracovať v takmer reálnom čase, s dôrazom na čo najmenšie oneskorenie. Navrhnutý algoritmus je uvedený v algoritme 3.

Algoritmus 3 Určenie smeru pohybu na základe nevyváženia I a Q zložiek

```
1: if Inicializácia ešte neprebehla then
2:         if Nový bit je nenulový then
3:                 if Sme v nenulovom úseku then
4:                         if Prvé minimum abs. hodnoty l je už nastavené then
```

```
5:                          if Druhé minimum abs. hodnoty l je už nastavené then
6:                              Zmenši hodnotu prvého minima o 1
7:                              Zmenši hodnotu druhého minima o 1
8:                          else
9:                              Zmenši hodnotu prvého minima o 1
10:                         end if
11:                     else
12:                         Nič nerob
13:                     end if
14:                 else                                    ▷ Sme v nulovom úseku
15:                     if Druhé minimum abs. hodnoty l je už nastavené then     ▷ Pokiaľ sme v nulovom úseku, prvé minimum už nastavené byť musí
16:                         Zmenši hodnotu prvého minima o 1
17:                         Zmenši hodnotu druhého minima o 1
18:                         Zmeň indikátor povahy úseku na nenulový
19:                     else
20:                         Zmenši hodnotu prvého minima o 1
21:                         Zmeň indikátor povahy úseku na nenulový
22:                     end if
23:                 end if
24:             else                                        ▷ Nový bit je nulový
25:                 if Sme v nenulovom úseku then
26:                     if Prvé minimum abs. hodnoty l je už nastavené then
27:                         if Druhé minimum abs. hodnoty l je už nastavené then
28:                             Zmenši hodnotu prvého minima o 1
29:                             Zmenši hodnotu druhého minima o 1
30:                             Prehlás výpočet za inicializovaný
31:                             Nájdi znamienko r-signálu v maxime l-signálu a ulož ho
32:                             Zmeň hodnotu indexu prvého minima na hodnotu indexu druhého minima a hodnotu indexu druhého minima zas na hodnotu posledného indexu
33:                             Zmeň indikátor povahu úseku na nulový
34:                         else
35:                             Prehlás druhé minimum za nastavené
36:                             Zmenši hodnotu prvého minima o 1
37:                             Zmenši hodnotu druhého minima o 1
38:                             Zmeň indikátor povahy úseku na nulový
39:                         end if
40:                     else
41:                         Prehlás prvé minimum za nastavené
42:                         Zmenši hodnotu prvého minima o 1
43:                         Zmeň indikátor povahy úseku na nulový
44:                     end if
45:                 else                                    ▷ Sme v nulovom úseku
46:                     if Druhé minimum abs. hodnoty l je už nastavené then     ▷ Pokiaľ sme v nulovom úseku, prvé minimum už nastavené byť
```

```
    musí
47:                             Zmenši hodnotu prvého minima o 1
48:                             Zmenši hodnotu druhého minima o 1
49:                         else
50:                             Zmenši hodnotu prvého minima o 1
51:                         end if
52:                     end if
53:                 end if
54: else                                          ▷ Inicializácia už prebehla
55:                 if Nový bit je nenulový then
56:                     if Sme v nenulovom úseku then
57:                         Zmenši hodnotu prvého minima o 1
58:                         Zmenši hodnotu druhého minima o 1
59:                     else
60:                         Zmenši hodnotu prvého minima o 1
61:                         Zmenši hodnotu druhého minima o 1
62:                         Zmeň indikátor povahy úseku na nenulový
63:                     end if
64:                 else                                      ▷ Nový bit je nulový
65:                     if Sme v nenulovom úseku then
66:                         Zmenši hodnotu prvého minima o 1
67:                         Zmenši hodnotu druhého minima o 1
68:                         Nájdi znamienko r-signálu v maxime l-signálu a ulož
    ho
69:                         Zmeň hodnotu indexu prvého minima na hodnotu in-
    dexu druhého minima a hodnotu indexu druhého minima zas na hodnotu
    posledného indexu
70:                         Zmeň indikátor povahy úseku na nulový
71:                     else                                ▷ Sme v nulovom úseku
72:                         Zmenši hodnotu prvého minima o 1
73:                         Zmenši hodnotu druhého minima o 1
74:                     end if
75:                 end if
76: end if
```

Niektoré pojmy z tohoto popisu iste nie sú zrejmé a je preto na mieste vysvetlenie:

Nulový či nenulový *úsek* znamená, že aspoň jeden predošlý bit odpovedal nulovej či nenulovej hodnote. Nulový úsek predstavuje priechod signálu nulou. Keďže je ale vzorkovacia frekvencia ďaleko vyššia, než frekvencia dopplerovská, nulové body sa zobrazia skutočne ako *rady* núl. Nenulovým úsekom by potom malo byť všetko ostatné.

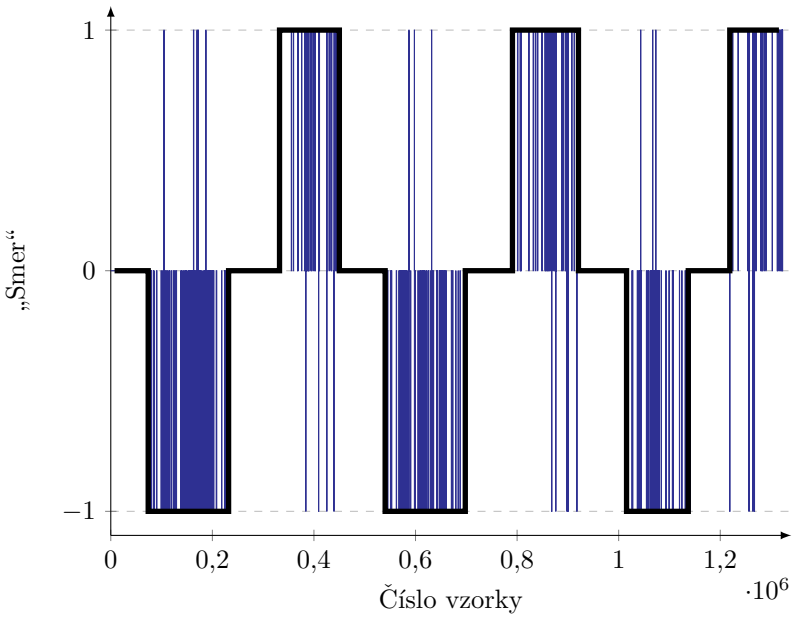
Prvé a druhé minimum sú minimá absolútnych hodnôt vzoriek súfázovej zložky. *Prvé* minimum pritom nazývame vyhodnotené minimum, ktoré prišlo do bufferu ako *predposledné* a *druhé* minimum to, ktoré prišlo do bufferu ako *posledné*.

Prvé a druhé maximum sú maximá absolútnych hodnôt vzoriek kvadratúrnej zložky medzi prvým a druhým minimom súfázovej zložky, resp. medzi

druhým minimom súfázovej zložky a posledným prvkom bufferu. Tieto sú vyhľadávané v riadkoch 31 a 68 algoritmu 3. Medzi nimi je potom vyhľadávané maximum vzoriek súfázovej zložky a v ňom sa pozeráme na znamienko vzorky zložky kvadratúrnej.

Inicializovaný výpočet znamená, že do bufferu prišli už aspoň 3 tesne za sebou nasledujúce začiatky radov miním vzoriek súfázovej zložky.

Implementácia v MATLABe sa nachádza v kóde E.17. Po vyskúšaní na vzorke $\mathcal{N}_{X3}$, ktoré bolo namerané v blízkej zóne, sa zdalo, že funguje výborne – výsledok sa nachádza na obrázku 3.34.

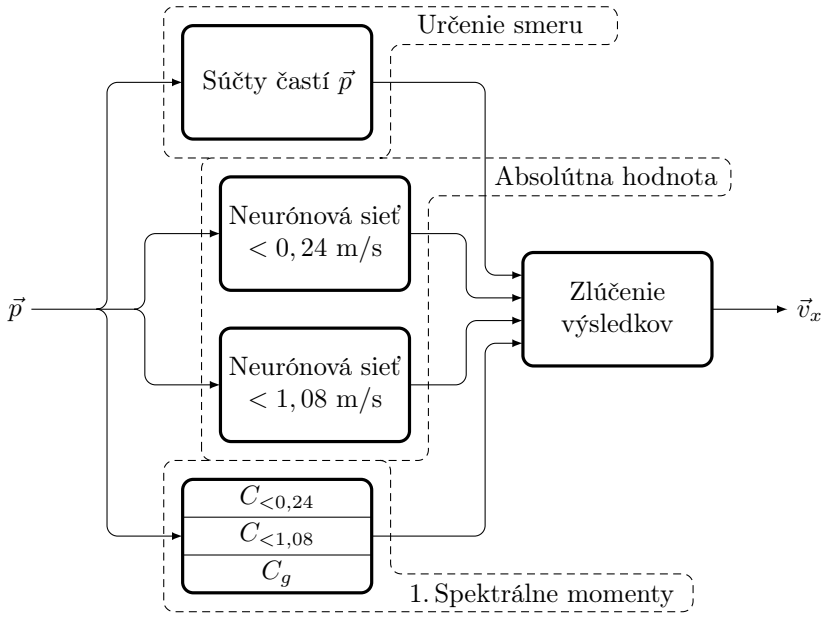


Obr. 3.34: Výsledok aplikácie vyhodnotenia dát $\mathcal{N}_{X3}$ algoritmom 3. Hrubá čierna čiara zobrazuje výsledok modusovej filtrácie o šírke okna 15000 bodov. Skript, ktorého výsledkom je obrázok, sa nachádza v kóde E.17.

Žiaľ pri meraniach vo vzdialenej zóne bolo použitie tohoto algoritmu bez dobrých výsledkov. Nielen, že sme nedosahovali dostatočnú jednoliatosť výsledkov kvôli neschopnosti zvoliť vhodnú dĺžku okna modusového filtru, ale častokrát ani samotné znamienka kvadratúrnej zložky v nájdených maximách zložky súfázovej nezodpovedali tomu, čo by sme chceli. Pre reálne merania je teda tento spôsob určenia smeru, žiaľ, nevhodný.

3.5 Výber vyhodnocovacích mechanizmov

Preskúmali sme výsledky niekoľkých spôsobov naloženia so spektrom pre zisk informácií o absolútnej hodnote rýchlosti, ktorej odpovedá a smeru pohybu, ktorý je v ňom zachytený. Do našeho komplexného softvéru však využijeme iba niektoré z nich. Po mnohých pokusoch dosahovalo najlepšie výsledky usporiadanie ktoré je schématicky znázornené na obrázku 3.35.



Obr. 3.35: Schéma prechodu vektoru spektrálnej výkonovej hustoty, $\vec{p}$, vyhodnocovacími mechanizmami, ktoré sú použité vo finálnom programe.

Rozhodli sme sa pre použitie

Súčtov PSD pre vyhodnotenie smeru pohybu. Tento mechanizmus vykazoval spomedzi skúmaných jednoznačne najlepšie výsledky. Tréning neurónovej siete, ktorá by sa starala o vyhodnotenie smeru pohybu, bol neúčinný – dôvodom je hlavne to, že každý z podspektrogramov je dobre parametrizovateľný na základe fyzikálneho princípu[xxvi] a odchýlky od predpokladaného priebehu, ktoré sieť môže použiť na kategorizáciu vstupného podspektrogramu, sú len neželaným šumom. Na vyhodnotenie nahrávok z usporiadaní, kedy bolo anténne pole senzoru mimo blízkej zóny, nebolo účinné použiť ani algoritmus 3, ktorý využíva nevyváženie signálov súfázovej a kvadratúrnej časti.

Prvého spektrálneho momentu na určenie absolútnej hodnoty rýchlosti, pričom tento mechanizmus bol implementovaný s istou lokálnou adaptabilitou. Použie maxím spektrogramu sme zavrhli z dôvodu problematickej filtrácie, i keď v niektorých oblastiach bolo účinnejšie ako prvý spektrálny moment. Neurónové siete, trénované na dátach z našich generátorov, neboli úspešné.

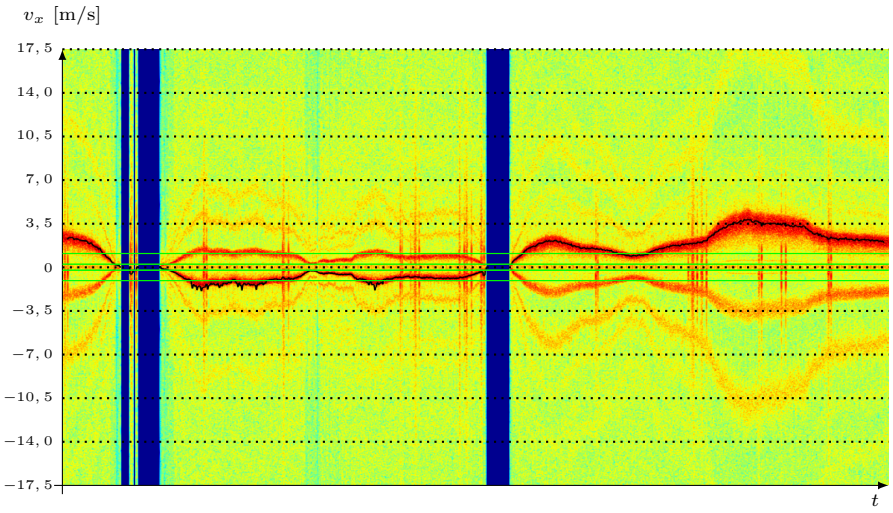
Neurónových sietí na odhad absolútnej hodnoty rýchlosti, ktorý slúži ako pomocný nástroj pre prispôsobenie sa rozmedzia PSD, z ktorého je počítaný prvý spektrálny moment, situácii na podspektrograme.

Modusového filtru na filtráciu výsledkov a medzivýsledkov.

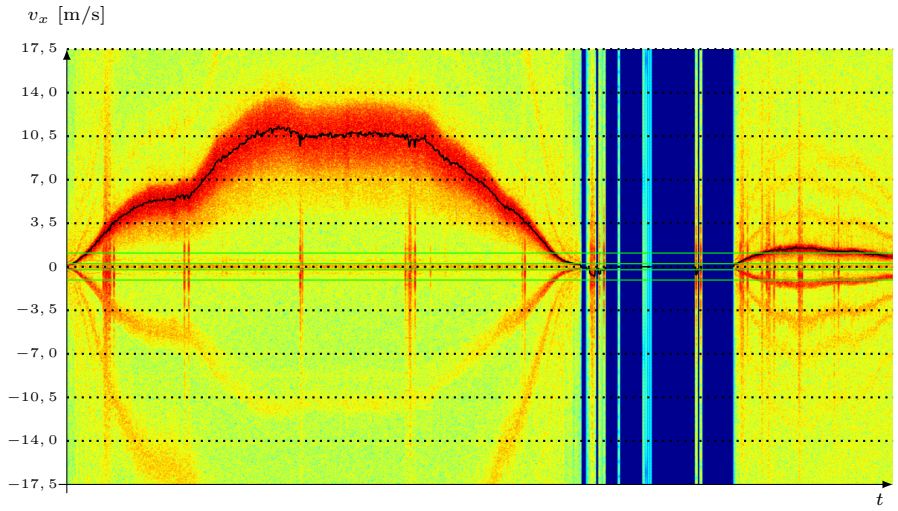
[xxvi] Viď časť 1.3. Parametrami môžu byť napríklad súčty PSD v rôznych oblastiach alebo charakter predchádzajúcich podspektrogramov.

3.6 Vyskúšanie súhrnného vyhodnotenia v MATLABe

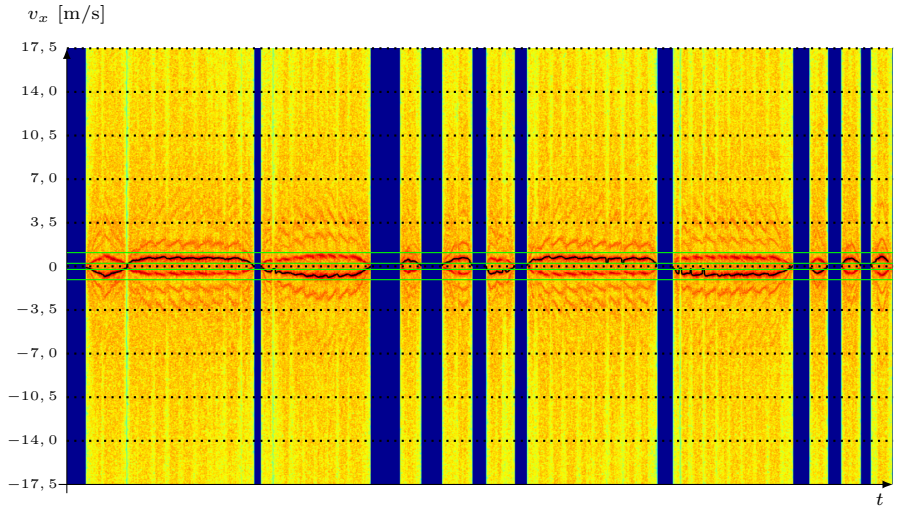
Na vyskúšanie spolupráce použitých postupov, zatiaľ však v *postprocessingovom* režime, sme implementovali skript, uvedený ako E.20. Tento funguje presne tak, ako je znázornené na obrázku 3.35 – zo vstupnej nahrávky vo WAVE formáte je vypočítaný spektrogram, jednotlivé podspektrogramy sú ohodnotené rozlišovačom smeru pohybu (3.4.1) a sú natrénovanými sieťami zadelené do troch skupín podľa absolútnej hodnoty rýchlosti. Pre každý z nich sú tiež vypočítané hodnoty „všetkých troch“ prvých spektrálnych momentov. Na základe týchto parametrov, ktorých výsledky prechádzajú modusovou kĺzavou filtráciou odpovedajúcej dĺžky, je nakoniec vybratá pre každý z podspektrogramov *jediná* hodnota, ktorá má odpovedať rýchlosti v_x na ňom. Výsledky sú naškálované tak, aby ich bolo možné zobraziť na pozadí, tvorenom vykresleným spektrogramom a sú na ňom zobrazené spolu s hranicami rozmedzí rýchlosti pre lokálnu adaptabilitu. Výsledky pre dve z nahrávok z motorového vozidla a dve z nahrávok z ručného vozíku je možné vidieť na nasledujúcich obrázkoch.



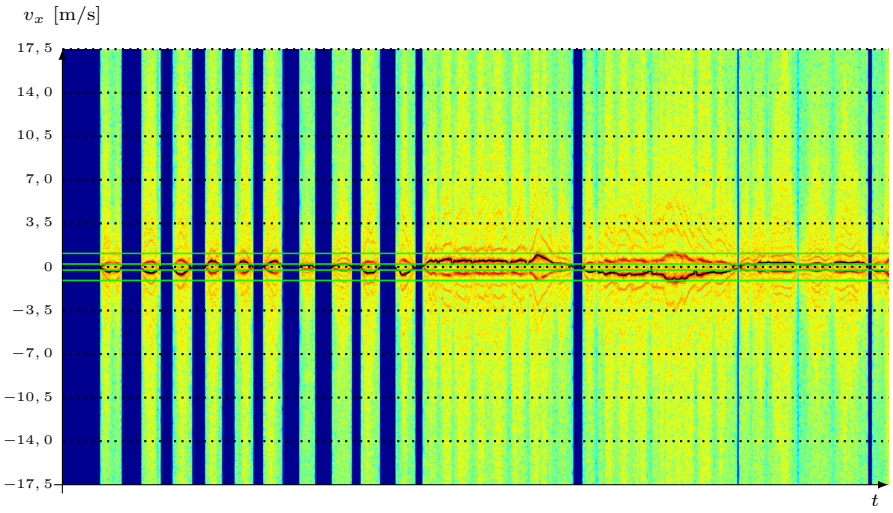
Obr. 3.36: Vyhodnotenie nahrávky $\mathcal{N}_5$ pomocou kódu E.20. Je vidieť, že problémom je hlavne voľba rozmedzí pre lokálnu adaptabilitu určenia absolútnej hodnoty rýchlosti. Pokiaľ je rýchlosť v spektrograme na hranici dvoch takýchto rozmedzí, náš program nepracuje uspokojivo. Implementácia iného rozdelenia by však iba kopírovala prístup doterajší a je do veľkej miery záležitosťou celkového rozsahu rýchlostí, ktoré je našim cieľom skúmať a požiadavkami na presnosť v jednotlivých rozmedziach. Druhým problémom sú záchvevy, ktoré mimo oblasť kontinuálneho pohybu spôsobujú utopenie užitočných hodnôt v šume. Záchvevy sú však problémom veľmi špecifickým, týkajúcim sa iba niektorých meraní na motorovom vozidle. Veríme, že správnym riešením tohoto problému nie je implementácia softvérového prostriedku (Implementované bolo ich rozoznávanie pomocou neurónovej siete, no do programu sme sa ho rozhodli nezahrnúť.) na ich rozoznávanie a potlačenie, ale ich potlačenie *mechanické* či hardvérové.



Obr. 3.37: Vyhodnotenie nahrávky $\mathcal{N}_2$.



Obr. 3.38: Vyhodnotenie nahrávky $\mathcal{N}_{m1}$. Nahrávka vznikla v usporiadaní aparatúry podľa obrázku 2.12 a zachytáva nízke rýchlosti. Vozík je pritom do veľkej miery ovplyvnený terénnymi nerovnosťami v pravidelných intervaloch, čo spôsobuje evidentné zvlnenie hodnôt rýchlosti (dopplerovskej frekvencie) na ňom.



Obr. 3.39: Vyhodnotenie nahrávky $\mathcal{N}_{m3}$. Nahrávka je z usporiadania z obrázku 2.12.

3.7 Real-time prístup

I keď výsledky doterajšieho programu sú dobré, cieľom práce je vytvoriť softvér, ktorý umožní výpočet rýchlosti v reálnom čase. Už vzhľadom na to, že spektrum počítame z niekoľkých po sebe nasledujúcich vzoriek, nedokážeme reálnočasový prístup nikdy skutočne splniť. Cieľom teda bude to, aby program dával výsledky s takým oneskorením, ktoré nie je pre užívateľa obmedzujúce. Pri jednotlivých častiach prechodu vzorku vyhodnocovacím systémom budeme preto vždy myslieť na to, aké oneskorenie pri tom-ktorom úkone vzniká a v prípade neakceptovateľných hodnôt hľadať mechanizmy *náhradné*. Zamerajme sa postupne na jednotlivé časti výpočtu:

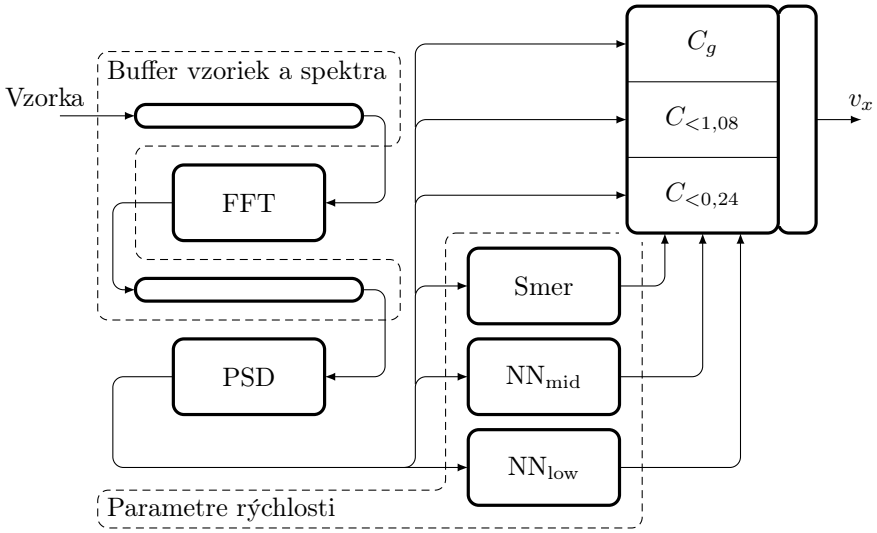
Výpočet PSD. Pri výpočte spektrálnej výkonovej hustoty vzniká oneskorenie, dané parametrami spektrogramu – v našom prípade počítame spektrum každých $N_{\mathrm{FFT}} - N_{\mathrm{noverlap}} = 8192 - 7168 = 1024$ vzoriek. Pri vzorkovacej frekvencii 44100 Hz to odpovedá $\frac{1024}{44100} \approx 0,0232$ s. Jeho zmena je možná zmenou dĺžky okna FFT alebo záporného prekryvu – obe tieto možnosti však ovplyvnia i rozlíšenie spektrogramu. Pretože jeho parametre chceme zachovať, budeme sa s týmto oneskorením musieť zmieriť.

Výpočet parametrov rýchlosti. Parametre rýchlosti sú počítané vždy z jediného vektoru spektrálnej výkonovej hustoty. Tento je sám o sebe oproti reálnemu času oneskorený o približne $0,0232$ s (viď predchádzajúci bod) a v prípade dokonale rýchleho výpočtu žiadne ďalšie oneskorenie nevzniká. Toto však vznikne v prípade, že výsledky týchto parametrov filtrujeme. V závislosti od dĺžky okna filtru w potom vznikne oneskorenie, ktoré trvá $w\frac{N_{\mathrm{FFT}}}{N_{\mathrm{noverlap}}} = 0,0232\,w$ sekúnd. Keďže najdlhšie okno filtru, ktorý doteraz využívame, má dĺžku 35 bodov a filtrácie jednotlivých parametrov môžu bežať paralelne, dostávame celkové oneskorenie

$0,0232 + 0,0232\,w = 0,0232 \cdot 36 = 0,836$ s. Takéto oneskorenie je už iste *postrehnuteľné* a je cenou za lepšie vyhladenie výsledkov a potlačenie skokov medzi hodnotami indikátoru smeru, ktoré sú pri väčšine aplikácií závažnejšie než nie celkom presne určená hodnota absolútnej hodnoty rýchlosti.

Zlúčenie výsledkov je opäť z hľadiska oneskorenia závislé výhradne na možnostiach našej výpočtovej techniky.

Samotný priestup toku vzoriek programom popíšeme pomocou obrázku 3.40.



Obr. 3.40: Schéma prechodu vzoriek vyhodnocovacím systémom v kvázireálnom čase.

Po príchode vzorky do systému nech je táto uložená do istého priebežného poľa, uchovávajúceho 8192 vzoriek. Odkedy sa toto pole zaplní postupujeme takto:

1. Po príchode každej 1024. vzorky necháme z bufferu vzoriek do buffferu spektra vypočítať obraz v DFT.

2. Toto spektrum prevedieme na PSD, z ktorého vypočítame potrebné parametre rýchlosti – smer pohybu a rozmedzie abs. hodnoty rýchlosti. Vypočítame z nej tiež prvé spektrálne momenty našich rozmedzí.

3. Nakoniec je výsledkami zadeľovacích sietí vybratý jeden z troch C a je tiež prenásobený znamienkom, daným hodnotou smeru pohybu. Hodnota výsledku sa uloží do pamäti a je platná ako *aktuálna hodnota* rýchlosti počas ďalších 1023 vzoriek, teda pokým nie je opäť počítaná DFT.

KAPITOLA 4

Implementácia finálneho softvéru

Potom ako sme funkčnosť zvolených vyhodnocovacích postupov vyskúšali, pristúpili sme k implementácii programu, od ktorého požadujeme:

- Schopnosť práce v kváziréalnom čase. Vstupom budú vzorky komplexného signálu, na výstupe chceme dostať rýchlosť v_x, ktorá je v nich uložená a to tak promptne, ako je to len možné.

- Program bude napísaný v štandardnom programovacom jazyku a bude možné ho dostať na čip. V našom prípade volíme jazyk C++ .

Myšlienka ďalšieho postupu je totiž taká, že radar bude tvoriť samostatnú jednotku s vlastným spracovaním bez potreby externej zvukovej karty a počítača. Neformálne vyjadrenú požiadavku na maximálne oneskorenie výstupu oproti vstupu pre použitie ako merač rýchlosti možno vyjadriť tak, že ak by celé zapojenie obsahovalo displej, zobrazujúci aktuálnu rýchlosť, nezaostávala by za rýchlosťou bežného tachometra v automobiloch. V nasledujúcej časti predstavíme implementáciu softvéru pre takéto zariadenie, upraveného tak, aby bolo možné pozorovať jeho výsledky i v súčasnom stave, kedy sú vzorky ukladané do WAVE súborov.

Základná schéma súboru mechanizmov, ktoré je potrebné implementovať, je zobrazená na obrázku 3.40. Logické celky programu ale – hoci táto schéma odpovedá skutočnosti – rozdelíme z dôvodu zlepšenia ich náväznosti trochu rozdielne. Na začiatku sa budeme venovať *simulácii toku vzoriek*, ktorá zabezpečí príchod vzoriek, akoby prichádzali z kaskády radar – A/D prevodník – dekóder. Následne sa budeme venovať určeniu *smeru pohybu* a približne určíme *absolútnu hodnotu* rýchlosti. V ďalšej časti vypočítame *prvý spektrálny moment* a v tej poslednej sa zameriame na filtráciu výsledkov.

4.1 Simulácia toku vzoriek

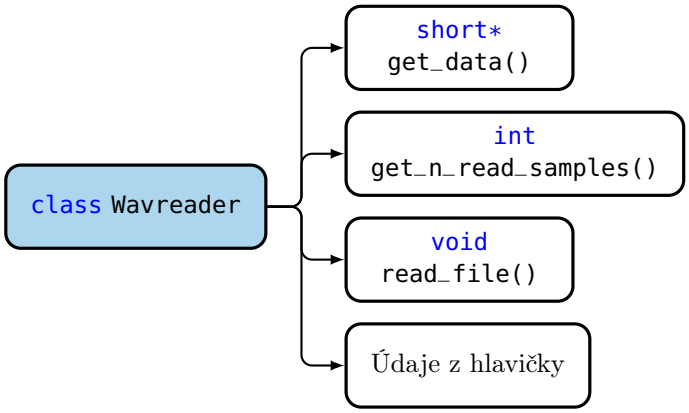
Jedinou časťou programu, ktorá nebude v rámci tejto práce odpovedať realite, je príchod vzoriek doň. Pretože táto práca je zameraná na softvérové spracovanie, nesnažili sme sa o náhradu zvukovej karty A/D prevodníkom a dekóderom, ale o implementáciu simulátoru, ktorý zo vstupu – nahrávky vo WAVE formáte – posiela do spracovania jeden po druhom časovo za sebou nasledujúce komplexné vzorky. Celý program je potom závislý na súslednom príchode týchto vzoriek a v prípade, že ich budeme získavať iným spôsobom, než z nahrávky, upravíme iba túto vstupnú časť.

Takýmto simulátorom je prvá časť programu. Ešte predtým, ako popíšeme samotnú triedu `Wavreader`, ktorej objekty sú na to určené, vysvetlíme funkciu triedy `Constants_datastore`. Táto obsahuje *všetky konštanty*, ktoré budú použité počas behu programu. Ide o konštanty ako napríklad *dĺžka okna FFT*, *počet načítaných vzoriek* či *cesty k súborom s parametrami sietí*. Definícia triedy `Constants_datastore` a jej členských funkcií sa nachádza v kóde E.22.

O samotné načítanie súboru vo formáte WAVE sa postará objekt triedy `Wavreader`. Tento obsahuje

- Všetky časti hlavičky WAVE súboru, popísané v [7],
- Pole typu `short data`, ktoré má obsahovať vzorky zo súboru a ktorého dĺžka je daná atribútom `subchunk2Size` z hlavičky súboru,
- Prístupové metódy `get_data()` a `get_n_read_samples()`, vracajúce ukazovateľ na pole načítaných vzoriek a počet načítaných vzoriek.

Trieda `Wavreader` je znázornená na obrázku 4.1 a jej zdrojový súbor je kódom E.23.



Obr. 4.1: Trieda `Wavreader`, slúžiaca na načítanie vzoriek z WAVE súboru.

Po načítaní vzoriek sú tieto uložené do poľa typu `double` mimo objekt triedy `Wavreader` a na tento objekt môžeme volať deštruktor.

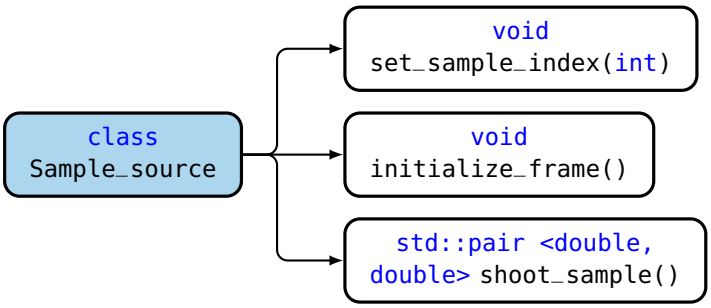
Ďalšou triedou, starajúcou sa o simuláciu toku dát je trieda `Sample_source`, ktorej štruktúra je na obrázku 4.2 a jej funkcie a jej členské funkcie sú:

Konštruktor, ktorý berie za vstup ukazovateľ na pole so vzorkami signálu, ukazovateľ na objekt triedy `Constants_datastore` a ukazovateľ na pole typu `fftw_complex` [4], ktorého dĺžka je nastavená tak, aby odpovedala dĺžke FFT. Toto pole budeme v ďalšom texte nazývať *operačným*.

`std::pair <double, double> shoot_sample()` , čo je funkcia, ktorá vráti v poradí ďalšiu[i] nasledujúcu komplexnú vzorku signálu. Názov funkcie je zvolený s ohľadom na jej analógiu so zariadením, „strieľajúcim“ vzorky.

`void initialize_frame()`, ktorá zaplní operačné pole prvými N_{FFT} vzorkami signálu. Konštanta N_{FFT} je členskou premennou triedy `Constants_datastore` a – ako už bolo mnohokrát spomínané v predošlom texte – v našom prípade ide o číslo 8192. Jeho hodnota nie je priamo zadaná z dôvodu prehľadnosti. Táto funkcia volá príslušný počet krát funkciu `std::pair <double, double> shoot_sample()`.

`void set_sample_index(int)` ako funkcia, nastavujúca index nasledujúcej vzorky.



Obr. 4.2: Trieda `Sample_source`, slúžiaca na simuláciu príchodu vzoriek komplexného signálu.

Poslednou doposiaľ neriešenou činnosťou, ktorú musí zvládnuť simulátor toku vzoriek, je cyklický posun prvkov operačného poľa a pridanie vzorky nasledujúcej na jeho okraj. Smer, ktorým to bude vykonané, nie je podstatou úlohy ovplyvnený a prakticky si ho môžeme zvoliť podľa našich preferencií. V našej implementácii pridávame novú vzorku na *koniec* poľa a teda vzorky z predchádzajúcej iterácie posúvame smerom „vľavo.“ O toto sa stará členská funkcia `void add_new_sample(std::pair <double, double>)` triedy `Frame_operator`. Pretože o tejto triede by bolo nutné tak či onak napísať v riadkoch tesne nasledujúcich, dovoľujeme si jej popis – hoci nie všetky jej funkcie sa týkajú oblasti, danej nadpisom – uviesť už teraz. Jej schéma je na obrázku 4.3. Významnými členskými premennými a funkciami tejto triedy sú:

Konštruktor, ktorý okrem iného alokuje pamäť pre pole typu `double` dĺžky N_{FFT} a zaplní ho prvkami Hammingovho okna tejto dĺžky. Výpočet jeho

[i] To, ktorú vzorku funkcia vráti, je dané členskou premennou `int sample_index`, ktorej hodnota je inkrementovaná s každým volaním funkcie `shoot_sample()`.

prvkov je tu robený podľa vzťahu

$$h_i = 0,54 - 0,46\cos(2\pi\frac{i}{N_{\text{FFT}} - 1}), \tag{4.1}$$

ktorý je prevzatý z dokumentácie MATLABu, odvolávajúcej sa na zdroj [13]. h_i tu značí i-ty prvok tohoto poľa.

Deštruktor, ktorý uvoľní pamäť, určenú poľu, alokovanému konštruktorom.

`void add_new_sample(std::pair <double, double>*)`, pridávajúca komplexnú vzorku, na ktorú odkazuje ukazovateľ zo vstupu, na koniec operačného poľa. Pred pridaním tiež cyklicky posunie všetky prvky operačného poľa o jedno miesto smerom „vľavo“.

`void hamming_the_frame()`, poprvkovo násobiaca operačné pole poľom $\vec{h}$, teda Hammingovym oknom, ktoré je inicializované pri konštrukcii objektu triedy `Frame_operator`.

`void revert_hamming_the_frame()`, ktorá operačné pole, ktoré je pri volaní tejto funkcie vynásobené Hammingovym oknom, týmto oknom opäť delí a vracia ho tak do „normálneho“ stavu, teda stavu, kedy operačné pole obsahuje čisté komplexné vzorky signálu radarového senzoru.

`void psd()`, čo je funkcia, ktorá vypočíta z frekvenčného operačného poľa[ii] spektrálnu výkonovú hustotu a uloží ju opäť ako prvky frekvenčného operačného poľa. Spektrálna výkonová hustota $\vec{p}$ je počítaná štandardne, podľa vzťahu

$$p_i = \frac{1}{2\pi\, N_{\text{FFT}}}|s_i|^2, \tag{4.2}$$

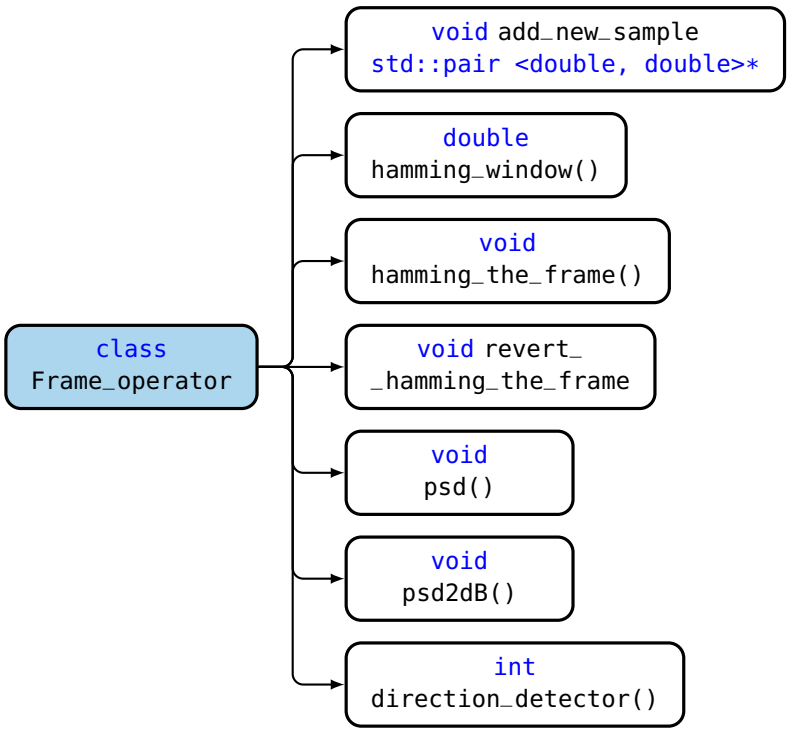
kde $\vec{s}$ je výkonovým spektrom signálu zo signálového operačného poľa. Nakoľko zo vzťahu (4.2) vyplýva, že ide o *reálne* hodnoty, na príslušných pozíciách položí táto funkcia imaginárnu zložku operačného poľa rovnú 0. Ďalším dôležitým znakom tejto funkcie je to, že nepočíta p pre všetkých N_{FFT} hodnôt frekvencie, ale iba pre tie indexy, ktoré sú pre ďalší výpočet „užitočné,“ čo znamená v praxi iba pre prvých `VX_LIMIT_INDEX` prvkov z *každej strany* operačného poľa. Hodnota `VX_LIMIT_INDEX` je členskou premennou triedy `Constants_datastore` a získanie jej hodnoty je popísané v predchádzajúcom texte.

`int direction_detector()`, ktorá vypočíta súčty spektrálnej výkonovej hustoty v „ľavej“ a „pravej“ polovici preusporiadaného operačného poľa `spectrum` a vráti znamienko rozdielu týchto súčtov.

4.1.1 Príklad podprogramu

Pre názornosť simulácie uvedieme príklad volania podprogramu, načítavajúceho vzorky signálu zo súboru `file.wav`, simulujúceho ich tok a ich prechod systémom, ktorý počíta spektrogram s parametrami

[ii] V skutočnosti ide samozrejme o dve rôzne operačné polia. Jedno z nich je to *signálové*, ktoré obsahuje N_{FFT} komplexných signálových vzoriek. To druhé obsahuje taktiež N_{FFT} komplexných hodnôt, no tentorkrát ide o Fourierov obraz signálového poľa. To, o ktorom operačnom poli je reč, by malo byť z kontextu zrejmé; Budeme sa však snažiť podporiť túto zrejmosť prívlastkami *signálové* alebo *frekvenčné*.



Obr. 4.3: Trieda `Frame_operator`, slúžiaca na prácu s operačným poľom.

- Šírka okna 8192 bodov
- Negative overlap 1024 bodov
- Hammingovo okno.

Vytvoríme objekt triedy `Wavreader` `wav` a necháme ho prečítať dáta z WAVE súboru.

```
1     Constants_datastore constants; // In constants object there is stored the ath to WAVE
          file file.wav.
2     Wavreader wav;
3
4     wav.read_file();
```

Následne alokujeme pamäť pre pole `data` typu `double` a zaplníme ho vzorkami, danými členskou premennou `short* wav.data`. V tomto momente už prestáva byť objekt `wav` potrebný a môžeme volať deštruktor.

```
5  double* data = (double*)malloc(2 * constants.get_N_READ_SAMPLES() * sizeof(double));
6  for (int i = 0; i < 2 * constants.get_N_READ_SAMPLES(); i++) {
7      data[i] = double(wav.get_data()[2 * constants.get_BEGINNING() + i] / constants.
           get_SHORTSIZE());
8  }
9
10     wav.~Wavreader(); // Destructing wavreader
```

Vytvoríme si polia typu `fftw_complex`, predstavujúce operačné polia. Dátový typ je takto zvolený kvôli tomu, že s týmito poľami bude neskôr pracovať knižnica *FFTW*, pre ktorú je tento typ priamo určený a ktorej je jeho definícia súčasťou [4]. Vytvoríme tiež *plán* FFT, čo objekt, obsahujúci parametre FFT.

```
11 fftw_complex* frame; // Input to complex, one-dimensional DFT.
```

```
12  fftw_complex* spectrum; // Output from complex, one-dimensional DFT.
13  frame = (fftw_complex*)fftw_malloc(constants.get_N_FFT() * sizeof(fftw_complex)); // 8192
        ongoing samples as an input for FFT computation
14  spectrum = (fftw_complex*)fftw_malloc(constants.get_N_FFT() * sizeof(fftw_complex)); //
        8192 frame's spectrum values
15
16  fftw_plan plan = fftw_plan_dft_1d(constants.get_N_FFT(), frame, spectrum, FFTW_FORWARD,
        FFTW_MEASURE);
```

Vytvoríme objekty tried `Sample_source` a `Frame_operator` (Ich zdrojové súbory sú uvedené v kódoch E.24 a E.25). Inicializujeme signálové operačné pole prvými 8191 vzorkami signálu a deklarujeme tiež `std::pair <double, double> actual_sample`, do ktorého bude program ukladať ďalšiu načítanú vzorku. Kvôli jednoduchosti sme využili definíciu aliasu typu `std::pair <double, double>`, ktorú sme nazvali `ddpair_t`.

```
17  Sample_source sample_shooter(data, &constants, frame);
18  Frame_operator frame_handler(frame, spectrum, &constants);
19
20  sample_shooter.initialize_frame(); // Filling the frame with first 8191 samples
21  ddpair_t actual_sample(0.0,0.0); // Typedef of ddpair_t is given in types.h
```

Simulujeme tok vzoriek a počítame spektrogram. V cykle cez všetky načítané vzorky vždy „vystrelíme“ nasledujúcu, pridáme ju do operačného poľa, operačné pole prenásobíme Hammingovym oknom a toto hneď pošleme na vstup FFT. Po jej výpočte máme výsledok uložený vo frekvenčnom operačnom poli `spectrum`. Nakoniec musíme signálový operačný rámec opäť poprvkovo vydeliť Hammingovym oknom – potom je signálové pole pripravené na príjem ďalšej vzorky.

```
22  for (int i = 0; i < (constants.get_N_READ_SAMPLES() - (constants.get_N_FFT())); i++) { //
        For number of samples loaded from .wav file minus number of samples, which were feed
         to frame by initialize_frame() function.
23      actual_sample = sample_shooter.shoot_sample(); // Sample comes
24      frame_handler.add_new_sample(&actual_sample); // Add it to frame
25
26      if (i % (constants.get_N_FFT() - constants.get_NOVERLAP()) == 0) { // every 1024th
            sample
27          frame_handler.hamming_the_frame(); // Multiplying the frame by Hamming window.
28          fftw_execute(plan); // Proceed FFT computation
29
30          /* Now we have result saved in spectrum array */
31
32          frame_handler.revert_hamming_the_frame(); // Dividing the frame by Hamming window
33      }
34
35  }
```

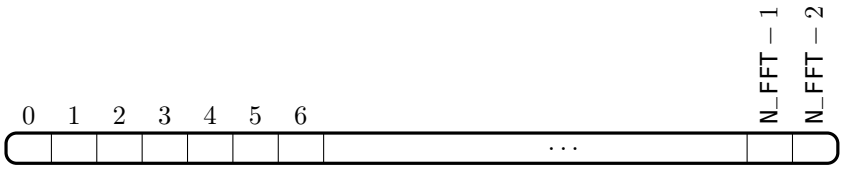
Na záver nesmieme zabudnúť uvoľniť pamäť:

```
36  fftw_destroy_plan(plan);
37  fftw_free(frame);
38  fftw_free(spectrum);
39  free(data);
```

4.2 Určenie smeru pohybu

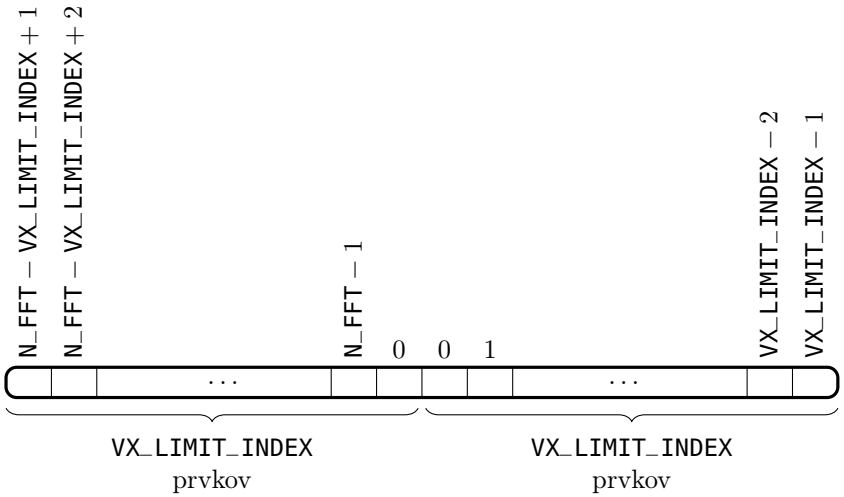
Pre určenie smeru pohybu sme – ako už bolo spomínané v časti 3.5 – zvolili ako najspoľahlivejšie vyhodnotenie pomocou súčtu spektrálnej výkonovej hustoty medzi dvoma stranami *preusporiadaného* vektoru spektrálnej výkonovej hustoty. Hoci toto preusporiadanie sme používali už mnohokrát počas testovania algoritmov v Matlabe, považujeme za vhodné ho uviesť na pravú mieru pomocou indexovania, vlastného jazyku C++ a s použitím konštánt, ktoré náš

program používa a ktoré sú súčasťou inštancií triedy `Constants_datastore`. Majme najskôr vypočítanú spektrálnu výkonovú hustotu, ktorá je uložená v operačnom poli `spectrum` na jeho pozíciách, odpovedajúcich reálnej časti:



Obr. 4.4: Základné indexovanie frekvenčného operačného poľa. Každý z jeho prvkov predstavuje komplexné číslo. Pri reálnej veličine – spektrálnej výkonovej hustote – však používame iba jeho reálnu časť.

V ďalšom spracovaní budeme *poľom* $\vec{p}$ nazývať iba niektoré prvky operačného poľa – to, o ktoré ide, je zrejmé z obrázku 4.5.



Obr. 4.5: Znázornenie indexov frekvenčného operačného poľa, ktoré sú podstatné pre výpočet rýchlosti v_x a ich usporiadanie do mysleného poľa, s ktorým pracuje funkcia `void Frame_operator::psd()`

.

Znamienko rozdielu súčtov spektrálnej výkonovej hustoty $\vec{p}$ medzi dvoma polovicami operačného poľa zistíme najpriamočiarejším možným spôsobom – implementovaný je ako funkcia `int direction_detector()` triedy `Frame_operator`.

4.2.1 Príklad podprogramu

Rovnako ako v predchádzajúcej časti i tu uvedieme krátky príklad programu, počítajúceho smer pohybu (indikácia $+1$ znamená *vpred*, -1 zase *vzad*), resp. detekujúceho neprítomnosť pohybu (indikácia 0). Začiatok funkcie `main()` ne-

chajme rovnaký, ako v prípade predchádzajúcom a pozrime sa iba na to, akým spôsobom možno získať spomínanú informáciu o smere pohybu:

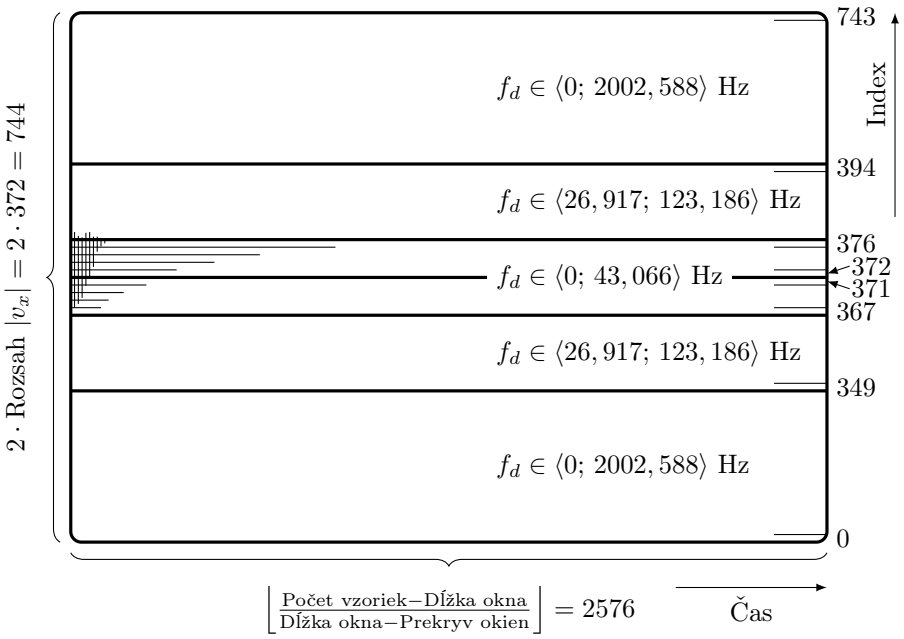
```
21  ...
22  for (int i = 0; i < (constants.get_N_READ_SAMPLES() - (constants.get_N_FFT())); i++) { //
        For number of samples loaded from .wav file minus number of samples, which were feed
        to frame by initialize_frame() function.
23      actual_sample = sample_shooter.shoot_sample(); // Sample comes
24      frame_handler.add_new_sample(&actual_sample); // Add it to frame
25
26      if (i % (constants.get_N_FFT() - constants.get_NOVERLAP()) == 0) { // every 1024th
            sample
27          frame_handler.hamming_the_frame(); // Multiplying the frame by Hamming window.
28          fftw_execute(plan); // Proceed FFT computation
29          frame_handler.psd(); // Transfer it to normalized PSD
30
31          /* Now we have PSD saved in spectrum array */
32
33          /* If the "int direction" variable is initialized, we can assign the direction
                sign to it using: */
34          dir = frame_handler.direction_detector();
35
36          frame_handler.revert_hamming_the_frame(); // Dividing the frame by Hamming window
37      }
38
39  }
40  ...
```

4.3 Zaradenie absolútnej hodnoty rýchlosti

Spôsob zaradenia absolútnej hodnoty rýchlosti do troch skupín bol problémom, ktorý nemusel byť diskutovaný hlbšie z hľadiska voľby nástroja – neurónová sieť sa javila suverénne ako najspoľahlivejšia, pričom jej nespornou výhodou je jednoduchosť implementácie prechodu vstupného vektoru ňou. Z množstva použitých neurónov a voľby prechodových funkcií potom vyplýva celková výpočtová náročnosť. Záležitosť, ktorou sa bolo potrebné väčšmi zaoberať, bola v tomto prípade voľba hraníc úsekov, ktoré postačia na dostatočne jemné rozdelenie celého rýchlostného rozsahu vstupného vektoru $\vec{p}$, tak, aby výsledok prvého spektrálneho momentu z týchto úsekov na seba v bodoch „prepnutia“ hladko naväzoval. Po prehliadnutí si veľkého množstva spektrogramov, vykreslených pomocou jednej z trojrozmerných[iii] vykresľovacích funkcií Matlabu, sme zvolili hraničné hodnoty pre *nízku*, *strednú* a *vysokú* rýchlosť tak, ako je uvedené na obrázku 4.6. Pretože pri pôvodnej myšlienke, rozdeliť celý rozsah na tri *neprekrývajúce sa* intervaly bola zmena pri zmene podintervalu vždy skoková, pridali sme istý nezanedbateľný prekryv. Výsledkom sú pre tri rozsahy rýchlostí tri podintervaly, z ktorých počítame prvý spektrálny moment. Neurónová sieť sa potom stará o zaradenie aktuálneho „spektra“ do týchto podintervalov. Ich rozloženie sme zhrnuli do tabuľky 4.1

Po vyriešení týchto záležitostí sme mohli pristúpiť k tréningu dvoch sietí s pomocou NNT. Každá z nich indikuje svojim výstupom príslušnosť vstupného poľa k jednému z dvoch podintervalov. Obe siete sú typu *pattern-recognition*, teda ich výstupom je v ideálnom prípade dokonalého tréningu postupnosť núl a jednotiek, zhodujúca sa s postupnosťou núl a jednotiek, tvoriacou jeden z vektorov trénovacej množiny cieľov, $\odot$.

[iii] Tretím rozmerom tu rýchlosti a čase bola vo všetkých prípadoch *farba*, predstavujúca spektrálnu výkonovú hustotu v decibeloch.



Obr. 4.6: Indexovanie a rozsahy pôsobnosti podintervalov celého skúmaného intervalu rýchlostí.

Trieda `Neural_network`, reprezentujúca natrénovanú neurónovú sieť „nášho typu“, je znázornená na obrázku 4.7

Jej zdrojový kód je uvedený ako kód E.26 a skladá sa z týchto významných premenných a funkcií:

`double* input` je pole, do ktorého sa skopíruje potrebná časť obsahu „reálnej časti“ frekvenčného operačného poľa, v ktorom je v momente kopírovania uložená spektrálna výkonová hustota vzoriek komplexného signálu v odpovedajúcom signálovom operačnom poli. Výber indexov vo frekvenčnom operačnom poli, ktoré budú kopírované do poľa `input` je vytvorený ako pole `int reshape_ind[constants->get_VX_LIMIT_INDEX()]`, teda pole indexov, ktorým odpovedajúce prvky vo frekvenčnom operačnom poli majú byť namapované na indexy 0 až 743 podľa obrázku 4.6.

`std::pair <double, double> output` je pár, do ktorého sú ukladané výstupy neurónovej siete. Obe siete, ktoré pre vyhodnocovanie používame, majú na výstupe usporiadanú dvojicu reálnych čísel – `std::pair` sa preto na tento účel hodí.

Matice $\mathbb{W}$, vektory $\vec{b}$ a riadkové extrémy. Ide o premenné typu `std::vector <double>`. Tieto obsahujú váhy a kvocienty jednotlivých vrstiev sietí, natrénovaných v prostredí NNT a maximá a minimá riadkov matice, použitej na tréning. V implementácii triedy `Neural_network` ich nazývame `w_input`, `w_hidden`, `b_input`, `b_hidden`, `input_row_min` a `input_row_max`.

Rozsah frekvencií, vyhodnocovaných sieťou [Hz]	**Rozsah frekvencií, z ktorých je počítaný** C [Hz]
0 až 26,917	0 až 43,066
26,917 až 123,186	26,917 až 123,186
123,186 až 2002,588	0 až 2002,588

Rozsah rýchlostí, vyhodnocovaných sieťou [m/s]	**Rozsah rýchlostí, z ktorých je počítaný** C [m/s]
0 až 0,2367	0 až 0,3787
0,2367 až 1,0832	0,2367 až 1,0832
1,0832 až 17,6088	0 až 17,6088

Rozsah indexov, vyhodnocovaných sieťou	**Rozsah indexov, z ktorých je počítaný** C
0 až 4	0 až 7
4 až 22	4 až 22
22 až 371	0 až 371

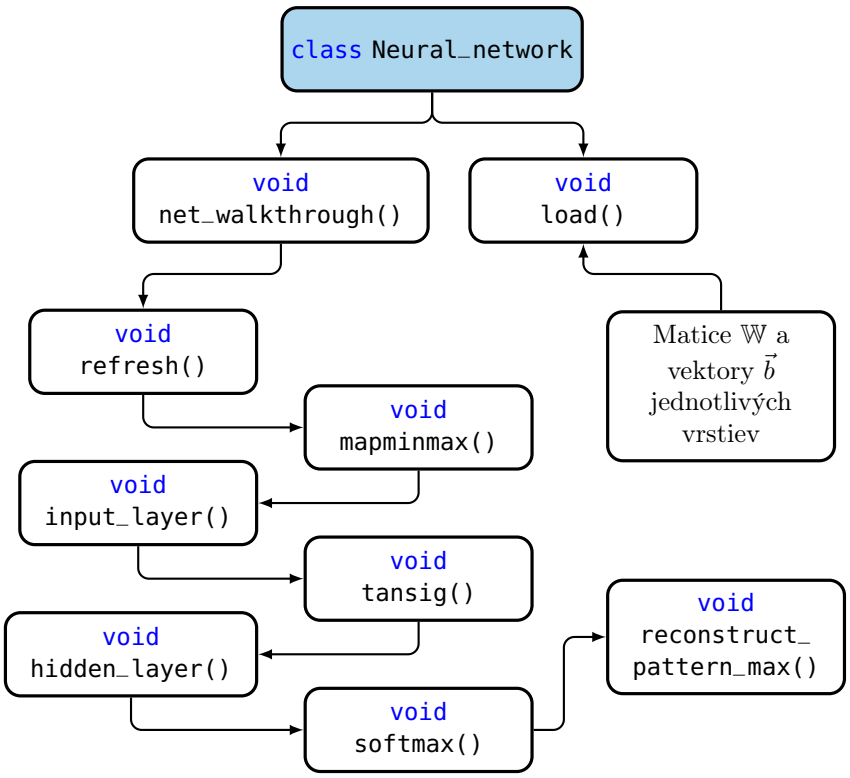
Tab. 4.1: Zhrnutie frekvenčných, rýchlostných a indexových intervalov, do ktorých siete zadeľujú svoje vstupné vektory („spektrá“). Intervaly sú písané iba pre kladný smer; V zápornom smere platia samozrejme analogicky. Indexami sú myslené indexy frekvenčných bodov vo frekvenčnom operačnom poli `spectrum`.

Konštruktor inicializuje vektor vstupu `input`, pár výstupu `output`, a alokuje a naplní mapu indexov `reshape_ind`.

`void load()` je funkcia, ktorá zo súborov, ktoré sme získali uložením váh a kvocientov natrénovaných sietí v MATLABe, načítava dáta do operačnej pamäti. Okrem váh a kvocientov sú načítané tiež maximá a minimá riadkov matice, ktorá má v stĺpcoch trénovacie dáta – tieto sú potom používané na normovanie prvkov poľa `input`. To, ktoré z dát načítať, je indikované vymenovacím typom `enum which_nn`, ktorý je jedným z parametrov konštruktoru. Pre sieť, rozhodujúcu o priradení absolútnej hodnoty rýchlosti vstupu medzi 0 a 0,2367 m/s je zvolený alias `abs_low` a pre tú, čo rozhoduje o priradení absolútnej hodnoty rýchlosti vstupu medzi 0,2367 a 1,0832 m/s zas `abs_mid`.

`void net_walkthrough()`, ktorá predstavuje celý prechod vstupného poľa sieťou. Táto funkcia iba volá funkcie, starajúce sa o jednotlivé podúlohy. Konkrétne ide o:

`void refresh()`, predstavujúca nástroj na kopírovanie reálnej časti prvkov frekvenčného operačného poľa do poľa `input`.



Obr. 4.7: Významné funkcie a premenné triedy `Neural_network`.

`void mapminmax()` následne normalizuje hodnoty `input`u do intervalu $\langle -1; 1\rangle$ na základe extrémov riadkov matice trénovacích dát, uložených v programe ako `input_row_min` a `input_row_max`. Použitý je vzťah

$$y = 2\,\frac{x - x_m}{x_M - x_m}\,, \tag{4.3}$$

kopírujúci implementáciu funkcie `mapminmax` v MATLABe, kde x a y sú vstupná a výstupná hodnota prvku na i-tej pozícii, x_m minimum i-teho riadku trénovacej matice a x_M maximum toho istého riadku.

`void input_layer()` jednoducho násobí maticu váh vstupnej vrstvy `w_input` sprava poľom `input`. Na záver k výsledku pripočíta kvocienty vstupnej vrstvy `b_input`.

`void tansig()` je prechodovou funkciou vstupnej vrstvy – jej definícia je popísaná v časti 3.1.

`void hidden_layer()` analogicky s `input_layer()` predstavuje aritmetiku v skrytej vrstve.

`void softmax()` je prechodovou funkciou skrytej vrstvy a jej definícia je uvedená v časti 3.1.

`void reconstruct_pattern_max()` vyhľadáva maximum spomedzi dvojice prvkov výstupného páru `output`, na jeho pozíciu uloží

jednotku a na tú druhú pozíciu nulu. Výstupné *patterny*[iv] totiž očakávame v tvare $[0, 1]$ alebo $[1, 0]$. Jednotkou na prvej pozícii (output.first) indikujeme *príslušnosť* vstupu k cieľovej skupine (ňou je rozmedzie rýchlostí) a jednotkou na pozícii druhej zas jeho *neprísluš*nosť k nej. O príslušnosti vstupu k danej cieľovej skupine teda môžeme v ideálnom prípade rozhodnúť už na základe hodnoty prvku `output.first`. Inou možnosťou rekonštrukcie výstupu je porovnávanie oboch prvkov s hodnotou $x_{\text{threshold}} = 0,5$ a priradenie hodnoty 1 na pozíciu, kde je pôvodná hodnota vyššia než $x_{\text{threshold}}$. Tento prístup je implementovaný ako funkcia `void Neural_network::reconstruct_pattern_half()`.

`std::pair <double, double> get_result()` vracia hodnotu páru `result`.

4.3.1 Príklad podprogramu

Po predstavení triedy i tu predveďme jednoduchý príklad jej použitia. Začneme rovnako ako pri vyhodnocovaní smeru v časti 4.2.1, no s tým rozdielom, že tentokrát ešte predtým vytvoríme objekt triedy `Neural_network net_abs_low`, ktorý bude svojim výstupom indikovať príslušnosť vstupu do rozmedzia rýchlostí 0 až 0,2367 m/s. Zavoláme tiež funkciu `void Neural_network::load()`, ktorá zariadi načítanie potrebných dát, týkajúcich sa natrénovanej siete, do pamäti.

```
18  ...
19  Neural_network nn_abs_low(&constants, abs_low, spectrum);
20  nn_abs_low.load();
21  ...
22  for (int i = 0; i < (constants.get_N_READ_SAMPLES() - (constants.get_N_FFT())); i++) { //
        For number of samples loaded from .wav file minus number of samples, which were feed
        to frame by initialize_frame() function.
23      actual_sample = sample_shooter.shoot_sample(); // Sample comes
24      frame_handler.add_new_sample(&actual_sample); // Add it to frame
25
26      if (i % (constants.get_N_FFT() - constants.get_NOVERLAP()) == 0) { // every 1024th
            sample
27          frame_handler.hamming_the_frame(); // Multiplying the frame by Hamming window.
28          fftw_execute(plan); // Proceed FFT computation
29          frame_handler.psd(); // Transfer it to normalized PSD
30          nn_abs_low.net_walkthrough(); // Put it into network input
31
32          /* If the "net_abs_low_result" variable is initialized, we can assign the network
                result to it using: */
33          net_abs_low_result = nn_abs_low.get_result().first;
34
35          frame_handler.revert_hamming_the_frame(); // Dividing the frame by Hamming window
36      }
37
38  }
39  ...
```

4.4 Výpočet prvého spektrálneho momentu a určenie rýchlosti

Pri experimentoch s rôznymi možnosťami vyhodnotenia spektrálnej výkonovej hustoty v MATLABe sme sa mohli presvedčiť, že najlepšie výsledky spomedzi skúšaných dosiahlo vyhodnotenie pomocou výpočtu prvého spektrálneho

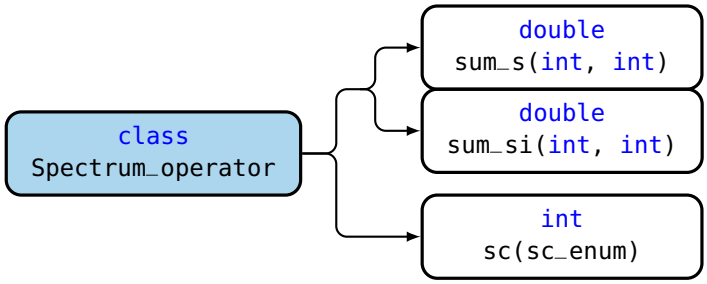
[iv] Ide o termín, používaný pre vektory tvaru $(v_1, v_2, \ldots, v_n)$; $v_n \in \{0; 1\}$.

momentu C. Pripomeňme vzťah pre výpočet prvého spektrálneho momentu spektrálnej výkonovej hustoty $\vec{p}$:

$$C = \frac{\sum_{i=0}^{N-1} \vec{p}_i \cdot i}{\sum_{i=0}^{N-1} \vec{p}_i} \tag{4.4}$$

Výsledkom je, ako je zo vzťahu (4.4) zrejmé, *index* prvého spektrálneho momentu. Hoci by bolo možné mapovať na hodnotu indexu hodnotu rýchlosti v_x už pred započatím výpočtu a dostať tak na výstupe *rýchlosť*, odpovedajúcu prvému spektrálnemu momentu, nebudeme to tak robiť a mapovanie vykonáme až pre výsledný index.

Implementácia využíva triedu `Spectrum_operator`, ktorú si, ako je v tejto časti zvykom, priblížime. Jej hlavné časti sú zhrnuté na obrázku 4.8 a ich detailnejší popis v nasledujúcich riadkoch.



Obr. 4.8: Významné funkcie a premenné triedy `Spectrum_operator`.

`double sum_s(int, int)` počíta a vracia súčet frekvenčného operačného poľa od indexu, zadaného prvým vstupom do indexu, zadaného vstupom druhým, oba vrátane.

`double sum_si(int, int)` robí veľmi podobný výpočet, ako predchádzajúca funkcia, ale každý z prvkov frekvenčného poľa, vstupujúceho do súčtu, váhuje jeho indexom v operačnom poli.

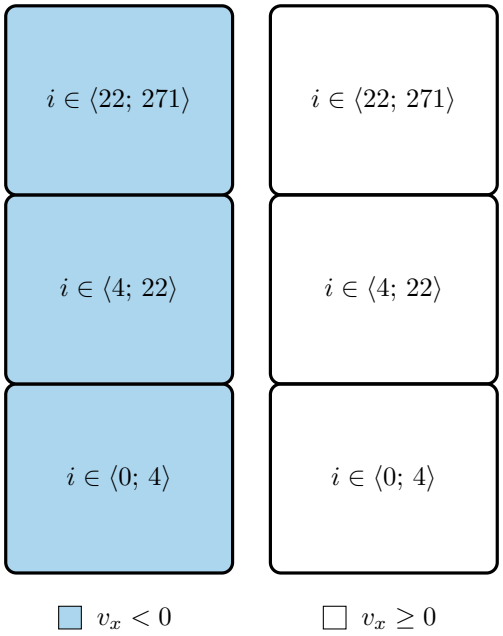
`enum sc_enum` nadobúda hodnoty `low`, `mid` alebo `all` a slúži na prehľadnejšiu indikáciu obmedzenia indexov, s ktorými pracujeme. Typ je zadefinovaný v hlavičkovom súbore `types.h` (kód E.28).

`int sc(sc_enum)` počíta a vracia prvý spektrálny moment (jemu odpovedajúci index) medzi indexami operačného poľa, ktoré sú vlastné každej z hodnôt `sc_enum`. Výpočet presne kopíruje vzťah (4.4) a návratová hodnota indexu je počítaná v rámci *celého* operačného poľa, nie v rámci obmedzujúcich indexov, daných tabuľkou 4.1.

4.4.1 Príklad podprogramu

S výpočtom prvého spektrálneho momentu ide ruka v ruke i identifikácia „stavu,“ v ktorom sa daná rýchlosť má nachádzať. 6 možných stavov je pripomenutých na obrázku 4.9. Pretože vektor $\vec{p}$, resp. frekvenčné operačné pole

môže nadobúdať *práve jeden* z týchto stavov, pričom znamienkom nie je výpočet absolútnej hodnoty rýchlosti ovplyvnený[v], počítanie prvého spektrálneho momentu pre *každý* z troch stavov, opísaných na obrázku 4.9 by bolo neefektívne. 2 výpočty C z troch by boli totiž pre ďalší postup redundantné[vi]. Preto funkcia `int sc(sc_enum)` počíta so vstupom, ktorý určí rozmedzie absolútnych hodnôt rýchlosti.



Obr. 4.9: Stavy, ktoré môže nadobúdať vektor spektrálnej výkonovej hustoty $\vec{p}$ po vyhodnotení predchádzajúcimi mechanizmami. i značí index vo frekvenčnom operačnom poli.

Program začnime rovnako ako v časti 4.3.

```
13  ...
14      Sample_source sample_shooter(data, &constants, frame);
15      Frame_operator frame_handler(frame, spectrum, &constants);
16      Spectrum_operator spectrum_handler(spectrum, &constants);
17      Neural_network nn_abs_low(&constants, abs_low, spectrum);
18      Neural_network nn_abs_mid(&constants, abs_mid, spectrum);
19      nn_abs_low.load();
20      nn_abs_mid.load();
21  ...
22  for (int i = 0; i < (constants.get_N_READ_SAMPLES() - (constants.get_N_FFT())); i++) { //
        For number of samples loaded from .wav file minus number of samples, which were feed
        to frame by initialize_frame() function.
23      actual_sample = sample_shooter.shoot_sample(); // Sample comes
24      frame_handler.add_new_sample(&actual_sample); // Add it to frame
25
26      if (i % (constants.get_N_FFT() - constants.get_NOVERLAP()) == 0) { // every 1024th
            sample
27          frame_handler.hamming_the_frame(); // Multiplying the frame by Hamming window.
28          fftw_execute(plan); // Proceed FFT computation
```

[v] Môžeme sa teda obmedziť na *riadky* z obrázku 4.9.

[vi] V časti 4.5 je písané okrem iného o tom, prečo sa v praxi bez výpočtu všetkých troch nezaobídeme.

```
29          frame_handler.psd(); // Transfer it to normalized PSD
30          dir = frame_handler.direction_detector();
31
32          nn_abs_low.net_walkthrough();
33          nn_abs_mid.net_walkthrough();
34
35          /* Considering net_abs_low_result and net_abs_mid_result initialized */
36          net_abs_low_result = nn_abs_low.get_result();
37          net_abs_mid_result = nn_abs_mid.get_result();
38
39          if (net_abs_low_result.second && net_abs_mid_result.second) { // If velocity is
                higher than mid level
40              cout << (dir * spectrum_handler.sc(all)) << endl;
41          }
42          else if (net_abs_low_result.second && !(net_abs_mid_result.second)) { // If
                velocity falls to mid band
43              cout << (dir * spectrum_handler.sc(mid)) << endl;
44          }
45          else { // If velocity falls to low band
46              cout << (dir * spectrum_handler.sc(low)) << endl;
47          }
48
49          frame_handler.revert_hamming_the_frame(); // Dividing the frame by Hamming window
50      }
51
52  }
53  ...
```

Program najskôr zistí znamienko a absolútnu hodnotu rýchlosti a na základe toho potom spočíta prvý spektrálny moment v odpovedajúcom rozsahu indexov. Tento postupne pre prebiehajúce momenty výpočtu FFT vypisuje.

4.5 Modusový filter

Ako sme zistili už pri návrhu postupu vyhodnocovania rýchlosti v MATLABe, výsledok, dosiahnutý pomocou postupu z časti 4.4, je nutné filtrovať. Hlavnými (a iste celkom prirodzenými) dôvodmi sú nepresnosti algoritmov, vyhodnocujúcich jednotlivé parametre[vii] rýchlosti, ktoré spôsobujú „veľké“ zmeny alebo snaha o dosiahnutie prirodzeného priebehu rýchlosti, ktorý nebude rozkmitaný – v tomto prípade ide skôr o akúsi estetickú záležitosť; Stále však dosť podstatnú na to, aby sme jej venovali pozornosť.

Pre filtráciu výsledkov našich vyhodnocovacích mechanizmov sme v časti 3.6 používali ako hlavný filter filter *modusový*. Keďže štandardný balíček MATLABu neobsahuje implementáciu niečoho ako *moving mode*, vytvorili sme si vlastný. Našu implementáciu možno zhliadnuť v kóde E.29. Pri jej vytváraní sme počítali s tým, že máme k dispozícii kompletné výstupy determinantov rýchlosti pre celú nahrávku. V praxi (a v tejto časti) je to však nedosiahnuteľný stav a tak sme museli vytvoriť implementáciu inú, počítajúcu s príchodom jednotlivých hodnôt signálu[viii] postupne v čase.

Vytvorili sme triedu `Mode_filter`, ktorej popis funkcií by mal nasledovať až *po* objasnení štruktúry, s ktorou pracuje. V prvom rade je nutné ujasniť si mapovanie výstupu na vstup. Pri počítaní modusu zo vzoriek v okne dĺžky N_{mode} sme sa rozhodli priradiť výsledok tej vzorke vstupného signálu, ktorá sa nachádza *uprostred* okna, teda vzorke s indexom[ix] $\lfloor N_{\text{mode}}/2 \rfloor - 1$. Pri filtrácii,

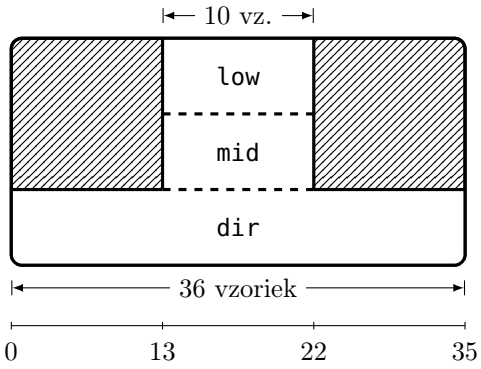
[vii] Ako sa čitateľ iste dovtípil z predchádzajúceho textu, ide o zaradenie absolútnej hodnoty rýchlosti do podintervalov a určenie smeru.

[viii] V tomto prípade *signál* označuje bitový tok z výstupu mechanizmov vyhodnotenia rýchlosti, približne tisícdvadsaťštyrikrát pomalší, než tok vzoriek z radarového senzoru.

[ix] Indexujeme od 0.

využívajúcej okno istej dĺžky musíme počítať s oneskorením, ktoré vznikne naplnením vyhodnocovacieho okna do tej miery, aby z neho bolo možné určiť relevantný výsledok – pri našom prístupe priradenia výsledku vzorke uprostred okna potrebujeme, aby bola uprostred okna vzorka užitočného signálu a nie iba nejaká, hoci známa, hodnota, na ktorú sme prvky okna inicializovali. S týmto faktom samozrejme nie je žiaden problém. Ten ale (i keď len malý) nastane v prípade, že používame pre užitočný signál niekoľko rôznych filtrov o rozdielnej dĺžke okna. Musíme zabezpečiť, aby boli okná v takej vzájomnej polohe[x], aby si odpovedali ich stredy. Filtre s kratším oknom potom preberú oneskorenie toho filtru, ktorého okno je najdlhšie – stretávame sa tu teda s prirodzenou požiadavkou, aby pre zdarné vyhodnotenie vstupu boli prítomné výsledky *všetkých* mechanizmov, ktoré sa starajú o jeho vyhodnotenie.

Pretože naše mechanizmy sú tri a vhodná dĺžka ich okien bola určená pomocou postprocessingového vyhodnotenia v MATLABe, ako okno pobitového algoritmu sme využili štruktúru z obrázku 4.10. Ide o pole o rozmeroch 3×36,



Obr. 4.10: Trojokno modusového filtru, použité v triede `Mode_filter`.

kde prvý riadok vo svojich prvkoch s indexami 13 až 22 obsahuje okno pre filtráciu výsledkov neurónovej siete, zadeľujúcej vstup do rozmedzia rýchlostí 0 až 0,2367 m/s, druhý riadok na tých istých indexoch okno pre filtráciu výsledkov siete, ktorá určuje prítomnosť vstupu do rozmedzia rýchlostí 0,2367 až 1,0832 m/s a tretí riadok v celej svojej dĺžke tvorí okno pre filtráciu výsledkov mechanizmu, určujúceho na základe súčtov PSD vo vyššie spomínaných medziach smer pohybu.

Samotná trieda `Mode_filter` je načrtnutá na obrázku 4.11 a skladá sa z týchto dôležitých premenných a funkcií:

`int window_width_<dir/low/mid>` sú len akýmisi aliasmi pre hodnoty postupne 35, 10 a 10. Pre lepšiu čitateľnosť kódu sme dali prednosť týmto pred priamym použitím číselných hodnôt.

`int* window` je ukazovateľom na pole takého typu, aký je načrtnutý na obrázku 4.10. Rozmery taktiež odpovedajú rozmerom z obrázku. Pri indexovaní tohoto poľa nevyužívame „dvojrozmerný“ zápis, ale manuálny prepočet vhodných indexov v jednorozmernom poli.

[x]Možno by bolo názornejšie nazvať to neformálne „Aby boli okienka tak poposúvané po vstupnom signále.“

`int n_samples_inside` je inicializovaná nulou a inkrementovaná pri každom volaní funkcie `void add_values(int, int, int)`, ktoré prebieha pred takzvanou *inicializáciou*. Cieľom tejto premennej je indikovať dosiahnutie minimálneho počtu užitočných signálových hodnôt, načítaných do okien (riadkov) poľa `window`. Tento počet je rovný polovici dĺžky najdlhšieho[xi] okna, teda $36/2 = 18$. Po dosiahnutí tejto hodnoty je preputá indikácia `bool initialized` do stavu `true`.

`int[3][3] howmany` je poľom, ktorého riadky odpovedajú riadkom poľa `window` a stĺpce obsahujú počet hodnôt -1, 0 a 1 (v poradí zľava doprava) v tej časti príslušného riadku poľa `window`, ktorá odpovedá oknu príslušného modusového filtru. Indexáciu okien v jednotlivých riadkoch a tiež v celkovom poli možno zhrnúť tak, ako v nasledujúcej tabuľke. Pretože spracované hodnoty z výstupu použitých neurónových sietí, na-

Indexy v riadku	Indexy v celom poli
13 až 22	13 až 22
13 až 22	49 až 58
0 až 35	72 až 107.

rozdiel od hodnôt ukazovateľa smeru pohybu, neobsahujú hodnotu -1, v prvých dvoch riadkoch bude hodnota v prvom stĺpci vždy *nulová*.

`int[3] UNINITIALIZED_RETURN_VALUE` a `modes` sú pomocné polia, súce dereferenciou návratovej hodnoty funkcie `int* mode()`. Referencia na prvé z nich je navrátená v prípade neinicializovaného výpočtu a slúži teda ako indikátor tohoto faktu. Tá druhá obsahuje užitočné výsledky.

`int max` a `bool initialized` sú pomocnými premennými, zlepšujúcimi čitateľnosť kódu.

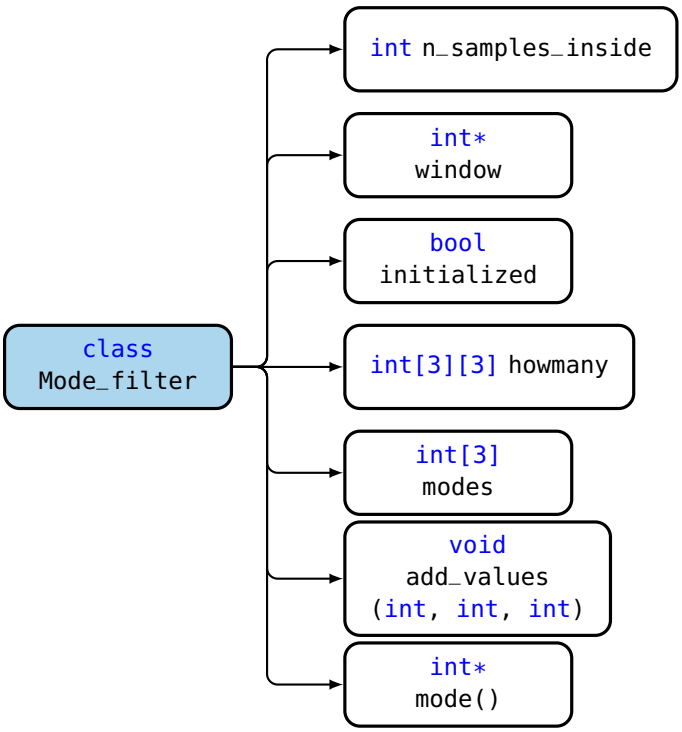
Konštruktor priradí aliasom šírky okien ich hodnoty, alokuje pamäť pre pole `window` a toto pole inicializuje nulovými hodnotami. Inicializuje tiež hodnoty pomocných premenných.

`void add_values(int, int, int)` posunie hodnoty v rámci každého riadku poľa `window` o jednu doprava a do prvého stĺpca vloží vektor

$$\begin{pmatrix} \text{prvý argument} \\ \text{druhý argument} \\ \text{tretí argument} \end{pmatrix}. \tag{4.5}$$

Pokiaľ výpočet nie je inicializovaný, teda pokiaľ je indikátor `initialized` rovný `false`, je navyše inkrementovaná hodnota premennej `n_samples_inside`. Následuje test na hodnotu tejto premennej a v prípade jej zhody s požadovaným minimom je do stavu `true` prepnutá premenná `initialized`.

`int* mode()` zabezpečuje výpočet samotného modusu z jednotlivých okien poľa `window`. Vráti ukazovateľ na pole `modes` alebo ukazovateľ na pomocné pole `UNINITIALIZED_RETURN_VALUE` a to v závislosti od stavu naplnenia najdlhšieho z okien.



Obr. 4.11: Hlavné premenné a funkcie triedy `Mode_filter`.

Vzhľadom na oneskorenie filtrovaných hodnôt oproti prichádzajúcim hodnotám vektorov spektrálnej výkonovej hustoty $\vec{p}$, ktoré je rovné[xii]

$$18 \text{ vzoriek} = 18 \cdot 1024 \cdot \frac{1}{44100} \text{ s} \approx 0,42 \text{ s}, \tag{4.6}$$

je nutné uchovávať taktiež posledných 18 hodnôt prvých spektrálnych momentov, ktoré sú na výstupe celého spracovania ovplyvňované filtrovanými hodnotami výsledkov determinantov parametrov rýchlosti. Oproti vyhodnoteniu bez filtrácie nám to prináša dve nevýhody, ktorými sú

- Už spomenutá nutnosť ukladať niekoľko posledných hodnôt prvých spektrálnych momentov C
- Vzhľadom na neznalosť výstupov neurónových sietí, zadeľujúcich rýchlosť do istých intervalov, v momente výpočtu C je potrebné počítať C zo *všetkých troch* rozmedzí.

Na uloženie predpočítaných trojíc prvých spektrálnych momentov bola použitá šablóna `Delay_line`, ktorá reprezentuje FIFO štruktúru s fixnou dĺžkou. Jej názov odpovedá účelu použitia pri spracovaní signálu lepšie, než napríklad `Cyclic_buffer`. Jej preťažený konštruktor obsahuje povinný parameter typu `int`, udávajúci dĺžku oneskorovacej linky a nepovinný parameter typu `int`,

[xi] Takže toho na filtráciu smeru pohybu.
[xii] Veríme, že takáto hodnota oneskorenia je akceptovateľná.

ktorý značí počet „riadkov“ tejto štruktúry – pokiaľ by malo n diskrétnych signálov mať rovnaké oneskorenie, možno pre ne použiť jednu linku s n riadkami.

Preťažená funkcia `void push_back(parametre)` umožňuje v závislosti od vstupných parametrov (buď jednoduchá hodnota alebo ukazovateľ na pole hodnôt) na koniec linky uložiť hodnoty. Pokiaľ je linka viacriadková a parametrom je referencia na pole, sú do riadkov postupne vkladané hodnoty z odkazovaného poľa. Pokiaľ je linka viacriadková a vstupom je jediná hodnota, je táto uložená na koniec *každého* z riadkov.

Funkcia `T* get_first()` vracia referenciu na pole, pozostávajúce z hodnôt na nultých indexoch jednotlivých riadkov a to i v prípade, že je linka jednoriadková.

Šablóna `Delay_line` je uvedená v kóde E.30.

4.6 Príklad kompletného programu

Vzhľadom na to, že filtrácia výsledkov rýchlostných kvalifikátorov je posledným krokom[xiii] na ceste od vzoriek k výstupu, rozhodli sme sa nepokračovať v tradícii písania podprogramov pre každú logickú časť výpočtu. Namiesto toho uvedieme kompletný program, simulujúci príchod vzoriek a výpočet rýchlosti z nich.

Program využíva knižnice z tabuľky 4.2 a nami napísané hlavičkové súbory,

`cstdlib`	`cstdio`	`fstream`	`iostream`
`cstring`	`cmath`	`fftw3.h`	`iomanip`
`climits`	`vector`	`float.h`	`algorithm`

Tab. 4.2: Knižnice, použité v hlavnom programe.

obsahujúce definície tried a typov, ktoré sú zhrnuté v tabuľke 4.3.

`types.h`	`Constants_datastore.h`	`Mode_filter.h`
`Wavreader.h`	`Sample_source.h`	`Frame_operator.h`
`Spectrum_operator.h`	`Neural_network.h`	`Delay_line.h`

Tab. 4.3: Hlavičkové súbory, použité v hlavnom programe.

Program samotný začína volaním konštruktora triedy `Constants_datastore` – v objekte tejto triedy sú, ako už bolo spomenuté, uložené konštanty, potrebné pre rôzne ďalšie výpočty a ukazovateľ naň poznajú mnohé ďalšie objekty. Hneď prvým z nich je objekt triedy `Wavreader`, ktorý sa postará o čítanie dát zo súboru vo WAVE formáte. Cestu k tomuto súboru zadáme ako argument funkcie `main`.

```
1  int main(int argc, char* argv[]) {
2  Constants_datastore constants;
3  Wavreader wav(&constants);
4
5  if (argc == 2) {
```

[xiii] Reálne zostáva ešte prepočet z indexu v poli PSD na hodnotu rýchlosti; Nakoľko však ide o bijektívne zobrazenie, je táto úloha triviálna.

```
6       wav.read_file(argv[1]); // Argument should be an character array, containing path to .
            wav file. This wav file should be 60s long recording, recorded by radar_recorder
            .m
7   }
8   else {
9       cout << "Error in function main: Unexpected number of arguments. Main function expects
             one argument, which is a path to .wav file. Example: ./velocity_main \"/path/to
            /file\"." << endl;
10      exit(EXIT_FAILURE);
11  }
12  ...
13  }
```

Dáta zo súboru máme teraz uložené v jednej z premenných objektu `wav`. Pretože môžeme chcieť nenačítať všetky vzorky zo súboru, nasledujú kroky výberu vzoriek. V premennej `int constants::N_READ_SAMPLES` je uvedený celkový počet vzoriek, ktoré chceme načítať; Index prvej z nich je zas uložený v premennej `int constants::BEGINNING`. Pokiaľ chceme tieto hodnoty zmeniť, je najjednoduchšie prepísať konštruktor triedy `Constants_datastore`.

Prebehne test na to, či sú naše požiadavky oprávnené, teda či je k dispozícii požadovaný počet vzoriek od požadovaného začiatku. V prípade, že nie, je začiatok nastavený na prvú vzorku a počet načítavaných vzoriek na maximálny možný. Toto nastavenie prebehne i vtedy, keď ako požadovaný počet vzoriek uvedieme limitnú konštantu `INT_MAX`. Alokujeme pamäť na požadované vzorky ako pole `double data` a naplníme ho nimi. Vzorky máme uložené a objekt `wav` môžeme deštruovať.

```
12  ...
13  if ((constants.get_N_READ_SAMPLES() == INT_MAX) || (constants.get_N_READ_SAMPLES() > (wav.
        get_n_read_samples() - wav.get_BEGINNING()))) { // If indicated so or if requested
        amount of samples is not disponsible
14      constants.set_N_READ_SAMPLES(wav.get_n_read_samples()); // Set number of read samples
            to maximum which actual .wav file can offer.
15      constants.set_BEGINNING(); // Set beginning to zero.
16  }
17
18  double* data = (double*)malloc(2 * constants.get_N_READ_SAMPLES() * sizeof(double));
19  for (int i = 0; i < 2 * constants.get_N_READ_SAMPLES(); i++) {
20      data[i] = double(wav.get_data()[2 * constants.get_BEGINNING() + i] / constants.
            get_SHORTSIZE());
21  }
22
23  wav.~Wavreader(); // Destructing wavreader
24  ...
```

Nasleduje klasické vytvorenie pracovných polí pre FFTW a tiež pre ďalšie výpočty. Polia `fftw_complex[] frame` a `fftw_complex[] spectrum` sme si už priblížili. Ich počet prvkov je daný parametrami z objektu `constants`. Argument `flags` vo funkcii na vytvorenie plánu FFT sme zvolili `FFTW_MEASURE`, ktorým je deklarovaná požiadavka na to, aby na začiatku prebehlo niekoľko testov na výpočtový čas FFT pre transformáciu požadovanej dĺžky a na základe ich výsledkov bol vytvorený plán. Toto nastavenie je odporúčané použiť pre prípady, kedy sme ochotní obetovať rýchlosť inicializácie za výhodu v podobe rýchlejších transformácií počas behu programu [4]. Pokiaľ by sa stalo, že oneskorenie na začiatku by bolo pri aplikácií programu v samostatnej výpočtovej jednotke neakceptovateľné a pokiaľ by bola táto dostatočne rýchla[xiv], je tento argument jednou z vecí, ktoré bude potrebné v programe zmeniť.

[xiv] Ako už bolo spomínané, tento softvér by sme chceli využiť v samostatnej radarovej jednotke.

```
24  ...
25  fftw_complex* frame; // Input to complex, one-dimensional DFT.
26  fftw_complex* spectrum; // Output to complex, one-dimensional DFT.
27  frame = (fftw_complex*)fftw_malloc(constants.get_N_FFT() * sizeof(fftw_complex)); // 8192
        ongoing samples as an input for FFT computation
28  spectrum = (fftw_complex*)fftw_malloc(constants.get_N_FFT() * sizeof(fftw_complex)); //
        8192 frame's spectrum values
29  fftw_plan plan = fftw_plan_dft_1d(constants.get_N_FFT(), frame, spectrum, FFTW_FORWARD,
        FFTW_MEASURE);
30  ...
```

Ako ďalší krok vytvoríme objekty na simuláciu toku vzoriek a na prácu s operačnými poľami. Môžeme tiež rovno vytvoriť dva objekty neurónových sietí a naplniť ich odpovedajúce premenné potrebnými dátami – váhami, kvocientami a extrémami riadkov matíc tréningových dát, ktoré sme získali exportom z Matlabu napríklad pomocou skriptu E.21. Ďalej konštruujeme modusový filter a vytvoríme si pomocné premenné na odovzdávanie výsledku.

Táto časť bude v ďalšej aplikácií iste pozmenená – nebudeme totiž dostávať vzorky zo súboru a odpadne teda nutnosť vytvárania objektu triedy `Sample_source`.

```
30  ...
31  Sample_source sample_shooter(data, &constants, frame);
32  Frame_operator frame_handler(frame, spectrum, &constants);
33  Spectrum_operator spectrum_handler(spectrum, &constants);
34
35  Neural_network nn_abs_low(&constants, abs_low, spectrum);
36  Neural_network nn_abs_mid(&constants, abs_mid, spectrum);
37  nn_abs_low.load(); // Loading weight and bias data and row maxima of training matrix
38  nn_abs_mid.load(); // Loading weight and bias data and row maxima of training matrix
39
40  Mode_filter modefilter; // Mode filter, filtering results of direction determination and
        results of neural networks for ansolute value of velocity determination.
41  int* modes; // Pointer to result of mode filter.
42  Delay_line <double> sc(17, 3); // Length: length of direction mode filter window width / 2
        - 1. Number of rows: 3.
43  double sc_actual[3]; // Actual values of all three spectral centroids. Temporary variable.
44  ...
```

Vytvorme si ešte premenné na ukladanie výsledkov a môžeme pristúpiť k „vpúšťaniu“ vzoriek signálu do spracovania. V tomto programe je príchod vzorky simulovaný funkciou `Sample_source::shoot_sample()`, v samostatnej aplikácii budeme očakávať príchod vzorky z výstupu A/D prevodníku. Nakoľko sme teraz obmedzení dĺžkou nahrávky, museli sme ďalšie spracovanie uzavrieť do cyklu, ktorý prebieha cez počet načítaných vzoriek z premennej `data`. Pri použití externého načítavania vzoriek potom použijeme cyklus nekonečný s vhodne zvoleným indikátorom jeho ukončenia. Všetko ďalšie spracovanie bude prebiehať len každých 1024 vzoriek, čo zaručíme podmienkou z riadku 49 nasledujúceho úseku kódu. Telo programu pri jej splnení tu vynecháme a budeme sa mu venovať až po ďalšom vysvetlení.

```
44  ...
45  sample_shooter.initialize_frame(); // Filling the frame with first 8191 samples
46  ddpair_t actual_sample(0.0,0.0);
47  ddpair_t net_abs_low_result(0.0, 0.0);
48  ddpair_t net_abs_mid_result(0.0, 0.0);
49
50  for (int i = 0; i < (constants.get_N_READ_SAMPLES() - (constants.get_N_FFT())); i++) { //
        For number of samples loaded from .wav file minus number of samples, which were feed
         to frame by initialize_frame() function.
51
52      actual_sample = sample_shooter.shoot_sample(); // Sample comes
53      frame_handler.add_new_sample(&actual_sample); // Add it to frame
54
```

```
55      if (i % (constants.get_N_FFT() - constants.get_NOVERLAP()) == 0) { // every 1024th
            sample
56          ...
57      }
58      ...
59  }
60  ...
```

Pri nekonečnom cykle by sme riadky 50 až 59 predchádzajúceho úseku kódu nahradili napríklad takto:

```
50  int index = 0;
51  while(true) {
52
53      ... // Save incoming sample to frame
54
55      index++;
56
57      if (index == 1023) {
58
59          ... // Some processing
60
61          index = 0;
62      }
63
64      if (/* Condition */) break;
65  }
```

V tele bloku tejto podmienky potrebujeme urobiť nasledujúce úkony:

1. Spočítať spektrálnu výkonovú hustotu $\vec{p}$ zo vzoriek, uložených v poli `frame`. To pozostáva z
   i. Vynásobenia vzoriek v poli `frame` Hammingovym oknom rovnakej dĺžky, ako toto pole
   ii. Výpočtu diskrétnej Fourierovej transformácie z takto upravených vzoriek
   iii. Prevodu výsledku na hodnoty spektrálnej výkonovej hustoty.
2. Volať mechanizmy, určujúce parametre rýchlosti a získať ich výsledky.
3. Vypočítať prvý spektrálny moment pre všetky tri rozmedzia z tabuľky 4.1 a trojice hodnôt uložiť do oneskorovacej linky, v ktorej „počkajú“ na filtrované výsledky determinantov charakteru rýchlosti.
4. V závislosti na charaktere rýchlosti vybrať ten správny spomedzi trojice C a vynásobiť ho tým správnym znamienkom.
5. Vrátiť výsledok. Vrátenie výsledku je v tejto časti realizované jednoduchým výpisom – pri testovaní sme mohli vypísané výsledky $\mathcal{V} \in \langle -371; 371 \rangle$, odpovedajúce rýchlosti

$$v_x = \mathcal{V} \cdot 0,0473 \text{ m/s} \tag{4.7}$$

načítať opäť do MATLABu, zobraziť ich a skontrolovať tak správnosť výpočtu.

Náš C++ zápis týchto krokov vyzerá tak, ako je uvedené v ďalšom úseku kódu. Týmto úsekom sú nahradené tri bodky v riadku 56 posledného uvedeného skutočného úseku kódu – ide teda o telo bloku podmienky.

```
56  /* ---------------------- Computing PSD --------------------- */
57  frame_handler.hamming_the_frame(); // Multiplying the frame by Hamming window.
58  fftw_execute(plan); // Compute FFT
59  frame_handler.psd(); // Transfer it to normalized PSD
60
61  /* --------------- Utilizing neural networks ---------------- */
62  nn_abs_low.net_walkthrough();
63  nn_abs_mid.net_walkthrough();
64
65  /* Adding values of velocity parameter to mode filter windows */
66  modefilter.add_values(nn_abs_low.get_result().second, nn_abs_mid.get_result().second,
        frame_handler.direction_detector());
67
68  /* -------------- Computing SC, storing them ---------------- */
69  sc_actual[0] = spectrum_handler.sc(low);
70  sc_actual[1] = spectrum_handler.sc(mid);
71  sc_actual[2] = spectrum_handler.sc(all);
72  sc.push_back(sc_actual);
73
74  /* ------------ Getting result from mode filter ------------- */
75  modes = modefilter.mode();
76
77  /* ---------- Influencing SC by velocity parameters --------- */
78  if((modes[0] == 0) && (modes[1] == 0)) { // If |velocity| is below 'low' level
79      cout << sc.get_first()[0] * modes[2] << endl;
80  }
81  else if ((modes[0] == 1) && (modes[1] == 0)) { // If |velocity| is in middle band
82      cout << sc.get_first()[1] * modes[2] << endl;
83  }
84  else if ((modes[0] == 1) && (modes[1] == 1)) { // If |velocity| is above the middle band
85      cout << sc.get_first()[2] * modes[2] << endl;
86  }
87  else cout << INT_MIN << endl; // Error indicator
88
89  /* ------------- Reverting changes to frame array ----------- */
90  frame_handler.revert_hamming_the_frame(); // Dividing the frame by Hamming window
```

Na záver samozrejme uvoľníme pamäť.

```
94  ...
95  fftw_destroy_plan(plan);
96  fftw_free(frame);
97  fftw_free(spectrum);
98  free(data);
99
100 return EXIT_SUCCESS;
101 } /* ----------  end of function main  ---------- */
```

4.7 Použitie

Po overení správneho fungovania všetkých častí kódu sme pristúpili k vytvoreniu skriptov, ktoré umožnia jednoduché zostavenie programu a zobrazenie výsledkov. Všetky informácie v tejto časti predpokladajú prácu s obsahom archívu `velocity_cpp.tar.gz`, ktorý je prílohou tejto práce.

4.7.1 Makefile

Pre zjednodušenie kompilácie a zostavovania programu bol vytvorený *Makefile*. Makefile je súbor so špecifickou syntaxou, obsahujúci inštrukcie programu *GNU Make* [16]. Na základe nich program *make* určuje poradie prekladu a zostavovania programových súborov, nastavuje správanie kompilátoru a linkeru či odstraňuje niektoré súbory, ktoré už nie sú potrebné.

Pred samotným prekladom je nutné nainštalovať knižnicu *FFTW3* [4]. Na jej webových stránkach sa nachádzajú všetky potrebné informácie o postupe.

Aby nebolo nutné sťahovať túto knižnicu z externého zdroja, bola zahrnutá do priloženého archívu. Po extrakcii súborov z archívu do zložky `folder` zadáme pre inštaláciu knižnice *FFTW3* do príkazového riadku jednoducho

```
~/folder$ make fftwinstall
```

a pre kompiláciu a linking našeho programu

```
~/folder$ make all
```

V zložke nám pribudne súbor `velocity_main`. Tento je spustiteľný a vyžaduje *práve jeden* argument, ktorým je kompletná cesta k súboru s nahrávkou z radaru. Po spustení program začne vypisovať jeden po druhom čísla n, ktoré sú obrazmi dopplerovských frekvencií zo skúmaného rozsahu v bijektívnom zobrazení κ. Pre inverzný prevod platí

$$\kappa^{-1}(n) = f_d = \frac{f_s}{N_{\mathrm{FFT}}}\, n \approx 5,3833 \cdot n \text{ Hz}\,. \tag{4.8}$$

Prepočtom podľa vzťahu (1.3) potom možno jednoducho získať hodnotu detekovanej rýchlosti.

Špeciálnym prípadom je výstup hodnoty `INT_MIN`, ktorá indikuje nedostatok vzoriek, načítaných do okna filtru – bližšie informácie o tejto problematike sa nachádzajú v časti 4.5.

4.7.2 Vizualizácia

Na jednoduché overenie správnej funkcie bol zvolený tento postup:

1. Necháme program `velocity_main` vyhodnotiť celú nahrávku.
2. Výstup z tohoto programu ukladáme do súboru.
3. Necháme Matlab vypočítať spektrogram z vyhodnocovanej nahrávky.
4. Spektrogram necháme vykresliť. Do Matlabu importujeme výstup nášho programu a pozmeníme iba jeho mierku, aby rozsahovo odpovedal hodnotám na osiach spektrogramu.

Aby nebolo nutné jednotlivé kroky robiť ručne, bol vytvorený bash skript, ktorý urobí všetky tieto úkony jeden po druhom. Pre vykreslenie výsledkov potom stačí zadať

```
~/folder$ ./view_results "</.../vyhodnocovana_nahravka.wav>" "</.../folder/>"
```

kde `folder` je aktuálna zložka, teda zložka, do ktorej boli extrahované súbory z pôvodného archívu.

KAPITOLA 5

Využitie v systéme určovania polohy

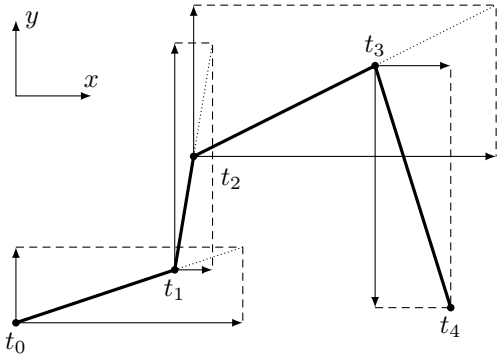
Využitie meradla rýchlosti v systéme určovania polohy otvára veľký priestor na experimenty. Pojem *systém určovania polohy* ako taký nijako nešpecifikuje, akým spôsobom má byť poloha určená. V tejto kapitole priblížime základnú myšlienku využitia zapojenia niekoľkých senzorov rýchlosti pre samostatný výpočet relatívnej polohy. Následne sa budeme v stručnosti venovať myšlienke aplikácie takéhoto senzoru na zlepšenie presnosti určenia polohy *družicovým* systémom.

5.1 Systém určovania relatívnej polohy

Myšlienka výpočtu polohy, vztiahnutej k istému známemu počiatku, známa ako *dead reckoning*, je založená na základnej rovnici kinematiky

$$\vec{r} = \vec{r}_0 + \int_T \vec{v}(t)\ \mathrm{d}t \tag{5.1}$$

a ilustrovaná na obrázku 5.1.



Obr. 5.1: K princípu využitia radarových meračov rýchlosti v systéme určenia relatívnej polohy.

Predstavme si situáciu pohybu bodu po rovine. Pohyb začína v bode, odpovedajúcom času t_0, ktorého polohu poznáme a označíme ako $\vec{r}_0$. Systém určovania polohy tu tvorí zapojenie dvoch meračov rýchlosti[i], z ktorých každý meria rýchlosť v jednom z dvoch navzájom kolmých smerov. Dáta z meračov sú zaznamenávané s intervalom Δt – na obrázku 5.1 teda platí $t_i = t_0 + i\,\Delta t$ pre $i \in \{0, 1, 2, 3, 4\}$. V každom z bodov merania sú rýchlosti, udané meračmi, znázornené dvoma na seba kolmými vektormi. Výsledná poloha v čase $t_k = t_0 + k\,\Delta t$ je na základe diskretizácie rovnice (5.1) daná v ideálnom prípade ako

$$\vec{r} = \vec{r}_0 + \sum_{i=0}^{k-1} (\vec{v}_i^{\,x} + \vec{v}_i^{\,y})\ \Delta t\,. \tag{5.2}$$

Tento prípad je prirodzene možné rozšíriť do trojrozmerného priestoru príslušnou úpravou použitých vzťahov. V ďalšom texte však naopak jeden z rozmerov uberieme a budeme sa zaoberať úlohou v jedinom rozmere – pohyb v jednotlivých smeroch kartézskych osí budeme považovať za nezávislý a vyriešenie trojrozmernej úlohy bude teda iba superpozíciou riešení troch jednorozmerných úloh.

Takýto systém neobsahuje žiadnu stabilizáciu – pokiaľ je v čase t_i rýchlosť meračom určená nesprávne, okamžite sa to premietne do nesprávneho výpočtu polohy v ďalšom časovom bode a všetkých časoch nasledujúcich. Je teda nutné zaviesť mechanizmus stabilizácie, garantujúci, že *evidentne nesprávnej* hodnote bude pri výpočte polohy v nasledujúcom časovom bode priradená menšia *váha*, než hodnote, ktorá fyzikálne odpovedá predpokladanému priebehu. Štandardným postupom pre takéto naloženie s dátami, ovplyvnenými neistotou, je *Kalmanova filtrácia*, ktorej algoritmus uvádzame v časti 5.2.

[i] Napríklad radarových.

5.2 Kalmanov filter

Algoritmus Kalmanovej filtrácie, publikovaný *Rudolfom E. Kálmánom*[ii] v roku 1960[iii] [18], si pre svoju priamočiarosť, implementačnú jednoduchosť a hlavne dobré výsledky získal celosvetovú obľubu a stal sa bežne používaným algoritmom, využívaným v širokom spektre aplikácií, siahajúcom od ekonomiky cez spracovanie obrazu a zvuku až napríklad po oblasť nášho záujmu – systémy určovania polohy.

Základnou premennou Kalmanovho filtru je vektor s názvom *stav* (*state*). Tento vektor predstavuje usporiadané hodnoty premenných istého *modelu*. Filter pracuje v diskrétnom čase[iv] a jednou z jeho úloh je pomocou modelu *odhadovať* stavy systému v časoch s istým rozostupom T[v]. Model je popisom správania popisovaného systému v čase. Je však iba modelom a reálne správanie systému sa môže odlišovať vplyvom vonkajších náhodných procesov, ktoré však kvôli ich náhodnému charakteru nemožno zahrnúť do deterministickej časti modelu. Pri zachovaní notácie z [18] možno túto časť vyjadriť ako

$$\vec{x}_k = \mathbb{F}_k\,\vec{x}_{k-1} + \mathbb{G}_k\,\vec{u}_k + \vec{w}_{k-1}\,, \tag{5.3}$$

kde $\vec{x}_{k-1}$ predstavuje stav v čase t, $\vec{x}_k$ stav v čase $t+T$, matica $\mathbb{F}_k$ model prechodu stavového vektoru od hodnoty v čase t k hodnote v čase $t+T$, $\mathbb{G}_k$ maticu popisu interných vplyvov[vi], ktorá modeluje prípadné ovplyvňovanie systému známymi veličinami v niektorých časových úsekoch – typicky napríklad občasné zapnutie horáku balónu, známe prúdenie pri pohybe plavidla po vode a tak ďalej. Vektor $\vec{u}_k$ vektor hodnôt interných vplyvov medzi časmi t a $t+T$ a vektor $\vec{w}_{k-1}$ gaussovský šum stavu s *nulovou* strednou hodnotou.

Druhou časťou procesu *odhadu* v Kalmanovom filtri je odhad *kovariančnej matice* stavu. Kovariančná matica náhodného vektoru $\vec{v} = (v_1,\, v_2, \ldots, v_n)^T$ je definovaná ako

$$\mathrm{Cov}[\vec{v}] = \begin{pmatrix} \mathrm{Var}[v_1] & \mathrm{Cov}[v_1,\, v_2] & \cdots & \mathrm{Cov}[v_1,\, v_n] \\ \mathrm{Cov}[v_2,\, v_1] & \ddots & & \vdots \\ \vdots & & & \\ \mathrm{Cov}[v_n,\, v_1] & \cdots & & \mathrm{Var}[v_n] \end{pmatrix}. \tag{5.4}$$

Pre prípad, kedy je náhodný vektor $\vec{v}$ tvorený našim vektorom stavu, $\vec{x_k}$, označíme jeho kovariančnú maticu $\mathbb{P}_k$ a pre jej odhad zo stavu minulého bude platiť

$$\mathbb{P}_k = \mathbb{F}_{k-1}\,\mathbb{P}_{k-1}\,\mathbb{F}_{k-1}^T + \mathbb{Q}_{k-1}\,, \tag{5.5}$$

kde $\mathbb{Q}_{k-1}$ predstavuje kovariančnú maticu externých vplyvov, ktoré spôsobujú šum $\vec{w}_{k-1}$. Predpokladáme, že toto rušenie bude mať normálne rozdelenie a nulovú strednú hodnotu.

[ii] Maďarský pôvod tohoto autora vysvetľuje použitie dĺžňov v jeho mene. Vzhľadom na celosvetové využitie tohoto algoritmu sa však dĺžne z názvu algoritmu vytratili a zaužíval sa bezdĺžňový pojem *Kalmanov filter* namiesto trochu menej praktického *Kálmánov filter*.

[iii] O podobné postupy a rôzne variácie tohoto filtru sa zaslúžili mnohí autori.

[iv] Existujú varianty pre spojitý čas, všeobecne nazývané *continous-time Kalman filter*. V tejto práci sa nimi nebudeme zaoberať.

[v] Tento rozostup je vo všeobecnosti jedným z parametrov filtru; v našom prípade ho však môžeme považovať za konštantu.

[vi] Po anglicky *control matrix*.

Pomocou rovníc (5.3) a (5.5) môžeme teda vytvárať odhady ďalších a ďalších stavov, pričom do výsledkov zahŕňame neistoty, ovplyvňujúce náš model. Predošlá časť algoritmu Kalmanovho filtru je nazývaná *predpoveďou*, resp. po anglicky *prediction*.

Tou druhou časťou filtrácie je *korekcia* predikovaných hodnôt pomocou meraní niektorých veličín modelu. Tu je vhodné upraviť značenie – času t už nebude odpovedať jediný stav, $\vec{x}_k$, ale stavy dva – $\hat{x}_k^-$ ako odhad stavu v čase t na základe hodnoty v čase $t-T$ a modelu a $\hat{x}_k^+$ ako stav, korigovaný *meraním* v čase t, pričom rovnaký spôsob zápisu použijeme i pre odhad kovariančnej matice stavu. Rovnicu (5.3) teda prepíšeme na

$$\hat{x}_k^- = \mathbb{F}_k\,\hat{x}_{k-1}^+ + \mathbb{G}_{k-1}\,\vec{u}_{k-1}\,. \tag{5.6}$$

a rovnicu (5.5) na

$$\mathbb{P}_k^- = \mathbb{F}_{k-1}\,\mathbb{P}_{k-1}^+\,\mathbb{F}_{k-1} + \mathbb{Q}_{k-1}\,. \tag{5.7}$$

Korekciu odhadu stavu $\hat{x}_k^-$ meraním $\vec{y}_k$ potom uskutočníme na základe vzťahu

$$\hat{x}_k^+ = \hat{x}_k^- + \mathbb{K}_k(\vec{y}_k - \mathbb{H}_k\,\hat{x}_k^-)\,, \tag{5.8}$$

v ktorom matica $\mathbb{H}_k$ predstavuje určovateľ lineárnej kombinácie stavových veličín, ktorá je v aktuálnom meraní zachytená – pokiaľ by teda vektor stavu pozostával postupne z hodnôt veličín a_1, a_2 a a_3 a náš senzor by meral veličinu a_2, matica $\mathbb{H}_k$ by vyzerala

$$\mathbb{H}_k = (0\ 1\ 0)\,. \tag{5.9}$$

Matica $\mathbb{K}_k$ predstavuje tzv. *Kalmanov zisk* v čase korekcie. Preň podľa [18] bez ďalšieho odvodenia platí

$$\mathbb{K}_k = \mathbb{P}_k^-\,\mathbb{H}_k^T(\mathbb{H}_k\,\mathbb{P}_k^-\,\mathbb{H}_k^T + \mathbb{R}_k)^{-1}\,. \tag{5.10}$$

Matica $\mathbb{R}_k$ v rovnici (5.10) predstavuje kovariančnú maticu šumu merania a je teda analógiou matice $\mathbb{Q}_{k-1}$ vo vzťahu (5.5). Pokiaľ meriame iba jedinú veličinu, prejde $\mathbb{R}_k$ na jediné číslo, vyjadrujúce neistotu merania, teda *rozptyl meranej veličiny*.

Po inicializácii známym stavom x_0^+ a jeho kovariančnou maticou $\mathbb{P}_0^+$ prebieha algoritmus tak, ako je uvedené v algoritme 4.

Algoritmus 4 Diskrétny Kalmanov filter

procedure KalmanFilter($\hat{x}$, $\mathbb{P}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{G}$)
  for $k \leftarrow 1, 2\ldots, n$ **do**
    $\mathbb{P} \leftarrow \mathbb{F}\,\mathbb{P}\,\mathbb{F}^{\mathbb{T}} + \mathbb{Q}$
    $\mathbb{K} \leftarrow \mathbb{P}\,\mathbb{H}^T(\mathbb{H}\mathbb{P}\mathbb{H}^T + \mathbb{R})^{-1}$
    $\hat{x} \leftarrow \mathbb{F}\,\hat{x} + \mathbb{G}\vec{u}$ ▷ Predikcia. $\vec{u}$ predpokladáme konštantný.
    if Sú k dispozícii nové merania **then**
      Načítaj nové merania vrátane matice $\mathbb{H}$
      $\hat{x} \leftarrow \hat{x} + \mathbb{K}(y - \mathbb{H}\,\hat{x})$ ▷ Korekcia.
    else
      $\mathbb{H} \leftarrow \mathbb{O}$ ▷ $\mathbb{O}$ predstavuje nulovú maticu príslušných rozmerov.
    end if
    $\mathbb{P} \leftarrow (\mathbb{I} - \mathbb{K}\,\mathbb{H})\,\mathbb{P}\,(\mathbb{I} - \mathbb{K}\,\mathbb{H})^T + \mathbb{K}\,\mathbb{R}\,\mathbb{K}^{\mathbb{T}}$

13: **end for**
14: **end procedure**

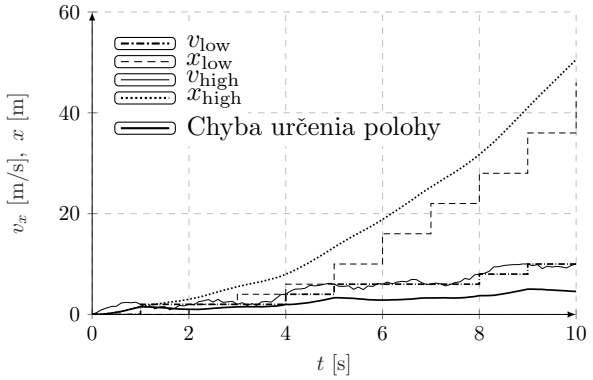
Pokiaľ nové merania k dispozícii nie sú, je matica $\mathbb{H}$ nulová, teda je nulový i Kalmanov zisk v ďalšom kroku a kovariančná matica i stavový vektor sú predikované, no nekorigované. V popise 4 je popisovaný filter s časovo konštantným modelom. Model je však možné predpokladať časovo premenný a príslušné matice ($\mathbb{F}$, $\mathbb{G}$, $\mathbb{Q}$, $\mathbb{R}$) v ktoromkoľvek kroku meniť. Rovnako ako rýdza predikcia bez korekcie pri absencii aktuálneho merania je možná i viacnásobná korekcia stavu v aktuálnom čase – možno tak využiť merania rôznych veličín z rôznych senzorov.

5.3 Rozšírený Kalmanov filter

Predpokladajme teraz nasledujúcu úlohu:

- Na automobile máme k dispozícii merania rýchlosti radarovým senzorom a merania absolútnej polohy a rýchlosti z istého družicového systému určenia polohy (bez ujmy na všeobecnosti predpokladajme GPS).
- Cieľom je čo najpresnejšie určovanie polohy vozidla v kvázireálnom čase.

Každý z týchto senzorov má svoje výhody i nevýhody. Družicový navigačný systém určí pri dobrom signále polohu i rýchlosť veľmi presne. Pokiaľ však signál nie je k dispozícii (typicky napríklad pri prechode automobilu tunelom), nemáme k dispozícii žiadnu informáciu o rýchlosti ani polohe. Ďalšou nevýhodou je pomerne veľký rozostup medzi jednotlivými údajmi z družicových meraní, čo spôsobuje chybu určenia stavu v časoch medzi časmi, v ktorých je stav vysielaný. Toto je ilustrované na obrázku 5.2.



Obr. 5.2: Simulácia vzniku chyby určenia polohy pri riedkom toku dát. Najskôr predpokladáme, že údaje o polohe užívateľa a jeho rýchlosti sú vysielané s nízkou frekvenciou a ich hodnota je platná po celú vysielaciu periódu. Nie je použitý žiaden odhad nasledujúceho stavu. Takéto priebehy sú znázornené ako x_{low} a v_{low}. Pri takejto frekvencii dát prichádzame o informácie o priebehu medzi meraniami – priebehy x_{high} a v_{high} znázorňujú rovnaké veličiny, no so stokrát vyššou frekvenciou vzoriek.

Naproti tomu stoja merania rýchlosti z radarového merača rýchlosti, ktorých hodnoty vieme získavať veľmi často, no polohu z nich vieme dopočítavať iba relatívne, na základe istého modelu.

Vyhodnocovaným stavom v tejto úlohe je stav

$$\vec{x} = (x,\, y,\, z,\, v_x,\, v_y,\, v_z)^T\,, \tag{5.11}$$

v ktorom vystupujú polohy $(x,\, y,\, z)$ a rýchlosti $(v_x,\, v_y,\, v_z)$ automobilu v troch navzájom kolmých smeroch. Zo snímača rýchlosti máme pre ktorýkoľvek (každý) časový bod údaje o rýchlosti. Pokiaľ označíme vektor radarových meraní y_r, bude teda platiť

$$\vec{y}_r = (v_{xr},\, v_{yr},\, v_{zr})^T\,. \tag{5.12}$$

Takéto meranie možno využiť v bežnom diskrétnom Kalmanovom filtri so základným kinematickým modelom, kde rovnica (5.6) má tvar

$$\hat{x}_k^- = \underbrace{\begin{pmatrix} 1 & 0 & 0 & T & 0 & 0 \\ 0 & 1 & 0 & 0 & T & 0 \\ 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbb{F}} \hat{x}_{k-1}^+\,. \tag{5.13}$$

Nakoľko platí

$$\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{pmatrix}, \tag{5.14}$$

za maticu výberu pre merania z radaru volíme

$$\mathbb{H} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{5.15}$$

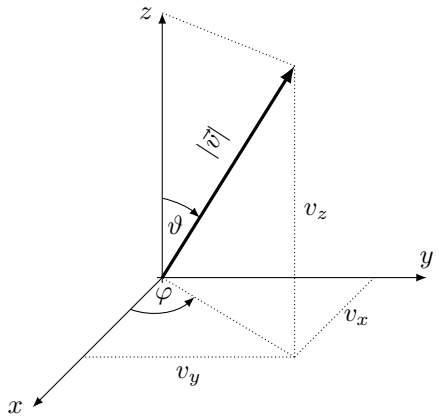
Druhým meraním, ktorým možno korigovať predikované hodnoty, bude meranie z GPS, poskytujúce vektor

$$\vec{y}_{\mathrm{GPS}} = (x_{\mathrm{GPS}},\, y_{\mathrm{GPS}},\, z_{\mathrm{GPS}},\, |\vec{v}|,\, \vartheta,\, \varphi)\,, \tag{5.16}$$

kde posledné tri prvky predstavujú vektor *zmeny polohy* za známy čas T_{GPS}[vii]. Známa zmena polohy za známy časový úsek, vynásobená týmto časovým úsekom, je priamo rýchlosťou. Problematický je prípad, kedy dostaneme vektor

[vii] Čas by mohol tvoriť tiež ďalšiu premennú stavu. Pretože ale merania z GPS, ktoré obsahujú tento údaj, prichádzajú s menšou frekvenciou, než merania z radarového senzoru rýchlosti, nebudeme čas zahŕňať do vektoru stavu a budeme predpokladať, že je jeho hodnota doručená ako samostatná premenná T_{GPS}. Ďalším postrehom k tomuto bodu je, že môže nastať o prípad, kedy nebudú dodané všetky zo zložiek merania $\vec{y}_{\mathrm{GPS}}$. Na ďalšie tvrdenie to nemá vplyv – pri korekcii stavu budeme v takomto prípade predpokladať, že chýbajúca súradnica ostáva z predošlého merania nemenná.

y_{GPS} vo vyššieuvedenom tvare, s hodnotami $|\vec{v}|$, ϑ a *phi*, odpovedajúcimi *sférickému* súradnicovému systému. Tento systém je kvôli ujasneniu značenia, spolu so systémom kartézskym, znázornený na obrázku 5.3.



Obr. 5.3: Objasnenie značenia v rovniciach (5.17)

V tomto momente nastáva pri použití štandardného diskrétneho Kalmanovho filtru problém – matica $\mathbb{H}_k$ v rovniciach (5.8) predstavuje koeficienty lineárnej kombinácie prvkov vektoru merania, y_k (resp. y_{GPS}), z ktorých dostávame prvky stavového vektoru. Vzťah medzi ($|\vec{v}|$, ϑ, φ) a (v_x, v_y, v_z) však nie je lineárny, nakoľko platí

$$\begin{aligned} |\vec{v}| &= \sqrt{v_x^2 + v_y^2 + v_z^2} \\ \vartheta &= \arccos\left(v_z / \sqrt{v_x^2 + v_y^2 + v_z^2}\right) \\ \varphi &= \arctan(v_y / v_x) \end{aligned} \tag{5.17}$$

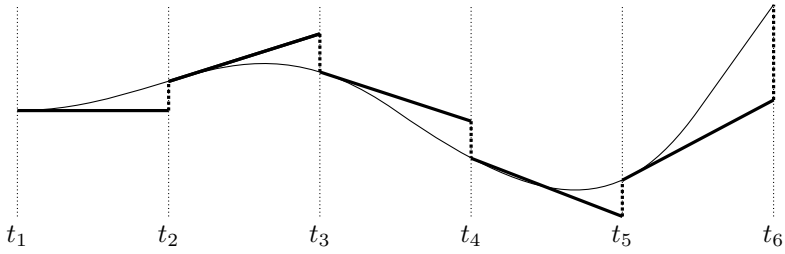
Na riešenie tohoto problému možno využiť *rozšírený Kalmanov filter*, ktorého účelom je práve poradiť si s nelineárnymi rovnicami, popisujúcimi skúmaný systém, ich linearizáciou v okolí časových bodov, v ktorých dochádza k prepočtu nového stavu.

V jednorozmernom prípade je najlepšou lineárnou aproximáciou diferencovateľnej funkcie v niektorom z bodov jej definičného oboru *dotyčnica*, teda priamka so smernicou, rovnou *prvej derivácii* tejto funkcie v našom bode. Jednorozmerná ilustrácia sa nachádza na obrázku 5.4. V prípade vektorových funkcií viacerých premenných je zovšeobecnením prvej derivácie *Jacobiho matica*. Predpokladajme vektorovú funkciu

$$\mathbf{f}(\vec{x}) = (f_1(\vec{x}),\, f_2(\vec{x}), \ldots,\, f_n(\vec{x}))\,. \tag{5.18}$$

Jacobiho maticu, odpovedajúcu tejto funkcii získame ako

$$\mathbb{J}_{\mathbf{f}}(\vec{x}) = \left(\frac{\partial f_i}{\partial x_j}\right),\ i = 1,\, 2, \ldots,\, n,\ j = 1,\, 2, \ldots, \text{počet premenných} \tag{5.19}$$



Obr. 5.4: Linearizácia diferencovateľnej funkcie v bodoch výpočtu nasledujúceho stavu filtru pomocou dotyčníc – ide o jednorozmerný prípad všeobecného algoritmu EKF.

Rozšírený Kalmanov filter (ďalej len EKF)[viii] využíva práve Jacobiho maticu na nájdenie matíc $\mathbb{F}$ a $\mathbb{H}$ štandardného Kalmanovho filtru, použitých pre výpočet predikcie a korekcie v každom kroku. Pri všeobecne nelineárnom systéme, v ktorom je nelineárny ako vzťah medzi dvoma za sebou nasledujúcimi odhadmi stavu, tak vzťah medzi hodnotami veličín z merania a hodnotami stavových veličín, prejdú rovnice (5.6) a (5.8) na

$$\begin{aligned} \hat{x}_k^- &= \mathbf{f}(\hat{x}_{k-1}^+) \\ \hat{x}_k^+ &= \hat{x}_k^- + \mathbb{K}_k(y_k - \mathbf{h}(\hat{x}_k^-)) \end{aligned} \tag{5.20}$$

Funkcia EKF je potom v každom časovom bode, kedy je predikovaný a korigovaný nový stav, v predikčnej časti využiť priamo vektorové funkcie $\mathbf{f}$ (v tomto prípade ju v možno vďaka jej linearite nahradiť maticou $\mathbb{F}$) a $\mathbf{h}$ (kde $h_1(\vec{x}) = \vec{x}_1 = x$, $h_2(\vec{x}) = \vec{x}_2 = y$, $h_3(\vec{x}) = \vec{x}_3 = z$ a posledné tri zložky sú určené rovnicami (5.17))a v korekčnej časti tieto funkcie linearizovať, teda položiť pre účely nasledujúceho kroku filtru

$$\begin{aligned} \mathbb{F}_k &= \mathbb{J}_{\mathbf{f}}(\hat{x}_{k-1}^+) \\ \mathbb{H}_k &= \mathbb{J}_{\mathbf{h}}(\hat{x}_k^-)\,. \end{aligned} \tag{5.21}$$

V ostatnom je algoritmus EKF zhodný s algoritmom 4. V prípade našej úlohy je prechodová funkcia lineárna a matica $\mathbb{F}$ zostáva rovnaká, ako v rovnici 5.13. Potrebujeme teda iba stanoviť Jacobiho maticu výberovej funkcie $\mathbf{h}$, z ktorej vždy pred korekciou odhadu stavu *meraním z GPS* určíme maticu $\mathbb{H}$. Na základe rovníc (5.17) dostávame

$$\begin{array}{ccc} \frac{\partial|\vec{v}|}{\partial v_x} = \frac{v_x}{\sqrt{v_x^2+v_y^2+v_z^2}} & \frac{\partial|\vec{v}|}{\partial v_y} = \frac{v_y}{\sqrt{v_x^2+v_y^2+v_z^2}} & \frac{\partial|\vec{v}|}{\partial v_z} = \frac{v_z}{\sqrt{v_x^2+v_y^2+v_z^2}} \\ \frac{\partial\vartheta}{\partial v_x} = \frac{x\,z}{(x^2+y^2+z^2)\sqrt{x^2+y^2}} & \frac{\partial\vartheta}{\partial v_y} = \frac{y\,z}{(x^2+y^2+z^2)\sqrt{x^2+y^2}} & \frac{\partial\vartheta}{\partial v_z} = -\frac{\sqrt{x^2+y^2}}{x^2+y^2+z^2} \\ \frac{\partial\varphi}{\partial v_x} = -\frac{v_y}{v_x^2+v_y^2} & \frac{\partial\varphi}{\partial v_y} = \frac{v_x}{v_x^2+v_y^2} & \frac{\partial\varphi}{\partial v_z} = 0 \end{array} . \tag{5.22}$$

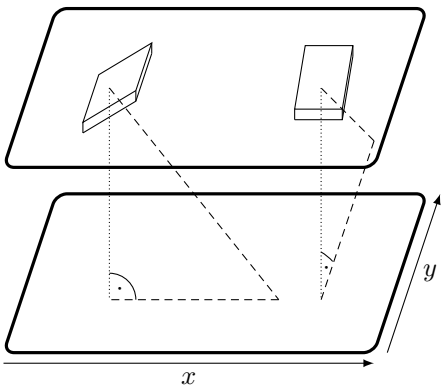
Pretože polohové zložky x, y a z sú na tých rýchlostných nezávislé a ich parciálne derivácie podľa x, y a z sú zjavné, môžeme písať

[viii] Extended Kalman filter.

$$\mathbb{J}_{\mathbf{h}}(\vec{x}) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{v_x}{\sqrt{v_x^2+v_y^2+v_z^2}} & \frac{v_y}{\sqrt{v_x^2+v_y^2+v_z^2}} & \frac{v_z}{\sqrt{v_x^2+v_y^2+v_z^2}} \\ 0 & 0 & 0 & \frac{x\,z}{(x^2+y^2+z^2)\sqrt{x^2+y^2}} & \frac{y\,z}{(x^2+y^2+z^2)\sqrt{x^2+y^2}} & -\frac{\sqrt{x^2+y^2}}{x^2+y^2+z^2} \\ 0 & 0 & 0 & -\frac{v_y}{v_x^2+v_y^2} & \frac{v_x}{v_x^2+v_y^2} & 0 \end{pmatrix}. \tag{5.23}$$

Pre získavanie polohy a rýchosti možno teda využiť takýto postup:

1. Na automobile budeme mať k dispozícii dva radarové merače rýchlosti, usporiadané podľa nasledujúceho obrázku:



   Vrchná rovina tu predstavuje rovinu umiestnenia radarových senzorov a spodná rovina odraznú plochu, teda povrch zeme, ktorý predpokladáme s rovinou umiestnenia radarov rovnobežný. Každý z nich zviera s týmito rovinami elevačný uhol α a ich *vzájomný* uhol je pravý. Pri predpoklade lúčového anténneho zväzku teda na základe rovnice (1.3) bude každý z nich podávať informáciu o rýchlosti v jednom z dvoch vzájomne kolmých smerov x a y[ix]. Rýchlosť v smere vertikálnej osi možno získať z ktoréhokoľvek z nich a preto je výhodné využiť oba a za výsledok brať ich aritmetický priemer. Takto získavame hodnoty vektoru $\vec{y}_r$ a to tak často, ako to dovolí vyhodnocovací softvér radaru, teda v ideálnom prípade raz za $\frac{1024}{44100} \approx 0,0232$ s.

2. Vektory y_r budú korigovať hodnoty odhadov stavov v *každom* časovom bode obnovy stavu. Matica $\mathbb{F}$ bude časovo nemenná a rovná tej, ktorá vystupuje vo vzťahu (5.13), kovariančnú maticu šumu systému $\mathbb{Q}$ zvolíme na základe odporúčaní v [19] a doladíme pri reálnych meraniach. Kovariančnú maticu šumu merania $\mathbb{R}$, ktorá v našom prípade obsahuje jediný prvok, rozptyl merania rýchlosti radarom, opäť najjednoduchšie určíme

[ix] To, či to tak skutočne je, sme overili pozorovaním spektrogramu, vypočítaného z nahrávky pri usporiadaní 2.12, na ktorej je zaznamenaný pohyb „vpred a vzad“ i pohyb „vpravo a vľavo“. Na spektrograme skutočne nebolo viditeľné nič, pokiaľ sa radar pohyboval po priamke *kolmej* na svoju predozadnú rovinu.

experimentálne.[x]. Tým získavame všetky potrebné premenné prvej časti filtrácie.

3. Druhými meraniami, ktoré budeme využívať, sú merania z GPS senzoru, tvorené spomínaným vektorom $\vec{y}_{\mathrm{GPS}}$, ktorého perióda aktualizácie je dlhšia, než perióda aktualizácie vektorov $\vec{y}_r$ z radaru. Jeho prítomnosť bude nejakým spôsobom indikovaná a v prípade, že sa tak stane, bude odhad stavu korigovaný i vektorom $\vec{y}_{\mathrm{GPS}}$. Vzhľadom na jeho tvar musíme voliť EKF, v ktorom matica $\mathbb{F}$ ostáva nezmenená, no matica $\mathbb{H}_k$ je pre daný korekčný krok vypočítaná z posledného dostupného korigovaného odhadu stavu ($\hat{x}_{k-1}^{+}$) podľa vzorca (5.23). Matice $\mathbb{Q}$ a $\mathbb{R}$ určíme na základe aktuálneho stavu GPS, pričom ich určenie pre jedno z podobných usporiadaní Kalmanovho filtru je uvedené v [20].

[x]Je, samozrejme, možné urobiť aspoň odhad jej hodnoty tak, že budeme merať viacero známych konštantných úrovní rýchlosti a rozptyl vypočítame z jeho definície. Takáto metóda je môže byť pre presné určenie pomerne pracná, no i pri malom množstve skúmaných rýchlostí by nám mala poskytnúť približnú hodnotu variancie rýchlosti, ktorá môže byť neskôr doladená na základe výsledkov Kalmanovej filtrácie. Pri meraní konštantnej rýchlosti $v_x = 0,2837$ m/s bol rozptyl stanovený ako $\sigma_m^2 = 0,0016$.

Záver

V tejto práci, naväzujúcej na výskum [1], sme sa venovali niekoľkým prístupom k vyhodnoteniu signálu z radarového senzoru, pričom sme sa snažili o podrobné preskúmanie ich vlastností a selekciu tých najúspešnejších. Tieto sme neskôr využili v návrhu softvéru pre nezávislé radarové zapojenie. K tomuto zapojeniu, obsahujúcemu okrem radarového senzoru a príslušných obvodov i procesor a prípadné doplnkové senzory, by sme v náväznosti na túto prácu chceli smerovať a vytvoriť tak spoľahlivý merač rýchlosti, využiteľný ako samostatne, tak v komplexnom systéme určovania polohy.

Jednotlivé algoritmy signálového spracovania boli zamerané na dosiahnutie čo najväčšej presnosti pri nízkych hodnotách rýchlosti, čo sa čiastočne podarilo vďaka implementácii lokálnej adaptability vyhodnocovacieho systému. Množstvo priestoru na zlepšenie vyhodnocovacích vlastností sa nachádza v zmenšovaní šírky jej pásiem.

Vzniknutý softvér počíta s príchodom signálových vzoriek v časovej následnosti a je schopný určiť z nich rýchlosť v čase iba málo oneskorenom oproti času reálnemu. Samotný softvér je implementovaný v jazyku C++ a vyskúšaný na niekoľkých osobných PC s rôznymi distribúciami GNU/Linux, kde bolo možné okamžite overovať správnosť jeho chodu.

V závere práce sme sa zaoberali možnosťou jeho využitia v systéme určovania polohy a predstavili sme návrh topológie Kalmanovho filtru, ktorý merania zo skúmaného rýchlomeru zúžitkuje na zlepšenie jeho presnosti.

DODATOK A

Popis algoritmu SGDM

Skratka *SGDM* ako taká sa skladá zo skratiek dvoch. *Stochastic gradient descent*, teda jej prvá časť, je názov algoritmu pre iteratívny výpočet hodnôt váh a posunutí pre neuróny siete, ktorý cieli na minimalizáciu odchýliek medzi známymi hodnotami výstupu a „aktuálnymi“ výstupmi siete. Časť *M*, *momentum*, je potom len algoritmom optimalizačným, ktorého úlohou je dosiahnuť uspokojivé výsledky siete na menší počet iterácií *SGD* algoritmu.

A.1 Stochastic gradient descent

Predpokladajme neurónovú sieť, pozostávajúcu z niekoľkých vrstiev, bez ujmy na všeobecnosti s jediným výstupným neurónom. Zaveďme nasledujúcu terminológiu:

γ bude označovať výstup zo siete,

τ bude známym výsledkom, odpovedajúci tomu istému vstupu, ktorého vyhodnotením je γ.

Trénovacia množina $\mathbb{T}$ je množina vektorov, ktorých obrazy poznáme. Pre obraz γ nech je vzorom vektor $\vec{x} \in \mathbb{T}$.

Množina cieľov $\odot$ predstavuje požadované (známe) obrazy vektorov z $\mathbb{T}$.

Ďalej budeme potrebovať zvoliť si *chybovú funkciu*. Táto funkcia má za úlohu hodnotiť, ako veľmi súhlasí výstup γ s cieľom τ. Pri tréningu našich sietí v prostredí *NNT* bola ako chybová funkcia použitá *stredná kvadratická odchýlka* $m(\vec{p})$, definovaná ako

$$m(\vec{p}) = \mathrm{E}((\vec{p} - \vec{t})^2)\,, \tag{A.1}$$

kde

$\vec{p}$ predstavuje istý *odhadovaný* vektor – v našom prípade sú jeho prvky tvorené hodnotami $\gamma_1, \gamma_2, \cdots \gamma_n$, získaných pre niekoľko vstupov $\vec{x}_1$, $\vec{x}_2, \cdots \vec{x}_n$ z trénovacej množiny $\mathbb{T}$. Je potrebné si uvedomiť, že $\vec{p}$ je funkciou vstupných vektorov $\vec{x}$ a váh a posunutí neurónov siete.

$\vec{t} = (t_1, t_2, \cdots t_n)^T$ je vektorom, predstavujúcim obrazy vektorov $\vec{x}_1, \vec{x}_2, \cdots \vec{x}_n \in \mathbb{T}$, ktoré by sme chceli, aby sme dostali na výstupe siete. Vektor $\vec{x}_i$ by teda mal byť sieťou v ideálnom prípade zobrazený na hodnotu t_i.

Posledným prípravným krokom je zvolenie si konštanty $n_{\text{minibatch}}$, n_{break} a ς. $n_{\text{minibatch}}$ predstavuje počet vstupov z množiny $\mathbb{T}$, vstupujúcich do algoritmu v každej iterácii, n_{break} maximálny počet bezprostredne za sebou nasledujúcich iterácií, kedy v každej ďalšej bude hodnota chybovaj funkcie *narastať*, na to, aby algoritmus pokračoval. ς je tzv. *learning rate*, pričom význam tohoto parametra bude jasný z algoritmu samotného. Ten možno teraz popísať nasledovne:

Algoritmus 5 Stochastic gradient descent

procedure SGD($\vec{h}, \vec{p}, \vec{t}$) ▷ Vektor $\vec{h}$ predstavuje všetky hodnoty váh a posunutí všetkých neurónov siete, usporiadané do jedného vektoru. Označme počet prvkov vektoru $\vec{h}$ ako l.
    Náhodne priraď hodnoty prvkov vektoru $\vec{h}$
    while $n \leq n_{\text{break}}$ **do**
        Vyber $n_{\text{minibatch}}$ náhodných vstupných vektorov
        $\vec{t} \leftarrow$ Cieľové obrazy vybraných vektorov
        Nechaj sieť vyhodnotiť vybrané vstupy
        $\vec{\nabla} \leftarrow \vec{0}$
        for $i = 1$ až $n_{\text{minibatch}}$ **do**
            $\vec{\nabla} = \vec{\nabla} + \left(\frac{\partial m}{\partial h_1}\Big|_{\vec{h}}, \frac{\partial m}{\partial h_2}\Big|_{\vec{h}}, \cdots \frac{\partial m}{\partial h_l}\Big|_{\vec{h}} \right)$
        end for
        $\vec{\nabla} \leftarrow \frac{\vec{\nabla}}{n_{\text{minibatch}}}$ ▷ $\vec{\nabla}$ teraz obsahuje „priemerný“ gradient chybovej funkcie v bode $\vec{h}$ v rámci skúmanej náhodne vybranej skupiny vstupných vektorov
        $\vec{h} \leftarrow \vec{h} - \varsigma \vec{\nabla}$ ▷ Všetky váhy a posunutia pozmeníme o *learning rate* v smere odhadovaného najväčšieho poklesu chybovej funkcie
        $n \leftarrow n + 1$
    end while
end procedure

A.1.1 Optimalizácia pomocou momentu

Pokiaľ smeruje gradient chybovej funkcie, počítanej pre všetky dostupné trénovacie dáta, pozdĺž priamky ϖ, je pravdepodobné, že pri jeho aktualizáciách pomocou malých skupiniek trénovacích dát bude aktuálny vektor $\vec{h}_i$ pozdĺž priamky ϖ *oscilovať*. Pre zmenšenie týchto oscilácií a teda i zrýchlenie konvergencie siete sa často používa optimalizácia pomocou *momentu*. Pri nej na výsledku i-tej aktualizácie kontribuujú i výsledky aktualizácií predchádzajúcich. To, do koľko prechádzajúcich aktualizácií má vplyv na aktualizáciu novú, je vyjadrené pomocou parametra ε. V prípade bez použitia momentu je vektor $\vec{h}$ v i-tej aktualizácií rovný

$$h_i = h_{i-1} - \varsigma \vec{\nabla} .$$

Pri použití SGDM bude tento vzťah upravený na

$$h_i = h_{i-1} - \varsigma \vec{\mathcal{G}}_i$$

kde pre $\vec{\mathcal{G}}_i$ platí

$$\vec{\mathcal{G}}_i = \varepsilon \vec{\mathcal{G}}_{i-1} + (1 - \varepsilon) \left(\left. \frac{\partial m}{\partial h_1} \right|_{\vec{h}_{i-1}}, \left. \frac{\partial m}{\partial h_2} \right|_{\vec{h}_{i-1}}, \ldots, \left. \frac{\partial m}{\partial h_l} \right|_{\vec{h}_{i-1}} \right) .$$

Hodnota parametra ε je v MATLABe štandardne zvolená ako $\varepsilon = 0,9$. Pre ani jednu zo sietí, použitých v tejto práci, nebola pri tréningu hodnota ε zmenená.

DODATOK B

Postupy prípravy trénovacích dát

Najskôr rozoberieme použitie priameho postupu, kedy subjektívne hodnotíme obrázky a získavame tak prvky cieľovej množiny $\odot$.

B.1 Ručná príprava, ručný zápis

Najpriamejším prístupom, ktorý bol najmä pri sieťach s binárnym výstupom i efektívny, bolo zobrazovať si vstupné obrázky (v prípade príliš úzkych obrázkov s umelo upravenou šírkou) na istej mierke, na základe ktorej hodnotiteľ ľahšie určí cieľovú hodnotu. Ako príklad uveďme prípad, kedy chceme ako trénovacie vstupy použiť 4 časové body široké podspektrogramy zo spektrogramu nahrávky $\mathcal{N}_2$. Najskôr sme si v prostredí MATLAB nechali vykresliť celý spektrogram (napríklad tak, ako v riadkoch 29 až 35 kódu E.2) a uložili ho do zložky `mother`. Tento sme orezali čo najpresnejšie v ľubovoľnom obrázkovom editore, aby sme sa zbavili artefaktov ako osi a biely okraj, ktorý MATLAB bežne pri ukladaní obrázkov vytvára.

Obrázok sme nazvali `sg_orig.bmp`. Ďalej by sme chceli, aby jeden pixel pôvodného obrázka odpovedal jednému časovému bodu, v ktorom bolo počítané spektrum. To môžeme – i keď iba približne – zariadiť naškálovaním obrázku na veľkosť, odpovedajúcu rozmerom matice pôvodného spektrogramu. Na tento účel sme využili program IMAGEMAGICK [6]. Z príkazového riadku voláme

```
convert sg_orig.bmp -resize 2576x744\! sg_res.bmp
```

a následne sme si vytvorili zložku, do ktorej chceme ukladať 4 px široké podspektrogramy.

```
mkdir ./stripes
```

Opäť pomocou IMAGEMAGICK túto zložku zaplníme:

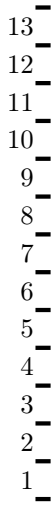
```
convert sg_res.bmp -crop 644x1@ +repage +adjoin ./stripes/stripe_%d.bmp
```

kde číslo $644 = \frac{2576}{4}$ je počet podspektrogramov v pôvodnom obrázku vo vodorovnom smere a 1 ich počet v smere zvislom. Podspektrogramy sme si rozšírili,

aby boli pohodlnejšie viditeľné pri zobrazení na displeji počítača. Na to sme použili opäť ImageMagick. Volaný bash skript vyzeral takto:

```
#! /bin/bash
mkdir ./wide # Creating folder for wide stripes
for i in {0..643}; do
    convert ./stripes/stripe_$i.bmp -resize 40x744\! ./wide/stripe_$i.bmp
done
rm -rf ./stripes
mv ./wide ./stripes
```

Pomocou softvéru našej preferencie sme si vytvorili mierku, ktorá odpovedá rozmedziu, ktoré máme za cieľ hodnotiť. V uvažovanom prípade povedzme, že cheme zadeliť rýchlosť na pospektrogramoch do trinástich kategórií, ktoré možno neskôr samozrejme prepočítať na rozmedzi rýchlostí. Naša mierka teda môže vyzerať napríklad tak, ako je uvedené na obrázku B.1.



Obr. B.1: Mierka na ručné hodnotenie podspektrogramov.

Posledným krokom je spojiť mierku a podspektrogramy, čo zariadime napríklad pomocou takéhoto skriptu:

```
#!/bin/bash
mkdir ./leveled
for i in {0..643}; do
    convert +append ruler.bmp ./stripes/stripe_$i.bmp ./leveled/stripe_$i.bmp
done
rm -rf ./stripes
mv ./leveled ./stripes
```

V zložke `stripes` máme teraz pripravené podspektrogramy i s mierkami a možno ich pohodlne prechádzať jeden po druhom a hodnotiť, do ktorej zo skupín 1 až 13 údaj na spektrograme patrí. Výsledky sme v našom prípade zapisovali do súboru, ktorý sme neskôr upravili na požadovaný vstup siete pomocou editoru Vim a načítali do Matlabu, kde sa odohrával samotný tréning.

Zložka `mother` s potrebnými obrázkami a skriptami tvorí prílohu `prepare_for_classification.tar.gz`.

B.2 Príprava i zápis v MATLABe

Alternatívou, ktorou sme sa pokúšali prispieť k zvýšeniu pohodlia ručného hodnotenia z časti B.1, bola MATLAB funkcia, ktorá zobrazila spektrogram, z ktorého podspektrogramy vyhodnocujeme, na pozadí a na ňom postupne vyznačovala úseky, ktoré mal užívateľ hodnotiť. Po každom vyznačení úseku funkcia čaká na vstup vo forme čísla, indikujúceho príslušnosť daného úseku do jednej z užívateľom definovaných kategórií a po jeho zadaní posunie vyznačenie na susedný úsek.

Hoci cieľom jej vytvorenia bolo urýchlenie celého hodnotiaceho procesu, s úspechom sa nestretla a jej použitie sa ukázalo ako nepraktické. Jej implementácia je uvedená v kóde E.6.

Nasledovať budú dva prístupy z „opačného konca," kedy pre známe prvky z cieľovej množiny $\odot$ vytvárame odpovedajúce prvky množiny $\mathbb{T}$.

B.3 Patchwork generátor

Prvý z použitých generátorov vytvára obrázky veľkosti 4 px široké obrázky tak, že na seba vrství typické súčasti každého podspektrogramu, ktorými sú

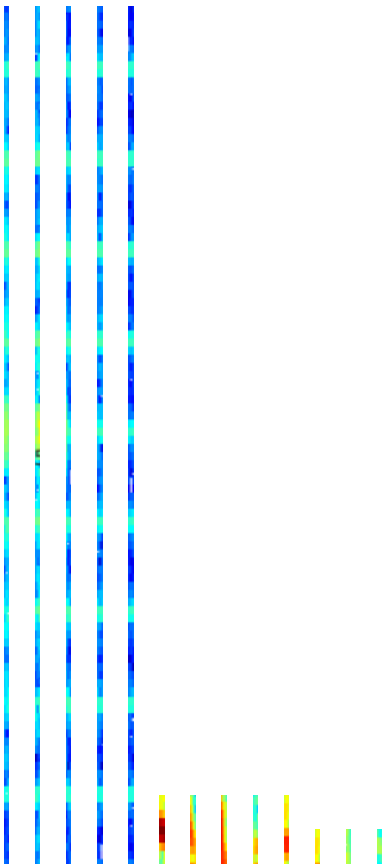
Pozadie, ktoré odpovedá časti skutočného spektrogramu, v ktorej je rýchlosť nulová konkrétne pozadie je vyberané náhodne spomedzi piatich takýchto vzoriek.

Šum, ktorý je umiestnený náhodne na 5 pozícií na vybranom pozadí.

Maximá spektrogramu, umiestnené symetricky podľa horizontálnej osi pozadia. Maximum je vyberané náhodne z piatich vzoriek a umiestňované ako *nepriehľadné* na tú stranu spektra, ktorej prísluší a s istou priehľadnosťou na opačnú stranu spektra, čím sa snažíme simulovať nedokonalosti zmiešavačov a iných obvodov, spôsobujúce artefakty tohoto typu na takmer každom zo spektrogramov.

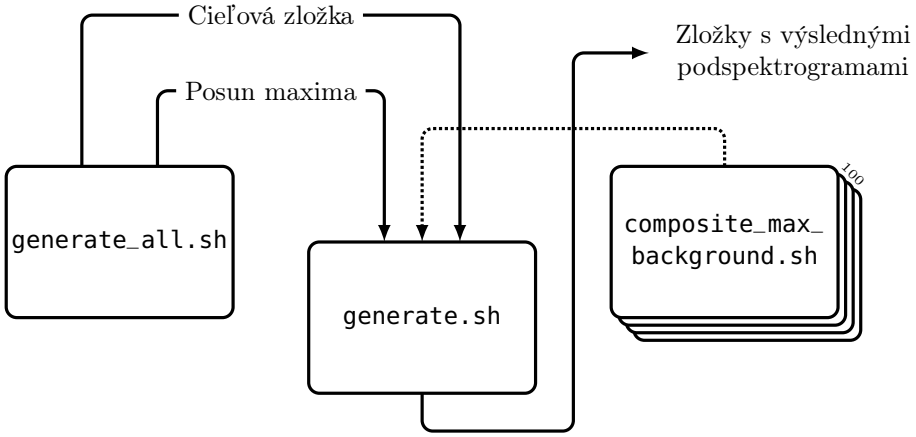
Ozveny, ktoré sú umiestňované na pozície druhej harmonickej maximálnej frekvencie (nepriehľadne) a druhej harmonickej maximálnej frekvencie so záporným znamienkom (s priehľadnosťou).

Jednotlivé zložky generovaných podspektrogramov sa nachádzajú na obrázku B.2.



Obr. B.2: Jednotlivé zložky pre generovanie pseudodát – pozadia (a – e), maximá (f – j), „ozveny“ (k – l) a šum (m)

Patchwork generátor ako taký je tvorený skriptami a obrázkami z prílohy `patchwork_generator.tar.gz`, a jeho schéma je na obrázku B.3. Skript `generate.sh` očakáva dva vstupy – tým prvým je posun začiatku maxima od horného okraja obrázku a druhým umiestnenie cieľového súboru. Tieto dva vstupy mu zabezpečuje skript `generate_all.sh`, mení polohu maxima s istým zadaným krokom a cieľovú zložku taktiežmení odpovedajúcim spôsobom. Skript `generate.sh` si potom stokrát (tento počet samozrejme možno jednoducho nastaviť) volá skript `composite_max_background.sh`, ktorý vytvorí koláž o žiadaných parametroch a uloží ju do tej správnej zložky. Pred trénin-gom zo skriptu E.7 je navyše ešte nutné volať skript `mapping_folders.sh`, ktorý zmení názvy zložiek s generovanými dátami na také, aby ich názov číselne odpovedal desaťnásobku rýchlosti na podspektrogramoch v nich.



Obr. B.3: Schéma patchwork generátoru.

Generátor je prílohou tejto práce s názvom `patchwork_generator.tar.gz`. Tréning siete na takýchto dátach by mohol vyzerať napríklad tak, ako je uvedené v kóde E.7.

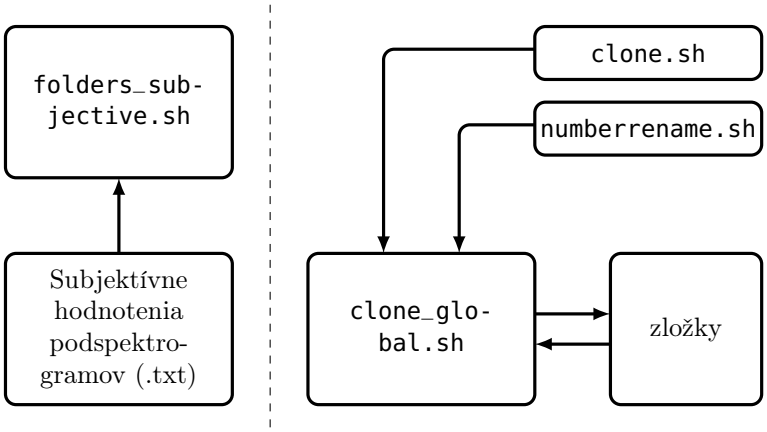
B.4 Priemerovací generátor

Implementácia priemerovacieho generátoru nasleduje algoritmus 6.

Algoritmus 6 Priemerovací generátor

procedure AvgGen($\vec{F}$, M) ▷ Prvky vektoru $\vec{F}$ sú zložky, z ktorých každá obsahuje niekoľko (viac než jeden) podspektrogramov a názov každej z ktorých je desaťnásobkom hodnoty rýchlosti, ktorá je zachytená na podspektrogramoch v nej. M je cieľová hodnota počtu podspektrogramov v jednej zložke. Predpokladáme, že podspektrogramy v zložke sú pomenované p_1, p_2, …, p_k.
    for $i \leftarrow 1, 2, \ldots, length(\vec{F})$ **do**
        Operuj v zložke F_i
        $n \leftarrow$ Počet podspektrogramov v zložke
        while $n < M$ **do**
            $r_1 \leftarrow$ Náhodné číslo od 1 do n
            $r_2 \leftarrow$ Náhodné číslo od 1 do n
            $p_n \leftarrow RGBmean(p_{r_1}, p_{r_2})$ ▷ Funkcia *RGBmean* vracia obrázok, ktorý je priemerom RGB hodnôt dvoch vstupných obrázkov o rovnakých rozmeroch.
            Ulož p_n do zložky F_i
            $n \leftarrow n + 1$
            Premenuj podspektrogramy v ľubovoľnom poradí na $p_1, p_2, \ldots, p_n$.
        end while
    end for
end procedure

Ona samotná tvorí prílohu `priemgen_final.tar.gz` tejto práce. Schéma generátoru sa nachádza na obrázku B.4.



Obr. B.4: Schéma priemerovacieho generátoru. V prvej časti sú na základe súboru, v ktorom sa nachádzajú subjektívne vyhodnotené rýchlosti, roztriedia jednotlivé podspektrogramy do zložiek. Následne (na obrázku nie je naznačené) je nutné volať skript `folders_subjective.sh`, ktorý zmení názvy zložiek tak, aby boli desaťnásobkom rýchlosti na podspektrogramoch v nej. V hlavnej časti voláme skript `clone_global.sh`, ktorý volá skripty `slone.sh` a `numberrename.sh` a zabezpečuje tým vytváranie požadovaného počtu podspektrogramov v každej zo zložiek.

DODATOK C

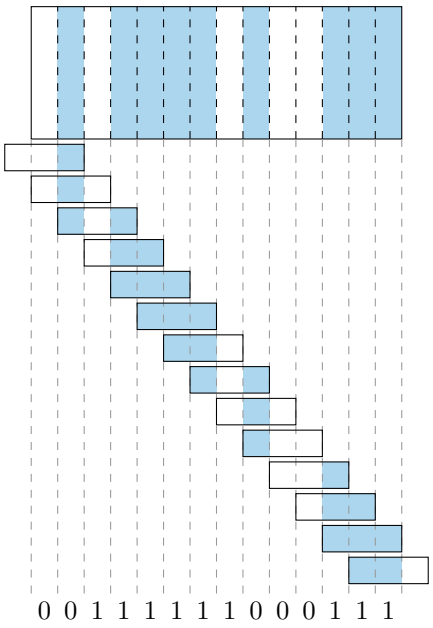
Modusový filter

Modus je definovaný pre vektor $\vec{v}$ ako *jeho najfrekventovanejšia zložka*. Pokiaľ je výsledkom najčastejšej hodnoty *niekoľko* hodnôt, je nutné toto zobrazenie dodefinovať. Predpokladajme, že filter má za úlohu filtrovať vektor hodnôt -1, 0 a 1, ktoré indikujú smer pohybu *vzad*, *stav bez pohybu* a *vpred*. Uveďme stavy, kedy sa v našom vektore nachádzajú dve najčastejšie hodnoty a pozrime sa na ich význam.

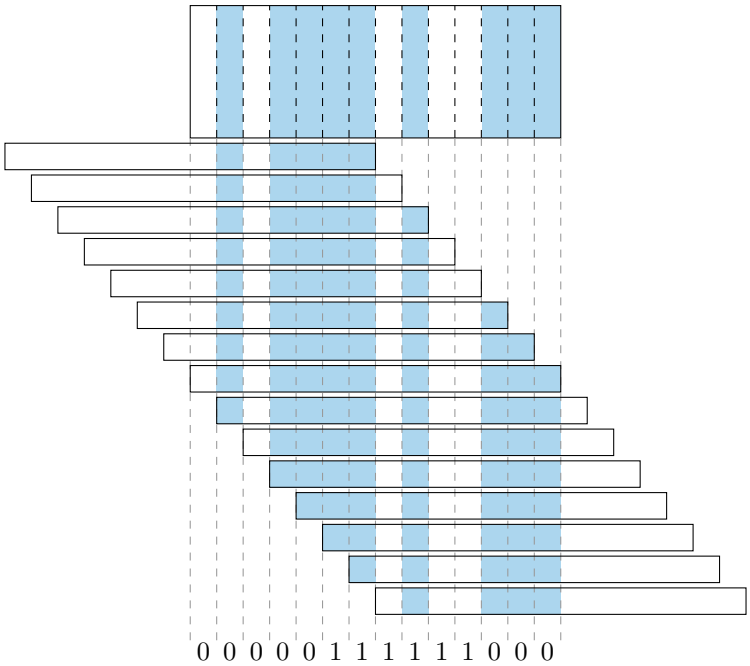
Najčastejšie hodnoty	**Význam**
$0,\ 1$	Hranica úseku pohybu
$0,\ -1$	Hranica úseku pohybu
$1,\ -1$	Chyba

Stavy, ktoré nie sú chybou (nekonečne rýchlu zmenu zo stavu pohybu vpred do stavy pohybu vzad nepredpokladáme) možno dodefinovať priradením ktorejkoľvek z dvoch hodnôt do výsledku. Prienikom hodnôt našich dvoch stavov je hodnota 0 – definujme teda modus v prípade, že sa vo vektore v nachádzajú dve najčastejšie hodnoty, ako *nulu*. Kľúčové pre dobré výsledky filtrácie je zvoliť vhodnú dĺžku okna. Správanie modusového filtru pri príliš krátkom, vhodne zvolenom a príliš dlhom okne pri filtrácii vektoru niekoľkých hodnôt, nadobúdajúcich stavy *pohyb* (1) alebo *nepohyb* (0)[i] sa nachádza na obrázkoch C.1, C.2, C.3.

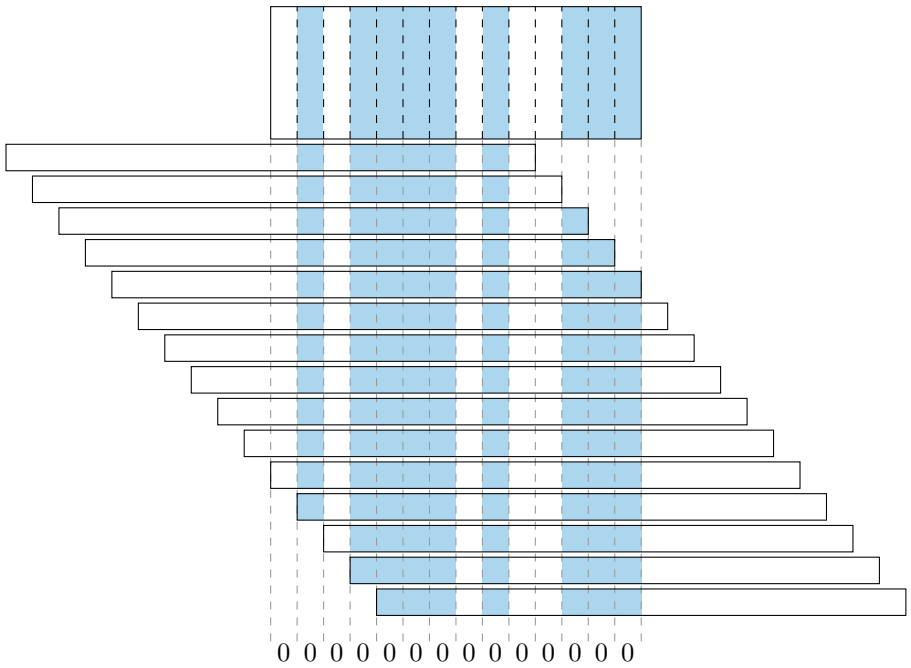
[i] Správanie sa filtru v hraničnom úseku oblastí pohybu vpred a vzad si na jej základe možno jednoducho domyslieť.



Obr. C.1: Príliš krátke výrezové okno spôsobí nedostatočné vyhladenie.



Obr. C.2: Vhodne zvolená dĺžka okna. Cenou za dobré vyhladenie je posunutie začiatku a konca radu hodnôt 1.



Obr. C.3: Príliš dlhé okno spôsobí neakceptovateľnú stratu informácie.

Algoritmus, ktorý sme použili pre implementáciu takéhoto filtru v tejto práci i finálnom programe, je spísaný ako algoritmus 7.

Implementácia v MATLABe využila zabudovanú funkciu `mode`, tá v C++ postupovala podľa algoritmu 7 – pri filtrácii polí, nadobúdajúcich tri stavy (−1, 0, 1), v nezmenenej podobe, pri filtrácii polí dvojstavových v odpovedajúcej úprave.

Algoritmus 7 Modusový kĺzavý filter

1: **procedure** ModeFilter(b, $\vec{w}$, Σ_{-1}, Σ_0, Σ_1) ▷ b je novou hodnotou, vstupujúcou do filtru, $\vec{w}$ zas okno filtru. Σ_{-1}, Σ_0 a Σ_1 sú počty hodnôt −1, 0 a 1 v okne $\vec{w}$.
2: **if** $b = 1$ **then**
3: $\Sigma_1 \leftarrow \Sigma_1 + 1$
4: **else if** $b = 0$ **then**
5: $\Sigma_0 \leftarrow \Sigma_0 + 1$
6: **else**
7: $\Sigma_{-1} \leftarrow \Sigma_{-1} + 1$
8: **end if**
9: **if** najstarší bit v okne = 1 **then**
10: $\Sigma_1 \leftarrow \Sigma_1 - 1$
11: **else if** najstarší bit v okne = 0 **then**
12: $\Sigma_0 \leftarrow \Sigma_0 - 1$
13: **else**
14: $\Sigma_{-1} \leftarrow \Sigma_{-1} - 1$
15: **end if**

```
16:         if aspoň dve z hodnôt Σ_{-1}, Σ_0, Σ_1 sú rovnaké then
17:             return 0
18:         end if
19:         if Σ_{-1} je najväčší zo súčtov then
20:             return −1
21:         else if Σ_0 je najväčší zo súčtov then
22:             return 0
23:         elseΣ_1 je najväčší zo súčtov
24:             return 1
25:         end if
26: end procedure
```

DODATOK D

Tréning sietí na lokálnu adaptabilitu prvého spektrálneho momentu

Pre zabezpečenie spoľahlivého priradenia aktuálneho spektra (podspektrogramu) k pásmu rýchlostí sme využili dve samostatné neurónové siete so zhodnou topológiou, no inými trénovacími množinami. V tejto časti si predstavíme najskôr ich spoločnú topológiu a potom konkrétne trénovacie dáta, ktré pre ne boli použité.

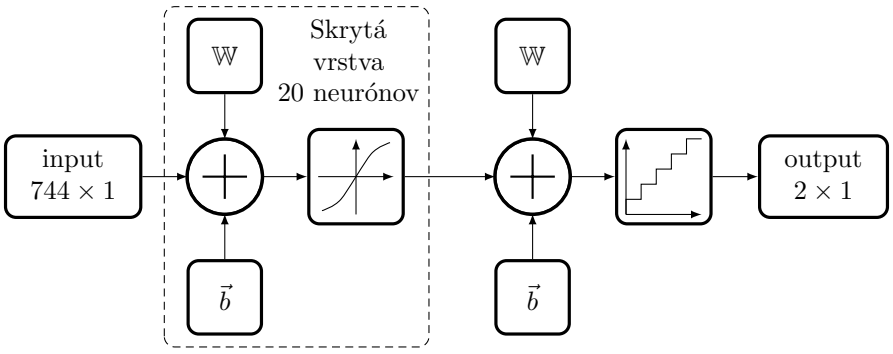
D.1 Topológia sietí

Od použitých sietí očakávame, že budú mať vstup vo forme niekoľkoprvkového spektra, pokrývajúceho dohodnutý rozsah frekvencií (rýchlostí v_x) a ich výstup bude rozhodovať o príslušnosti vstupu do jednej z dvoch skupín – rozmedzia rýchlosti *pod* istou hranicou a *nad* ňou.

Vstupnú vrstvu teda tvorí 744[i] neurónov s *normalizačnou* prechodovou funkciou, ktorá má za úlohu normalizovať vstupné hodnoty do rozmedzia -1 až 1. Nasleduje prvá a jediná vrstva *skrytá*, obsahujúca 30 neurónov s *hyperbolickým tangensom* ako prechodovou funkciou, pričom prepojenie existuje medzi každým neurónom vstupnej a každým neurónom skrytej vrstvy.

Ako výstupnú vrstvu sme zvolili dva neuróny, podporené prechodovou funkciou `softmax` (viď časť 3.1, vzťah (3.2)) namiesto jediného neurónu s binárnym výstupom, ktorý má sám o sebe dostatočný počet stavov na kódovanie nášeho výstupu, no výsledky s takým usporiadaním boli slabšie. Obrázok D.1 zobrazuje vyššiepopísanú topológiu; Prechodové funkcie sú v ňom znázornené iba schématicky.

[i] Zdôvodnenie tohoto počtu je dané našou konvenciou ohľadom vyhodnocovaného rozsahu rýchlostí a parametrami výpočtu spektra.



Obr. D.1: Schéma sietí, použitých pre zabezpečenie lokálnej adaptability vyhodnocovania absolútnej hodnoty rýchlosti.

D.2 Tréning siete pre zistenie príslušnosti spektra k najnižším rýchlostiam

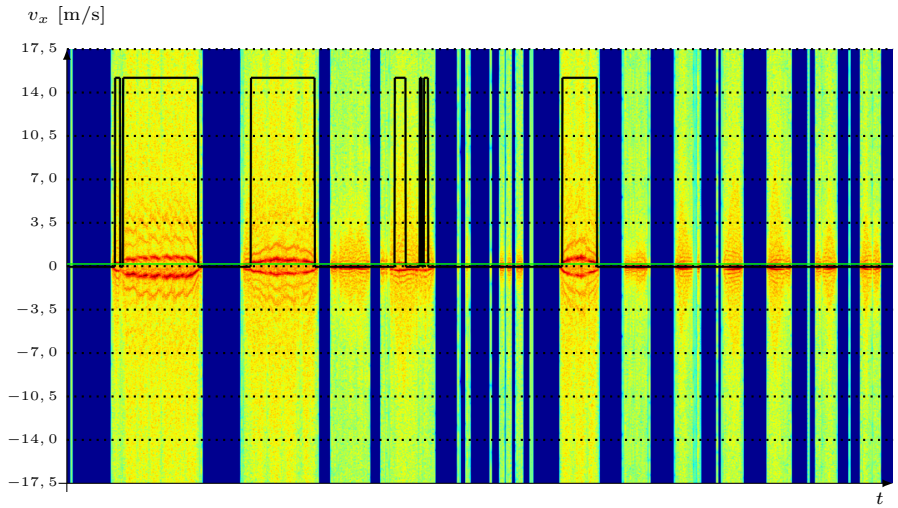
Ako prvý popíšeme tréning a výsledky siete, ktorá zisťuje príslušnosť vstupu (absolútnej hodnoty rýchlosti na ňom) k rozmedziu rýchlostí $\langle 0;\ 0,2367\rangle$ m/s. Vzhľadom na binárny výstup bolo hodnotenie trénovacích podspektrogramov rýchle a pohodlné i bez použitia pomocných skriptov – nechali sme si jednoducho zobraziť spektrogram, na ňom vyznačili úroveň, ktorá rozhoduje o hodnote výstupu a značili sme si indexy stĺpcov spektrogramu, v ktorých nastávajú prechody cez túto úroveň. Ako trénovacie dáta sme nakoniec použili vybrané časti z nahrávok

$$\begin{array}{ccc}\mathcal{N}_3 & \mathcal{N}_5 & \mathcal{N}_7\\ \mathcal{N}_4 & \mathcal{N}_6 & \mathcal{N}_8.\\ \mathcal{N}_2 & & \end{array}$$

Vybrané spektrogramy sme ohodnotili a „naukladali“ za seba do jednej matice. Túto maticu sme následne pridali za posledný vektor ešte raz, preklopenú podľa horizontálnej osi, aby sme získali väčšie množstvo trénovacích dát i na opačnej strane smeru rýchlosti. Tréningovú množinu potom tvoria dve takéto matice, uložené jedna za druhú[ii]. Textový súbor s hodnotami cieľov bol načítaný do MATLABu a prebehol tréning. Použitý skript je uvedený ako kód E.18.

Voľba trénovacích nahrávok prebehla prirodzeným vývinom, kedy sme natrénovali sieť, pozreli sa na výsledky a na ich základe sme ďalej pridávali trénovacie dáta, týkajúce sa takého charakteru rýchlosti na podspektrogramoch, v ktorých si doterajšia sieť nevedela dobre poradiť. Výsledok siete v konečnom stave, trénovanej priamo skriptom E.18, týkajúci sa nahrávky $\mathcal{N}_9$, možno zrieť na obrázku D.2.

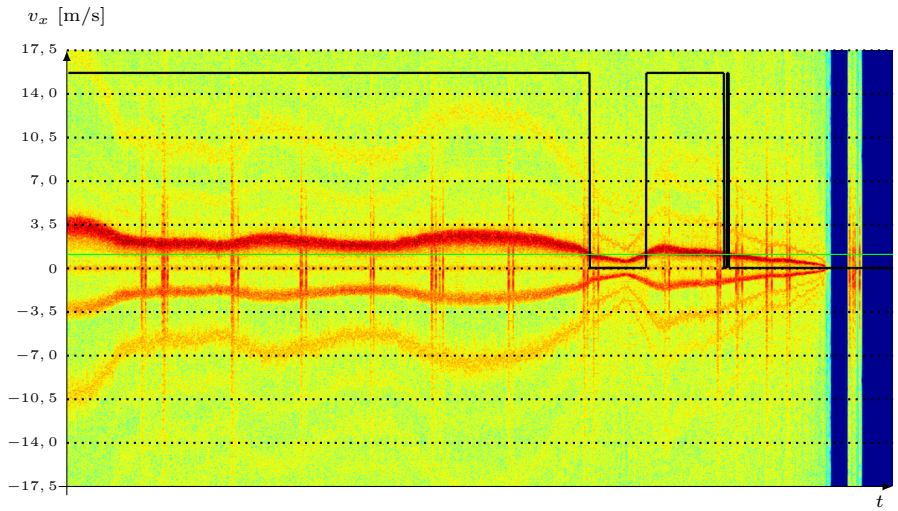
[ii] Tento popis je určený skôr na predstavu o vytváraní trénovacej množiny – ako je známe, v *množine* na poradí prvkov nezáleží.



Obr. D.2: Výsledky siete, priradzujúcej vstupné spektrá do rozmedzia rýchlostí 0 až 0, 2367 m/s. Vyššia úroveň čiernej čiary indikuje rýchlosť nad hranicou (zelená čiara), jej nulová úroveň rýchlosť pod hranicou.

D.3 Tréning siete pre zistenie príslušnosti spektra k rýchlostiam do 1, 0832 m/s

Spôsobom analogickým s tým, v časti D.2 sme vytvorili sieť pre rozoznávanie príslušnosti absolútnych hodnôt rýchlostí vstupov k intervalu $\langle 0;\ 1, 0832\rangle$ m/s. Tréning prebehol na častiach nahrávok



Obr. D.3: Výsledky siete, priradzujúcej vstupné spektrá do rozmedzia rýchlostí 0 až 1, 0832 m/s.

$$\begin{array}{lll} \mathcal{N}_8 & \mathcal{N}_5 & \mathcal{N}_{10} \\ \mathcal{N}_4 & \mathcal{N}_6 & \mathcal{N}_{11}. \\ \mathcal{N}_2 & & \end{array}$$

Skript, obsahujúci príkazy pre načítanie potrebných dát a tréning siete je uvedený v kóde E.19. Výsledky takto natrénovanej siete na dátach $\mathcal{N}_9$ sú zobrazené na obrázku D.3.

DODATOK E

Skripty

E.1 Matlab

```
recObj = audiorecorder(44100, 8, 2);
name = input('Name: ','s');
record_time = input('Record time: ','s');
disp('Start.'); beep;
recordblocking(recObj, str2num(record_time)); % Recording for record_time seconds
disp('Stop.'); beep;

y = getaudiodata(recObj);

audiowrite(['/path/to/folder' name '.wav'], y, 44100);
```

Kód E.1: Skript, použitý na nahrávanie wav súborov ako výstupu z radaru.

```
[y, fs] = audioread('<path_to_wav_file/file.wav>');

vx_limit_index = 372; % Maximal relevant index
nfft = 8192; % Window width
wind = hamming(nfft); % Windowing function
noverlap = (7/8) * nfft; % Negative overlap
num_time_pts = 1 + floor((numel(signal) - nfft) / (nfft * 1/8)); % Number of time points
     of spectrogram with given parameters, present in given signal

l_part = y(:,1); % In-phase component
r_part = y(:,2); % Quadrature component
signal = l_part + 1i * r_part; % Full complex signal

frames = nan(num_time_pts, nfft);
for n = 1:num_time_pts % Filling spectrogram matrix with signal "cuts"
    frames(n, :) = signal(((n-1) * (nfft - noverlap) + 1):((n-1) * (nfft - noverlap) +
         nfft));
end
frames = frames .* repmat(hamming(nfft)', num_time_pts, 1); % Applying Hamming window
p = (1 / (2 * pi * nfft)) * (abs(fft(frames ,nfft, 2)) .^ 2)'; % Power spectral density

p_resh = [p((nfft - vx_limit_index + 2):end, :); p(1, :); p(1:vx_limit_index, :)]; %
     Reshaped power spectral density
p_resh_db = 10 * log10(p_resh); % Reshaped power spectral density in decibels
[~, p_resh_db_max_ind] = max(p_resh_db); % Indices of maxima

f = 0:(fs/nfft):(fs-fs/nfft); % Frequency vector
alpha = pi / 4; % Elevation angle
vx = (3/241.5 * f)/(2 * cos(alpha)); % velocity in forward direction

```

```
%% Visualizing
figure
pc = pcolor(p_resh_db);
set(pc, 'linestyle', 'none');
colormap(jet);
xlim([0, num_time_pts]);
ylim([0, 2 * vx_limit_index]);
view(2); hold on;
plot(p_resh_db_max_ind, 'k', 'LineWidth', 2);
```

Kód E.2: Vyhodnotenie rýchlosti na preskupenom spektrograme pomocou hľadania jeho maxím.

```
...
p_resh = [p((nfft - vx_limit_index + 2):end, :); p(1, :); p(1:vx_limit_index, :)]; %
    Reshaped power spectral density
p_resh_db = 10 * log10(p_resh); % Reshaped power spectral density in decibels
[~, p_resh_db_max_ind] = max(p_resh_db); % Indices of maxima

p_resh_db_max_ind(p_resh_db_max_ind == 1) = 372;

f = 0:(fs/nfft):(fs-fs/nfft); % Frequency vector
alpha = pi / 4; % Elevation angle
vx = (3/241.5 * f)/(2 * cos(alpha)); % velocity in forward direction
...
```

Kód E.3: Vyhodnotenie rýchlosti na preskupenom spektrograme pomocou hľadania jeho maxím.

```
...
p_resh = [p((nfft - vx_limit_index + 2):end, :); p(1, :); p(1:vx_limit_index, :)]; %
    Reshaped power spectral density
p_resh_db = 10 * log10(p_resh); % Reshaped power spectral density in decibels
p_resh_db(1:372, :) = p_resh_db(1:372, :) + flipud(p_resh_db(373:744, :));
[~, p_resh_db_max_ind] = max(p_resh_db(373:744, :)); % Indices of maxima
p_resh_db_max_ind = p_resh_db_max_ind + 372;
p_resh_db(1:372, :) = p_resh_db(1:372, :) - flipud(p_resh_db(373:744, :));

f = 0:(fs/nfft):(fs-fs/nfft); % Frequency vector
alpha = pi / 4; % Elevation angle
vx = (3/241.5 * f)/(2 * cos(alpha)); % velocity in forward direction
...
```

Kód E.4: Vyhodnotenie absolútnej hodnoty rýchlosti pomocou maxím spektrogramu. Výsledok je na obrázku 3.18.

```
...
p_resh_db(1:372, :) = p_resh_db(1:372, :) - flipud(p_resh_db(373:744, :));
p_resh_db_max_ind = movmean(p_resh_db_max_ind, [6 0]);

f = 0:(fs/nfft):(fs-fs/nfft); % Frequency vector
alpha = pi / 4; % Elevation angle
vx = (3/241.5 * f)/(2 * cos(alpha)); % velocity in forward direction
...
```

Kód E.5: Filtrácia hodnôt z prechádzajúceho kódu kĺzavým priemerom. Výsledok vidno na obrázku 3.19.

```
function out = subjective_classificator(spectrogram_path)

width = 2576; % Width of input image (spectrogram)
const = 2576 / 4; % Number of time intervals that image will be cut into
factor = width/const; % Number of pixels in each of time intervals

%% Opening image, setting axes.
spectrogram = imread(spectrogram_path);
RI = imref2d(size(spectrogram));
```

```
RI.XWorldLimits = [0 15]; % 15 is more or less random number - it is set as it is only
     because of good aspect ratio of resulting view.
RI.YWorldLimits = [-2.5 2.5]; % -v axis

imshow(spectrogram, RI);
hold on;

rect = rectangle('position', [0 -2.5 15/const 5], 'edgecolor', 'm');

out = nan(1, floor(const));

for m = 1:floor(const)
   out(m) = input('Subjective velocity: ');
   set(rect, 'position', [m * 15 / const, -2.5, 15/const, 5])
end
tail = input('Subjective velocity (tail): ');

out = repmat(out, [factor, 1]);
out = [out(:)' repmat(tail, [1 fix((const-floor(const))*factor)])];

end
```

KÓD E.6: Funkcia na pomoc pri subjektívnom vyhodnocovaní hodnôt rýchlosti (alebo iných veličín) na podspektrogramoch. Funkcia berie ako vstupný argument cestu k `.bmp` obrázku celého spektrogramu, ktorý implicitne predpokladá široký 2576 px. Implicitným predpokladom je tiež vyhodnocovanie 4 px širokých podspektrogramov. Spôsob zmeny týchto parametrov by mal byť z použitého kódu zrejmý

```
%% Loading image data
testdata_path = '<path_to_testing_input_stripes>'; % Path leading to folder with input
     data
trainingdata_path = '<path_to_training_input_stripes>'; % Path leading to folder with
     training pictures.
training_data = imageDatastore(trainingdata_path, 'IncludeSubfolders',true,'LabelSource','
     foldernames'); % Image datastore declaration
test_data = imageDatastore(testdata_path);
test_data.Files = sort_nat(test_data.Files); % Sorting files in testing directory in
     numerical order. Requires sort_nat.m function.

%% Specifying layers of CNN
layers = [imageInputLayer([756 4 3])
          convolution2dLayer([1,2], 30)
          reluLayer % Rectified linear Unit layer - finite step response for every input
               geq 0
          maxPooling2dLayer([2,1])
          fullyConnectedLayer(23) % In this layer software multiplies inputs by weights
               and adds biases
          softmaxLayer
          classificationLayer()];

%% Specifying training options
options = trainingOptions('sgdm',... % Stochastic gradient descent with momentum
          'MaxEpochs', 10,... % Maximum nuber of training options
          'InitialLearnRate',.0001); % Initial learning rate (eta)

%% Train network using picture (spectrogram) data
net100 = trainNetwork(training_data, layers, options);

%% Classify rest of pictures and compute the accuracy
classify_test = classify(net100, test_data);

%% Converting categorical array into double array
classify_test_noncategorical = .1*str2double(cellstr(classify_test));

%% Computing absolute values of classify_test_noncategorical
classify_test_noncategorical_abs = abs(classify_test_noncategorical);

%% Viewing results
spectrogram_back = imread('clovek_tam_spat_rozdvojka_resized.bmp'); % Loading testing
     spectrogram image
```

```
figure
imshow(spectrogram_back);
hold on;
plot(linspace(0,1284,321), classify_test_noncategorical*756/5+756/2, 'LineWidth', 3, '
    color', 'k'); % Computed values
plot(linspace(0,1284,321), classify_test_noncategorical_abs*756/5+756/2, 'LineWidth', 3, '
    color', 'm'); % Computed values
```

Kód E.7: Tréning siete na dátach z patchwork generátoru a zobrazenie výsledkov. Skript pracuje s nahrávkou $\mathcal{N}_a$. Zložkou s trénovacími obrázkami je myslená zložka, ktorú dostávame ako výstup patchwork generátoru pri jeho volaní s krokom 30. Bližšie informácie možno nájsť v súbore README v archíve `patchwork_generator.tar.gz`, ktorý obsahuje skripty tohoto generátoru a ktorý je prílohou tejto práce.

```
%% Loading image data
trainingdata_path = '<path>'; % Path leading to folder with training pictures
training_data = imageDatastore(trainingdata_path, 'IncludeSubfolders',true,'LabelSource','
    foldernames'); % Image datastore declaration
testdata_path = '<path>'; % Path leading to folder with training data ("stripes").
test_data = imageDatastore(testdata_path);
test_data.Files = sort_nat(test_data.Files); % Sorting files in testing directory in
    numerical order. Requires sort_nat function

%% Specifying layers of CNN
layers = [imageInputLayer([756 2 3])
          convolution2dLayer([1,2], 25)
          reluLayer % Rectified linear Unit layer - finite step response for every input
              geq 0
          maxPooling2dLayer([2,1])
          fullyConnectedLayer(23) % In this layer software multiplies inputs by weights
              and adds biases
          softmaxLayer
          classificationLayer()];

%% Specifying training options
options = trainingOptions('sgdm',... % Stochastic gradient descent with momentum
          'MaxEpochs', 10,... % Maximum nuber of training options
          'InitialLearnRate',.0001); % Initial learning rate (eta)

%% Train network using picture (spectrogram) data
net100 = trainNetwork(training_data, layers, options);

%% Classify rest of pictures and compute the accuracy
classify_test = classify(net100, test_data);

%% Converting categorical array into double array
classify_test_noncategorical = .1*str2double(cellstr(classify_test));
```

Kód E.8: Vytvorenie štruktúry siete, ktorá má rozoznávať rýchlosť na 2 časové body širokých podspektrogramoch, jej tréning na dátach z priemerovacieho generátoru (B.4) a vyhodnotenie ponúknutých vstupov.

```
...
spectrogram_back = imread ('<image.bmp>'); % Loading testing spectrogram
ctn = classify_test_noncategorical;

for m = 3:(numel(ctn)-2)
    if (abs(mean([ctn(m-2), ctn(m-1), ctn(m+1), ctn(m+2)]) + ctn(m)) < .1)
        ctn(m) = -ctn(m);
    end
end

result = -1*ctn*756/6+756/2;
imshow (spectrogram_back); % Viewing it
hold on ; % We 'd like to plot into it
plot (linspace (0 ,2576 ,1288) , result, 'LineWidth', 3,'Color', 'k');
```

KÓD E.9: Implementácia päťbodového kĺzavého priemeru na vyhladenie skokov vo výsledkoch siete z kódu E.8.

```
%=============READING AUDIO DATA==================
[y,fs] = audioread('C:\Users\MO\Documents\bp\meranie_strahov\meranie10_rychlo50.wav');

l_part = y(:,1); % In-phase component
r_part = y(:,2); % Quadrature component
signal = l_part + 1i * r_part; % Full complex signal

%============PARAMETERS OF SPECTROGRAM===========
nfft = 8192; % Window width
wind = hamming(nfft); % Windowing function
noverlap = (7 / 8) * nfft;
num_time_pts = 1 + floor((numel(signal) - nfft) / (nfft * 1/8)); % For 60s recordings
     2576, for 30s 1284.

% %==========COMPUTING SPECTROGRAM VALUES==========
% [s, f, t, p] = spectrogram(signal, wind, noverlap, nfft, fs, 'yaxis');

frames = nan(num_time_pts, nfft);
for m = 1:num_time_pts
    frames(m, :) = signal(((m-1) * (nfft - noverlap) + 1):((m-1) * (nfft - noverlap) +
         nfft));
end
frames = frames .* repmat(hamming(nfft)', num_time_pts, 1);
p = (1 / (2 * pi * nfft)) * (abs(fft(frames, nfft, 2)) .^ 2)';

f = linspace(0, fs-(fs/nfft), nfft);

%===============PARAMETERS OF RADAR==============
alpha = pi / 4; % Elevation angle
vx = (3/241.5 * f)/(2*cos(alpha)); % velocity in forward direction

%===========SELECTING USEFUL PART OF SG==========
vx_limit_index = 372; % 2kHz, which is the maximal doppler frequency, we'd like to work
     with is on index 372 of 'f' vector.
spectrogram_pts = [p((nfft-vx_limit_index+2):end, :); p(1,:); p(1:vx_limit_index, :)];

%===============SPECTRAL CENTROID================
sc = nan(1,num_time_pts);
for m = 1:num_time_pts
    act = p(1:vx_limit_index, m)';
    positions = 1:vx_limit_index;
    sc(m) = (sum(act.*positions)/sum(act));
end

%====================RESULTS=====================
figure
pc = pcolor(10 * log10(spectrogram_pts));
set(pc, 'linestyle', 'none');
colormap(jet);
xlim([0, 2576]);
ylim([0, 2 * vx_limit_index]);
view(2); hold on;
plot((sc + vx_limit_index), 'k', 'LineWidth', 3);
```

KÓD E.10: Vyhodnotenie rýchlosti v_x na nahrávke pomocou prvého spektrálneho momentu, počítaného z celého skúmaného rozhasu rýchlostí.

```
%% Declaration of aliases. p01, p02 and p04 are matrices of PSD of recordings Nb, Nc and
     Nd, respectively. They must be saved in workspace at the time this script is being
     called.
m1 = [flipud(p01((nfft/2+1):end,:)), flipud(p03((nfft/2+1):end,:)), flipud(p04((nfft/2+1):
     end, 1:641))];
m1 = m1(1:50, :);
m2 = [p01(1:nfft/2, :), p03(1:nfft/2, :), p04(1:nfft/2, 1:641)];
m2 = m2(1:50, :);
inp_all = [flipud(m1); m2]; % alias for input matrix
```

```
7  tar_all = vaw0103_targets; % Alias for target matrix. Similarly with p0x matrices, targets
       vector must be saved in workspace before calling this script.
8
9  %% Cell array, where each cell contains indices of corresponding pattern in target matrix
10 indices = cell(1,31);
11 for m = 1:31
12     indices{m} = (tar_all(:,m) > .5); % "> .5" is there only not to compare with zero
           directly.
13 end
14
15 nsamples = 1000; % Goal number of samples for every pattern, which occured in original
       data at least once
16
17 %% Generating new training data by averaging the old
18 for m = 5:27 % Cells 1:4 and 28:31 were empty in original data
19     nind_m = sum(indices{m}); % Number of ones in indices cell indices{m}
20     while (nsamples - nind_m > 0) % Until we reach goal... (1000 samples | nsamples
           samples)
21         rn1 = floor(rand(1, 1) * (nind_m) + 1); % Random number from range 1:(nind_m)
               inclusively
22         rn2 = floor(rand(1, 1) * (nind_m) + 1); % The same
23         a = inp_all(:, indices{m}); % All columns of inp_all relevant to actual pattern (
               velocity)
24         inp_all = [inp_all, (a(:, rn1) + a(:, rn2)) / 2]; % Addind average of randomly
               chosen pair of 'a' columns to the very end
25         tar_all = [tar_all; zeros(1, 31)]; % Addind all-zero row to the end of tar_all
               matrix...
26         tar_all(end, m) = 1; % ...and changing value to 1 on proper position
27         perm = randperm(numel(tar_all(:,1))); % Random permutation of 1:numel(tar_all(:,1)
               )
28         inp_all = inp_all(:, perm); % Randomly shifting input columns
29         tar_all = tar_all(perm, :); % Shifting tar_all rows by the same pattern
30         for n = 1:31 % Recomputing numbers of indices for each pattern
31             indices{n} = (tar_all(:,n) - 1 > -.5);
32         end
33         nind_m = sum(indices{m}); % Number of ones in indices cell indices{m}
34         %disp(nind_m) % Just to control progress visually
35     end
36 end
37
38 %% Choosing a training function
39 trainFcn = 'trainscg'; % Scaled conjugate gradient backpropagation
40
41 %% Creating a network
42 n_hiddenNeurons = 20; % Number of hidden neurons
43 net = patternnet(n_hiddenNeurons, trainFcn);
44 net.trainParam.max_fail = 10; % Maximum number of epochs in a row with increasing value of
       gradient. Default == 6.
45
46 %% Training the network
47 inp = inp_all;
48 tar = tar_all';
49 [net, tr] = train(net, inp, tar);
```

Kód E.11: Genrovanie trénovacích dát pre bezobrázkovú sieť a komplexné vyhodnotenie rýchlosti na radarovej nahrávke.

```
1 [y, fs] = audioread('<path_to_file.wav>');
2
3 l = y(:,1);
4 r = y(:,2);
5 signal = r + 1i*l;
6
7 labs = abs(l);
8 rabs = abs(r);
```

Kód E.12: Načítanie súboru a výpočet absolútnych hodnôt signálových zložiek.

```
1 %% Finding minima of abs(r)
2 min_r_ind = 1:numel(r);
```

```
min_r_ind = min_r_ind(labs < 1e-30);

n_min_r_ind = numel(min_r_ind); % Number of all minima
for m = 1:n_min_r_ind-1
    if (min_r_ind(m+1)-min_r_ind(m) < 1.1)
        min_r_ind(m) = nan;
    end
end
min_r_ind = min_r_ind(~isnan(min_r_ind));

n_min_r_ind = numel(min_r_ind); % Number of abandoned minima

%% Finding minima of abs(l)
min_l_ind = 1:numel(l);
min_l_ind = min_l_ind(labs < 1e-30);

n_min_l_ind = numel(min_l_ind);
for m = 1:n_min_l_ind-1
    if (min_l_ind(m+1)-min_l_ind(m) < 1.1)
        min_l_ind(m) = nan;
    end
end
min_l_ind = min_l_ind(~isnan(min_l_ind));

n_min_l_ind = numel(min_l_ind);

%% Computing maxima of l
max_l_ind = nan(1,n_min_l_ind);
for m = 2:n_min_l_ind
    max_local = max(labs(min_l_ind(m-1):min_l_ind(m)));
    max_local_ind = find(labs(min_l_ind(m-1):min_l_ind(m)) == max_local);
    max_local_ind = max_local_ind(ceil(end/2));
    max_l_ind(m-1) = min_l_ind(m-1) + max_local_ind;
end
max_l_ind(isnan(max_l_ind)) = 1;
max_l_ind(end) = max_l_ind(end-1);

n_max_l_ind = numel(max_l_ind);

%% Computing maxima of r
max_r_ind = nan(1,n_max_l_ind);
for m = 2:n_max_l_ind
    max_local = max(rabs(max_l_ind(m-1):max_l_ind(m)));
    max_local_ind = find(rabs(max_l_ind(m-1):max_l_ind(m)) == max_local);
    max_local_ind = max_local_ind(ceil(end/2));
    max_r_ind(m-1) = max_l_ind(m-1) + max_local_ind;
end
max_r_ind(isnan(max_r_ind)) = 1;

%% Visualizing results
%labs(max_l_ind) = labs(max_l_ind) + .1;
%rabs(max_r_ind) = rabs(max_r_ind) + .1;
%
%a = 1e5; n = 0;
%figure; hold on;
%plot((n * a + 1):((n + 1)*a),l((n * a + 1):((n + 1)*a)),'r');
%plot((n * a + 1):((n + 1)*a),r((n * a + 1):((n + 1)*a)),'b');
```

KÓD E.13: Približné nájdenie maxím absolútnych hodnôt signálových zložiek.

```
%%UNCOMMENT THIS AFTER RUNNNING PREREQUIREMENT SCRIPTS ON chodba_fwd_30_04.wav. Numbers in
      first four code lines are selected empirically and will differ for another
     correction .wav sample.
%================================================================
isig = [zeros(49,1);r(50:1322580);zeros(419,1)];
qsig = [zeros(384,1);l(385:1322510);zeros(489,1)];

nperiods_i = floor(numel(max_r_ind(max_r_ind > 50 & max_r_ind < 1322580))/2);
nperiods_q = floor(numel(max_l_ind(max_l_ind > 384 & max_l_ind < 1322510))/2);

%%Computing DC components
beta_i = sum(isig) / nperiods_i;
beta_q = sum(qsig) / nperiods_q;
```

```
12
13  %%Subtracting the DC components
14  isig = isig - beta_i;
15  qsig = qsig - beta_q;
16
17  %%Computing c1 and xi1 sine and cosine
18  c1 = sqrt((2 * sum(isig .* isig)) / (nperiods_i));
19  sinxi1 = (2 / c1) * (sum(isig .* qsig)) / (nperiods_q);
20  cosxi1 = sqrt(1 - sinxi1 ^ 2);
21
22  %Computing A,B,C
23  A = 1 / c1;
24  B = -sinxi1 / (c1 * cosxi1);
25  C = 1 / cosxi1;
26  %================================================================
27
28  %%Phase correction
29  isig = A * (r - beta_i);
30  qsig = B * (r - beta_i) + C * (l - beta_q);
31
32  %%Absolute value correction
33  isig = 2.6 * isig; %2.6 --> experimental constant
34
35  %%%Visualizing complex signal plot
36  %sig = isig + 1i * qsig;
37  %figure
38  %plot(sig);
39  %hold on;
40  %plot([-.2,.2],[0,0]);
41  %plot([0,0],[-.2,.2]);
42  %xlim([-5,5]);
43  %ylim([-5,5]);
```

Kód E.14: Výpočet korekčných koeficientov a samotná korekcia signálu tak, ako je popísané v časti 3.4.2. Pred spustením tohoto skriptu je na skúmanej nahrávke nutné spustiť tiež postupne skripty E.12 a E.13.

```
1   [y, fs] = audioread('<path_to_file.wav>');
2
3   l = y(:,1);
4   r = y(:,2);
5   signal = r + 1i*l;
6
7   labs = abs(l);
8   rabs = abs(r);
9
10  % Finding minima of abs(r)
11  min_r_ind = 1:numel(r);
12  min_r_ind = min_r_ind(labs < 1e-30);
13
14  n_min_r_ind = numel(min_r_ind);
15  for m = 1:n_min_r_ind-1
16      if (min_r_ind(m+1)-min_r_ind(m) < 1.1)
17          min_r_ind(m) = nan;
18      end
19  end
20  min_r_ind = min_r_ind(~isnan(min_r_ind));
21
22  %% Finding minima of abs(l)
23  min_l_ind = 1:numel(l);
24  min_l_ind = min_l_ind(labs < 1e-30);
25
26  n_min_l_ind = numel(min_l_ind);
27  for m = 1:n_min_l_ind-1
28      if (min_l_ind(m+1)-min_l_ind(m) < 1.1)
29          min_l_ind(m) = nan;
30      end
31  end
32  min_l_ind = min_l_ind(~isnan(min_l_ind));
33
34  n_min_l_ind = numel(min_l_ind);
35
```

```
36 %% Computing maxima of l
37 max_l_ind = nan(1,n_min_l_ind);
38 for m = 2:n_min_l_ind
39     max_local = max(labs(min_l_ind(m-1):min_l_ind(m)));
40     max_local_ind = find(labs(min_l_ind(m-1):min_l_ind(m)) == max_local);
41     max_local_ind = max_local_ind(ceil(end/2));
42     max_l_ind(m-1) = min_l_ind(m-1) + max_local_ind;
43 end
44 max_l_ind(isnan(max_l_ind)) = 1;
45 max_l_ind(end) = max_l_ind(end-1);
46
47 n_max_l_ind = numel(max_l_ind);
48
49 %% Computing maxima of r
50 max_r_ind = nan(1,n_max_l_ind);
51 for m = 2:n_max_l_ind
52     max_local = max(rabs(max_l_ind(m-1):max_l_ind(m)));
53     max_local_ind = find(rabs(max_l_ind(m-1):max_l_ind(m)) == max_local);
54     max_local_ind = max_local_ind(ceil(end/2));
55     max_r_ind(m-1) = max_l_ind(m-1) + max_local_ind;
56 end
57 max_r_ind(isnan(max_r_ind)) = 1;
58
59 %% Determining motion direction
60 max_l_ind = max_l_ind(l(max_l_ind) > 0);
61
62 numelmaxlind = numel(max_l_ind);
63 direction = nan(1,numel(l));
64 direction(1:max_l_ind(1)) = ones(1,max_l_ind(1));
65 for m = 1:numelmaxlind-1
66     direction(max_l_ind(m):max_l_ind(m+1)) = r(max_l_ind(m))/abs(r(max_l_ind(m)));
67 end
68
69 %% Visualizing results
70 figure
71 area(direction);
```

KÓD E.15: Rozlíšenie smeru pohybu pomocou znamienka I signálu v maximách Q signálu.

```
59 %% Determining motion direction
60 max_l_ind = max_l_ind(l(max_l_ind) > 0);
61
62 numelmaxlind = numel(max_l_ind);
63 direction = nan(1,numel(l));
64 direction(1:max_l_ind(1)) = ones(1,max_l_ind(1));
65 for m = 1:numelmaxlind-1
66     direction(max_l_ind(m):max_l_ind(m+1)) = r(max_l_ind(m))/abs(r(max_l_ind(m)));
67 end
68
69 %% Smoothing result
70 windwidth = 3000; % Mode filter window width
71 direction_mode = nan(1,numel(direction)-windwidth); % Initialization of result vector
72 for m = 1:numel(direction)-windwidth
73     direction_mode(m) = mode(direction(m:m+windwidth));
74 end
75
76 %% Visualizing results
77 direction_mode(isnan(direction_mode)) = 0;
78 figure
79 plot(direction_mode);
80 ylim([-1.1,1.1]);
```

KÓD E.16: Doplnenie kódu E.15 o modusový filter.

```
59  %=========================================================================
60  % Constants
61  %=========================================================================
62  N_DIRBUFF = 8000;
63  N_RSIGNSINMAXL = 1323e3; % For 30s recordings 1323e3, for 60s 2*1323e3.
64  ZERO_LEVEL = 1e-30;
```

```
65  N_MODEFILTER = 15000;
66
67  %=========================================================================
68  % Signal
69  %=========================================================================
70  [y, fs] = audioread('<path_to_file.wav>');
71  signal_l = y(:,1);
72  signal_r = y(:,2);
73
74  %=========================================================================
75  % Variables
76  %=========================================================================
77  initialized = 0;
78  min_l_ind_1 = N_DIRBUFF;
79  min_l_ind_2 = N_DIRBUFF;
80  min_l_ind_1_set = 0;
81  min_l_ind_2_set = 0;
82  actual_sign = 0;
83  nul_l = 1;
84  next_sample = 1; % In Matlab we start indexing from 1
85  result_index = 1; % In Matlab we start indexing from 1
86  r_signs_in_max_l = nan(1,N_RSIGNSINMAXL);
87  r_signs_in_max_l_index = 1;
88
89
90  %=========================================================================
91  %=========================================================================
92  % Program body
93  %=========================================================================
94  %=========================================================================
95
96  number_of_samples = 1323e3; % For 30s recordings 1323e3, for 60s 2*1323e3.
97  beginning = 1;
98  l = signal_l(beginning:(beginning + number_of_samples - 1));
99  r = signal_r(beginning:(beginning + number_of_samples - 1));
100 % ... so now we have signal vecotors analogous with C++ implementation
101
102 % Initialization of dirbuff with zeros
103 dirbuff_l = zeros(1, N_DIRBUFF);
104 dirbuff_r = zeros(1, N_DIRBUFF);
105
106
107 for m = 1:N_RSIGNSINMAXL
108
109    [dirbuff_l, dirbuff_r] = add_sample(dirbuff_l, dirbuff_r, l, r, next_sample); % Adding
          new sample
110
111    if (min_l_ind_1 < (N_DIRBUFF / 4))
112        % reset
113        min_l_ind_1 = N_DIRBUFF; % Index of first minimum of l signal absolute values.
114        min_l_ind_2 = N_DIRBUFF; % Index of first minimum of r signal absolute values.
115        min_l_ind_1_set = 0; % Flag of setting of first minimum (see comment next to
               min_l_ind_1)
116        min_l_ind_2_set = 0; % Flag of setting of second minimum (see comment next to
               min_l_ind_1)
117        initialized = 0; % Flag of initialization progress. 1 == initialized, 0 == not
               initialized.
118        actual_sign = 0; % Result of this algorithm -- the sign of r signal in preceding
               maximum of l signal.
119        % end of reset
120    else
121        if (initialized == 0) % If still not initialized
122            if (abs(dirbuff_l(end)) > ZERO_LEVEL) % If NONzero came
123                if (nul_l == 0) % If we are NOT in zero queue
124                    if (min_l_ind_2_set) % If second minimum is set. First minimum must
                           be set already because we initialize the buffer with zeros
125                        min_l_ind_1 = min_l_ind_1 - 1;
126                        min_l_ind_2 = min_l_ind_2 - 1; % Shift both minima (meaning
                               indices of minima) one position left
127                    else % If second minimum is NOT set
128                        min_l_ind_1 = min_l_ind_1 - 1;
129                    end
130                else % If we are in zero queue
131                    if (min_l_ind_1_set) % If first minimum is set
```

```
132                         if (min_l_ind_2_set) % If second minimum is set
133                             min_l_ind_1 = min_l_ind_1 - 1;
134                            min_l_ind_2 = min_l_ind_2 - 1;
135                            initialized = 1;
136                            actual_sign = find_sign_in_max_l(dirbuff_l, dirbuff_r,
                                    min_l_ind_1, min_l_ind_2, actual_sign);
137                            % change_roles():
138                            min_l_ind_1 = min_l_ind_2;
139                            min_l_ind_2 = N_DIRBUFF;
140                            % end of change_roles()
141                            nul_l = change_nul_l(nul_l);
142                         else % If second minimum is NOT set
143                             min_l_ind_2_set = 1;
144                            min_l_ind_1 = min_l_ind_1 - 1;
145 %                             min_l_ind_2 = min_l_ind_2 - 1;
146                            nul_l = change_nul_l(nul_l);
147                         end
148                     else % If even first minimum is NOT set
149                         min_l_ind_1_set = 1;
150                         nul_l = change_nul_l(nul_l);
151                     end
152                 end
153              else % If zero came
154                 if (nul_l == 0) % If we are NOT in zero queue
155                     if (min_l_ind_2_set) % If second minimum is set. First minimum must
                            be set already because we initialize the buffer with zeros
156                         min_l_ind_1 = min_l_ind_1 - 1;
157                         min_l_ind_2 = min_l_ind_2 - 1;
158                         nul_l = change_nul_l(nul_l);
159                     else % If second minimum is NOT set
160                         min_l_ind_1 = min_l_ind_1 - 1;
161                         nul_l = change_nul_l(nul_l);
162                     end
163                  else % If we are in zero queue
164                     if (min_l_ind_1_set) % If first minimum is set
165                         if (min_l_ind_2_set) % If second minimum is set
166                             min_l_ind_1 = min_l_ind_1 - 1;
167                             min_l_ind_2 = min_l_ind_2 - 1;
168                         else % If second minimum is NOT set
169                             min_l_ind_1 = min_l_ind_1 - 1;
170                         end
171                     end % If first minimum is NOT set, do nothing -- the situation
                            remains the same as on origin
172                 end
173             end
174          else % If the computation is initialized. That means at least three zero queues
                 were present in incoming values, between each adjacent pair of which there
                 was at least one nonzero value.
175             if (dirbuff_l(end) >= ZERO_LEVEL) % If NONzero came
176                 if (nul_l == 0) % If we are NOT in zero queue
177                     min_l_ind_1 = min_l_ind_1 - 1;
178                     min_l_ind_2 = min_l_ind_2 - 1;
179                  else % If we are in zero queue
180                      min_l_ind_1 = min_l_ind_1 - 1;
181                     min_l_ind_2 = min_l_ind_2 - 1;
182                     actual_sign = find_sign_in_max_l(dirbuff_l, dirbuff_r, min_l_ind_1,
                            min_l_ind_2, actual_sign);
183                     % change_roles():
184                     min_l_ind_1 = min_l_ind_2;
185                     min_l_ind_2 = N_DIRBUFF;
186                     % end of change_roles()
187                     nul_l = change_nul_l(nul_l);
188                 end
189              else % If zero came
190                 if (nul_l == 0) % If we are NOT in zero queue
191                     min_l_ind_1 = min_l_ind_1 - 1;
192                     min_l_ind_2 = min_l_ind_2 - 1;
193                     nul_l = change_nul_l(nul_l);
194                  else % If we are in zero queue
195                     min_l_ind_1 = min_l_ind_1 - 1;
196                     min_l_ind_2 = min_l_ind_2 - 1;
197                 end
198             end
199         end
```

```
200     end
201
202      if (r_signs_in_max_l_index < N_RSIGNSINMAXL)
203          r_signs_in_max_l(r_signs_in_max_l_index) = actual_sign;
204          r_signs_in_max_l_index = r_signs_in_max_l_index + 1;
205      end
206
207      next_sample = next_sample + 1;
208
209  end
210
211 % Filtering result by mode filter
212 result_filtered = mode_filter(r_signs_in_max_l, N_MODEFILTER, N_RSIGNSINMAXL);
213
214 % Moving mean
215 mvm = movmean(r_signs_in_max_l,[15000,0]);
216 mvm = sign(mvm);
217
218 % Picture
219 figure
220 plot(r_signs_in_max_l);
221 hold on;
222 plot([nan(1, floor(N_MODEFILTER/2)), result_filtered, nan(1, ceil(N_MODEFILTER/2))], 'r',
        'LineWidth', 2);
223 plot(mvm, 'b',  'LineWidth', 2);
224
225
226 %========================================================================
227 % Local functions
228 %========================================================================
229
230 %-----------------beginning of function add_sample---------------------
231 function [buffl, buffr] = add_sample(arr_l, arr_r, sig_l, sig_r, next_sample)
232
233     buffl = [arr_l(2:end), sig_l(next_sample)];
234     buffr = [arr_r(2:end), sig_r(next_sample)];
235
236 end
237 %---------------------end of function add_sample----------------------
238
239
240 %-----------------beginning of function change_nul_l-------------------
241 function nul_l = change_nul_l(val)
242
243     if (val == 1)
244         nul_l = 0;
245     elseif (val == 0)
246         nul_l = 1;
247     else
248         disp('Error in function change_nul_l!');
249     end
250
251 end
252 %---------------------end of function change_nul_l---------------------
253
254
255 %--------------beginning of function find_sign_in_max_l----------------
256 function sign_in_max_l = find_sign_in_max_l(dirbuff_l, dirbuff_r, min_l_ind_1, min_l_ind_2
       , actual_sign)
257
258     N_DIRBUFF = numel(dirbuff_l);
259
260     max_r_ind_1 = find(abs(dirbuff_r(min_l_ind_1:min_l_ind_2)) == max(abs(dirbuff_r(
           min_l_ind_1:min_l_ind_2))));
261     max_r_ind_2 = find(abs(dirbuff_r(min_l_ind_2:N_DIRBUFF)) == max(abs(dirbuff_r(
           min_l_ind_2:N_DIRBUFF))));
262     max_r_ind_1 = min_l_ind_1 + floor(mean(max_r_ind_1)) - 1;
263     max_r_ind_2 = min_l_ind_2 + floor(mean(max_r_ind_2)) - 1;
264     max_l_ind = find(abs(dirbuff_l(max_r_ind_1:max_r_ind_2)) == max(abs(dirbuff_l(
           max_r_ind_1:max_r_ind_2))));
265     max_l_ind = max_r_ind_1 + floor(mean(max_l_ind)) - 1;
266     if (dirbuff_l(max_l_ind) > 0)
267     sign_in_max_l = sign(dirbuff_r(max_l_ind));
268     else
```

```
        sign_in_max_l = actual_sign;
    end

end
%------------------end of function find_sign_in_max_l------------------


%------------------beginning of function mode_filter-------------------
function filtered_result = mode_filter(arr, window_width, N_RSIGNSINMAXL)

    n_result = N_RSIGNSINMAXL - window_width;
    filtered_result = nan(1, n_result);

    for m = 1:n_result
        filtered_result(m) = mode(arr(m:(m + window_width)));
    end

end
%---------------------end of function mode_filter----------------------
```

KÓD E.17: Implementácia algoritmu 3 a vizualizácia výsledkov.

```
%================================
% SELECTING TRAINING SOURCES
%================================
target_sources = cell(1,7); % Some recordings as a training source data
        target_sources{1} = 'kolieskovyvozik_02';
    target_sources{2} = 'meranie14_tanec';
    target_sources{3} = 'meranie10_rychlo50';
    target_sources{4} = 'meranie15_tanec';
        target_sources{5} = 'meranie04_zkopceka';
        target_sources{6} = 'kolieskovyvozik_01';
        target_sources{7} = 'kolieskovyvozik_02';

%================================
% SETTTING LIMITS, MEMORY ALLOCATION
%================================
vx_limit_index = 372; % We will work only with a limited range of velocities. Let's say we
      'd like to work with velocities from 0 to 60 km/h \approx 17 m/s. That is equivalent
       to vx indices 1:372. Index 372 means velocity 17,5412 m/s.
recordings_path = '<path_to_folder_with_recordinds>'; % Path to folder with training
      recordings

%================================
% SPECTROGRAM PARAMETERS
%================================
nfft = 8192; % Window width
wind = hamming(nfft); % Windowing function
noverlap = (7 / 8) * nfft;
num_time_pts = 2576; % Number of time points in spectrogram of 60s recording.For 30s
      recordings is gets 1284.

%================================
% COMPUTING SPECTROGRAMS OF TRAINING RECORDINGS
%================================
for m = 1:(numel(target_sources))
    [y, fs] = audioread([recordings_path, target_sources{m}, '.wav']); % Reading recording
          to get I and Q samples (y) and sampling frequency (fs)

    l_part = y(:,1); % In-phase component
    r_part = y(:,2); % Quadrature component
    signal = l_part + 1i * r_part; % Full complex signal

    frames = nan(num_time_pts, nfft);
    for n = 1:num_time_pts
        frames(n, :) = signal(((n-1) * (nfft - noverlap) + 1):((n-1) * (nfft - noverlap) +
              nfft));
    end
    frames = frames .* repmat(hamming(nfft)', num_time_pts, 1);
    p = (1 / (2 * pi * nfft)) * (abs(fft(frames ,nfft, 2)) .^ 2)';
    %

    switch (m)
```

```
46          case 1
47              a = [[p((nfft - vx_limit_index + 2):end, 241:377); p(1, 241:377); p(1:
                    vx_limit_index, 241:377)],...
48                   [p((nfft - vx_limit_index + 2):end, 1005:1102); p(1, 1005:1102); p(1:
                         vx_limit_index, 1005:1102)],...
49                   [p((nfft - vx_limit_index + 2):end, 1104:1192); p(1, 1104:1192); p(1:
                         vx_limit_index, 1104:1192)],...
50                   [p((nfft - vx_limit_index + 2):end, 1204:1339); p(1, 1204:1339); p(1:
                         vx_limit_index, 1204:1339)],...
51                   [p((nfft - vx_limit_index + 2):end, 1643:1755); p(1, 1643:1755); p(1:
                         vx_limit_index, 1643:1755)],...
52                   [p((nfft - vx_limit_index + 2):end, 1826:1962); p(1, 1826:1962); p(1:
                         vx_limit_index, 1826:1962)],...
53                   [p((nfft - vx_limit_index + 2):end, 2371:2441); p(1, 2371:2441); p(1:
                         vx_limit_index, 2371:2441)]];
54           case 2
55              b = [[p((nfft - vx_limit_index + 2):end, 51:718); p(1, 51:718); p(1:
                    vx_limit_index, 51:718)],...
56                   [p((nfft - vx_limit_index + 2):end, 781:2271); p(1, 781:2271); p(1:
                         vx_limit_index, 781:2271)],...
57                   [p((nfft - vx_limit_index + 2):end, 2344:2576); p(1, 2344:2576); p(1:
                         vx_limit_index, 2344:2576)]];
58           case 3
59              c = [[p((nfft - vx_limit_index + 2):end, 1:1604); p(1, 1:1604); p(1:
                    vx_limit_index, 1:1604)],...
60                   [p((nfft - vx_limit_index + 2):end, 2080:2576); p(1, 2080:2576); p(1:
                         vx_limit_index, 2080:2576)]];
61           case 4
62              d = [[p((nfft - vx_limit_index + 2):end, 1:171); p(1, 1:171); p(1:
                    vx_limit_index, 1:171)],...
63                   [p((nfft - vx_limit_index + 2):end, 307:1300); p(1, 307:1300); p(1:
                         vx_limit_index, 307:1300)],...
64                   [p((nfft - vx_limit_index + 2):end, 1394:2576); p(1, 1394:2576); p(1:
                         vx_limit_index, 1394:2576)]];
65           case 5
66              e = [p((nfft - vx_limit_index + 2):end, 190:2170); p(1, 190:2170); p(1:
                    vx_limit_index, 190:2170)];
67           case 6
68              ff = [p((nfft - vx_limit_index + 2):end, 133:906); p(1, 133:906); p(1:
                    vx_limit_index, 133:906)];
69           case 7
70              g = [[p((nfft - vx_limit_index + 2):end, 1518:1559); p(1, 1518:1559); p(1:
                    vx_limit_index, 1518:1559)],...
71                   repmat([p((nfft - vx_limit_index + 2):end, 1545:1559); p(1, 1545:1559); p
                         (1:vx_limit_index, 1545:1559)], 1, 20),...
72                   repmat([p((nfft - vx_limit_index + 2):end, 1825:1830); p(1, 1825:1830); p
                         (1:vx_limit_index, 1825:1830)], 1, 20)];
73       end
74
75   end
76   spectrogram_pts = repmat([a, b, c, d, e, ff, g, rot90([a, b, c, d, e, ff, g], 2)], 1, 2);
77
78   %==============================
79   % LOADING TARGETS, SELECTING INPUTS
80   %==============================
81   targets_abs_low; % Calling the script with targets matrix (called tar_dir)
82   inp = spectrogram_pts;
83   tar = tar_abs_low';
84   tar = repmat([tar, fliplr(tar)], 1, 2);
85
86   %==============================
87   % NETWORK PARAMETERS
88   %==============================
89   training_function = 'trainscg'; % Scaled conjugate gradient with momentum
90   n_hidden = 20; % Number of hidden layer neurons
91   net_abs_low = patternnet(n_hidden, training_function); % Pattern-recognition network
92   net_abs_low.trainParam.max_fail = 30; % Maximal number of subsequent network performance
         value decreases to continue with training. Default value is 10.
93
94   %==============================
95   % TRAINING NETWORK
96   %==============================
97   [net_abs_low, tr] = train(net_abs_low, inp, tar);
```

KÓD E.18: Tréning neurónovej siete na priradenie podspektrogramu k rozmedziu absolútnych hodnôt rýchlostí 0 až 0,2367 m/s.

```
%==============================
% SELECTING TRAINING SOURCES
%==============================
target_sources = cell(1,5); % Three recordings as a training source data
    target_sources{1} = 'kolieskovyvozik_02';
    target_sources{2} = 'meranie14_tanec';
    target_sources{3} = 'meranie10_rychlo50';
    target_sources{4} = 'meranie15_tanec';
    target_sources{5} = 'meranie04_zkopceka';



%==============================
% SETTTING LIMITS, MEMORY ALLOCATION
%==============================
vx_limit_index = 372; % We will work only with a limited range of velocities. Let's say we
     'd like to work with velocities from 0 to 60 km/h \approx 17 m/s. That is equivalent
      to vx indices 1:372. Index 372 means velocity 17,5412 m/s.
recordings_path = '<path_to_folder_with_recordings>'; % Path to folder with training
     recordings

%==============================
% SPECTROGRAM PARAMETERS
%==============================
nfft = 8192; % Window width
wind = hamming(nfft); % Windowing function
noverlap = (7 / 8) * nfft;
num_time_pts = 2576; % For 60s recordings 2576, for 30s 1284.

%==============================
% COMPUTING SPECTROGRAMS OF TRAINING RECORDINGS
%==============================
for m = 1:(numel(target_sources))
    [y, fs] = audioread([recordings_path, target_sources{m}, '.wav']); % Reading recording
          to get I and Q samples (y) and sampling frequency (fs)

    l_part = y(:,1); % In-phase component
    r_part = y(:,2); % Quadrature component
    signal = l_part + 1i * r_part; % Full complex signal

    frames = nan(num_time_pts, nfft);
    for n = 1:num_time_pts
        frames(n, :) = signal(((n-1) * (nfft - noverlap) + 1):((n-1) * (nfft - noverlap) +
             nfft));
    end
    frames = frames .* repmat(hamming(nfft)', num_time_pts, 1);
    p = (1 / (2 * pi * nfft)) * (abs(fft(frames ,nfft, 2)) .^ 2)';

    switch (m)
        case 1
            a = [[p((nfft - vx_limit_index + 2):end, 241:377); p(1, 241:377); p(1:
                 vx_limit_index, 241:377)],...
                 [p((nfft - vx_limit_index + 2):end, 1005:1102); p(1, 1005:1102); p(1:
                      vx_limit_index, 1005:1102)],...
                 [p((nfft - vx_limit_index + 2):end, 1104:1192); p(1, 1104:1192); p(1:
                      vx_limit_index, 1104:1192)],...
                 [p((nfft - vx_limit_index + 2):end, 1204:1339); p(1, 1204:1339); p(1:
                      vx_limit_index, 1204:1339)],...
                 [p((nfft - vx_limit_index + 2):end, 1643:1755); p(1, 1643:1755); p(1:
                      vx_limit_index, 1643:1755)],...
                 [p((nfft - vx_limit_index + 2):end, 1826:1962); p(1, 1826:1962); p(1:
                      vx_limit_index, 1826:1962)],...
                 [p((nfft - vx_limit_index + 2):end, 2371:2441); p(1, 2371:2441); p(1:
                      vx_limit_index, 2371:2441)]];
         case 2
            b = [[p((nfft - vx_limit_index + 2):end, 51:718); p(1, 51:718); p(1:
                 vx_limit_index, 51:718)],...
```

```
                [p((nfft - vx_limit_index + 2):end, 781:2271); p(1, 781:2271); p(1:
                    vx_limit_index, 781:2271)],...
                [p((nfft - vx_limit_index + 2):end, 2344:2576); p(1, 2344:2576); p(1:
                    vx_limit_index, 2344:2576)]];
        case 3
            c = [[p((nfft - vx_limit_index + 2):end, 1:1604); p(1, 1:1604); p(1:
                vx_limit_index, 1:1604)],...
                [p((nfft - vx_limit_index + 2):end, 2080:2576); p(1, 2080:2576); p(1:
                    vx_limit_index, 2080:2576)]];
        case 4
            d = [[p((nfft - vx_limit_index + 2):end, 1:171); p(1, 1:171); p(1:
                vx_limit_index, 1:171)],...
                [p((nfft - vx_limit_index + 2):end, 307:1300); p(1, 307:1300); p(1:
                    vx_limit_index, 307:1300)],...
                [p((nfft - vx_limit_index + 2):end, 1394:2576); p(1, 1394:2576); p(1:
                    vx_limit_index, 1394:2576)]];
        case 5
            e = [p((nfft - vx_limit_index + 2):end, 190:2170); p(1, 190:2170); p(1:
                vx_limit_index, 190:2170)];
    end

end

disp(size(a))
disp(size(b))
disp(size(c))
disp(size(d))
disp(size(e))

spectrogram_pts = repmat([a, b, c, d, e, rot90([a, b, c, d, e], 2)], 1, 2);

%==============================
% LOADING TARGETS, SELECTING INPUTS
%==============================
targets_abs_mid; % Calling the script with targets matrix (called tar_dir)
inp = spectrogram_pts;
tar = tar_abs_mid';
tar = repmat([tar, fliplr(tar)], 1, 2);

%==============================
% NETWORK PARAMETERS
%==============================
training_function = 'trainscg'; % Scaled conjugate gradient with momentum
n_hidden = 20; % Number of hidden layer neurons
net_abs_mid = patternnet(n_hidden, training_function); % Pattern-recognition network
net_abs_mid.trainParam.max_fail = 30; % Maximal number of subsequent network performance
    value decreases to continue with training. Default value is 10.

%==============================
% TRAINING NETWORK
%==============================
[net_abs_mid, tr] = train(net_abs_mid, inp, tar);
```

Kód E.19: Tréning neurónovej siete na priradenie podspektrogramu k rozmedziu absolútnych hodnôt rýchlostí 0 až 1,0832 m/s.

```
%=========================================================================
%                 INITIAL STEPS
%=========================================================================

%=========================READING AUDIO DATA=============================
[y,fs] = audioread('<path_to_file.wav>');

l_part = y(:,1); % In-phase component
r_part = y(:,2); % Quadrature component
signal = l_part + 1i * r_part; % Full complex signal

%=======================PARAMETERS OF SPECTROGRAM=======================
nfft = 8192; % Window width
wind = hamming(nfft); % Windowing function
noverlap = (7 / 8) * nfft;
num_time_pts = 1 + floor((numel(signal) - nfft) / (nfft * 1/8)); % For 60s recordings
    2576, for 30s 1284.
```

```
17
18 %======================COMPUTING SPECTROGRAM VALUES=======================
19 frames = nan(num_time_pts, nfft);
20 for m = 1:num_time_pts
21     frames(m, :) = signal(((m-1) * (nfft - noverlap) + 1):((m-1) * (nfft - noverlap) +
           nfft));
22 end
23 frames = frames .* repmat(hamming(nfft)', num_time_pts, 1);
24 p = (1 / (2 * pi * nfft)) * (abs(fft(frames, nfft, 2)) .^ 2)'; % PSD from power spectrum
25
26 f = linspace(0, fs-(fs/nfft), nfft); % Frequency vector
27
28 %=========================PARAMETERS OF RADAR============================
29 alpha = pi / 4; % Elevation angle
30
31 %======================VELOCITY FROM DOPPLER SHIFT=======================
32 vx = (3/241.5*f)/(2*cos(alpha)); % velocity in forward direction
33
34 %======================SELECTING USEFUL PART OF SG=======================
35 vx_limit_index = 372; % 2kHz, which is the maximal doppler frequency, we'd like to work
       with is on index 372 of 'f' vector.
36 spectrogram_pts = [p((nfft - vx_limit_index + 2):end, :); p(1,:); p(1:vx_limit_index, :)];
        % Selecting important columns and reordering them.
37
38
39 %=========================================================================
40 %             VELOCITY PARAMETERS
41 %=========================================================================
42
43 %%============================= Direction ================================
44 N_MODEFILTER_DIR = 35; % Length of mode filter window
45 res_dir_sign = sign(sum(spectrogram_pts((size(spectrogram_pts, 1) / 2 + 1):end, :)) - sum(
       spectrogram_pts((1:(size(spectrogram_pts, 1) / 2)), :))); % Result of direction
       classification -- gets one value from set {-1, 0, 1}.
46 res_dir_sign = [nan(1, floor(N_MODEFILTER_DIR/2)), mode_filter(res_dir_sign,
       N_MODEFILTER_DIR, numel(res_dir_sign)), nan(1, ceil(N_MODEFILTER_DIR/2))];
47 res_dir = (vx_limit_index-30) * res_dir_sign + vx_limit_index + 1; % Fitting result to
       canvas.
48
49 %%================ Absolute value:scopes 1,0403 m/s ====================
50 N_MODEFILTER_SCOPES = 10;
51 res_scopes = net_abs_mid(spectrogram_pts);
52 [~, max_res_scopes_ind] = max(res_scopes);
53 res_scopes_sign = max_res_scopes_ind - 1; % To have 0 where absolute value is below the
       level (1,0403 m/s) and 1 where absolute value is above the level.
54 res_scopes_sign = [nan(1, floor(N_MODEFILTER_SCOPES/2)), mode_filter(res_scopes_sign,
       N_MODEFILTER_SCOPES, numel(res_scopes_sign)), nan(1, ceil(N_MODEFILTER_SCOPES/2))];
55 res_scopes = (vx_limit_index-40) * res_scopes_sign + vx_limit_index + 1;
56
57 % %%================ Absolute value: scopes 0,2 m/s ====================
58 N_MODEFILTER_SCOPES_LOW = 10;
59 res_scopes_low = net_abs_low(spectrogram_pts);
60 [~, max_res_scopes_low_ind] = max(res_scopes_low);
61 res_scopes_low_sign = max_res_scopes_low_ind - 1; % To have 0 where absolute value is
       below the level (1,0403 m/s) and 1 where absolute value is above the level.
62 res_scopes_low_sign = [nan(1, floor(N_MODEFILTER_SCOPES_LOW/2)), mode_filter(
       res_scopes_low_sign, N_MODEFILTER_SCOPES_LOW, numel(res_scopes_low_sign)), nan(1,
       ceil(N_MODEFILTER_SCOPES_LOW/2))];
63 res_scopes_low = (vx_limit_index-50) * res_scopes_low_sign + vx_limit_index +1;
64
65
66 %=========================================================================
67 %             SPECTRAL CENTROIDS
68 %=========================================================================
69
70 % ================== Overall spectral centroid ============================
71 SC_overall = nan(1,num_time_pts);
72 for m = 1:num_time_pts
73     act = p(1:vx_limit_index, m)';
74         positions = 1:vx_limit_index;
75     SC_overall(m) = (sum(act.*positions)/sum(act));
76 end
77
78 % Spectral centroid of 0,2346 -- 1,0403 m/s (equivalent to vx(6) -- vx(23))
```

```
SC_mid = nan(1,num_time_pts);
for m = 1:num_time_pts
    act = p(5:23, m)';
    positions = 5:23;
    SC_mid(m) = sum(act.*positions)/sum(act);
end

% = Spectral centroid of 0 -- 0,2346 m/s (equivalent to vx(1) -- vx(6)) ==
SC_low = nan(1,num_time_pts);
for m = 1:num_time_pts
    act = p(1:8, m)';
    positions = 1:8;
    SC_low(m) = sum(act.*positions)/sum(act);
end


%=========================================================================
%         APPLICATION OF COMPUTED PARAMETERS
%=========================================================================

% ========================= Direction ===================================
SC_overall = res_dir_sign .* SC_overall;
SC_mid = res_dir_sign .* SC_mid;
SC_low = res_dir_sign .* SC_low;

% ====================== Local adaptability ============================
SC_combined = SC_overall;
SC_combined(res_scopes_sign == 0) = SC_mid(res_scopes_sign == 0);
SC_combined(res_scopes_low_sign == 0) = SC_low(res_scopes_low_sign == 0);


%=========================================================================
%                VIEW RESULTS
%=========================================================================
figure

pc = pcolor(10 * log10(spectrogram_pts)); % Background -- spectrogram
set(pc, 'linestyle', 'none');
colormap(jet);

xlim([0, size(spectrogram_pts, 2)]);
ylim([0, 2 * vx_limit_index]);
view(2); hold on;

% plot(res_dir, 'k', 'LineWidth', 3);
% plot(res_scopes, 'k', 'LineWidth', 3);
% plot(res_scopes_low, 'k', 'LineWidth', 3);
% plot(1:n_timepts, (lim + 1) + (SC_overall), 'y', 'LineWidth', 2);
% plot(1:n_timepts, (lim + 1) + (SC_low), 'b', 'LineWidth', 2);
% plot(1:n_timepts, (lim + 1) + (SC_mid), 'm', 'LineWidth', 2);
plot(1:num_time_pts, (vx_limit_index + 1) + (SC_combined), 'k', 'LineWidth', 2);

plot([1,2576], [vx_limit_index + 1 + 5,vx_limit_index + 1 + 5], 'g', 'LineWidth', 1); %
    0,2364 m/s
plot([1,2576], [vx_limit_index + 1 -  5,vx_limit_index + 1 - 5], 'g', 'LineWidth', 1); %
    -0,2364 m/s
plot([1,2576], [vx_limit_index + 1 + 23,vx_limit_index + 1 + 23], 'g', 'LineWidth', 1); %
    1,0403 m/s
plot([1,2576], [vx_limit_index + 1 - 23,vx_limit_index + 1 - 23], 'g', 'LineWidth', 1); %
    -1,0403 m/s


%=========================================================================
%              LOCAL FUNCTIONS
%=========================================================================
%-----------------beginning of function mode_filter------------------
function filtered_result = mode_filter(arr, window_width, length_of_array)

    n_result = length_of_array - window_width;
    filtered_result = nan(1, n_result);

    for m = 1:n_result
        filtered_result(m) = mode(arr(m:(m + window_width)));
        if ( (filtered_result(m) ~= 0) && (filtered_result(m) ~= 1) && (filtered_result(m)
```

```
            ~= -1) )
149         disp('There is an unexpected value in result vector. Error in function
                mode_filter!');
150     end
151 end
152
153 end
154 %---------------------end of function mode_filter----------------------
```

Kód E.20: Kompletný program na vyhodnotenie dát z radarových nahrávok v postprocessingu.

```
1  % training_abs_low
2  % training_abs_mid
3
4  % Po kazdom treningu nutne premenovat "inp" na "inp_abs_low", resp.
5  % "inp_abs_mid".
6
7  dlmwrite('w_input_low.matrix', net_abs_low.IW, " ");
8  dlmwrite('w_hidden_low.matrix', net_abs_low.LW, " ");
9  dlmwrite('b_hidden_low.matrix', net_abs_low.b{2}, " ");
10 dlmwrite('b_input_low.matrix', net_abs_low.b{1}, " ");
11 row_min_low = min(inp_abs_low,[],2);
12 row_max_low = max(inp_abs_low,[],2);
13 dlmwrite('row_min_low.matrix', row_min_low, " ");
14 dlmwrite('row_max_low.matrix', row_max_low, " ");
15 dlmwrite('w_input_mid.matrix', net_abs_mid.IW, " ");
16 dlmwrite('w_hidden_mid.matrix', net_abs_mid.LW, " ");
17 dlmwrite('b_hidden_mid.matrix', net_abs_mid.b{2}, " ");
18 dlmwrite('b_input_mid.matrix', net_abs_mid.b{1}, " ");
19 row_min_mid = min(inp_abs_mid,[],2);
20 row_max_mid = max(inp_abs_mid,[],2);
21 dlmwrite('row_min_mid.matrix', row_min_mid, " ");
22 dlmwrite('row_max_mid.matrix', row_max_mid, " ");
```

Kód E.21: Pomocný skript, ktorý uloží potrebné dáta z natrénovaných neurónových sietí do textových súborov.

E.2 C++

```
1  /* class Constants_datastore
2   * ========================================================================
3   *        Class:  Constants_datastore
4   *  Description:  Contains globally-used constants
5   * ========================================================================
6   */
7  class Constants_datastore
8  {
9      public:
10         Constants_datastore();
11         ~Constants_datastore();
12         int get_N_FFT();
13         int get_NOVERLAP();
14         int get_N_DIRBUFF();
15         int get_N_RSIGNSINMAXL();
16         double get_ZERO_LEVEL();
17         int get_N_READ_SAMPLES();
18         int get_BEGINNING();
19         double get_SHORTSIZE();
20         string get_FILENAME_W_INPUT_low();
21         string get_FILENAME_W_HIDDEN_low();
22         string get_FILENAME_B_INPUT_low();
23         string get_FILENAME_B_INPUT_mid();
24         string get_FILENAME_B_HIDDEN_low();
25         string get_FILENAME_W_INPUT_mid();
26         string get_FILENAME_W_HIDDEN_mid();
27         string get_FILENAME_B_HIDDEN_mid();
28         string get_FILENAME_inp_row_max_mid();
```

```
29          string get_FILENAME_inp_row_min_mid();
30          string get_FILENAME_inp_row_max_low();
31          string get_FILENAME_inp_row_min_low();
32          int get_N_HIDDEN_mid();
33          int get_N_HIDDEN_low();
34          int get_VX_LIMIT_INDEX();
35          double get_PDB_MIN();
36          double get_PSD_MIN();
37          void set_N_READ_SAMPLES(int);
38          ddpair_t get_initpair();
39          int get_dir_mode_window_width();
40          int get_low_mode_window_width();
41
42          int get_mid_mode_window_width();
43          void set_BEGINNING();
44      private:
45          int N_FFT;
46          int NOVERLAP;
47          int N_DIRBUFF;
48          int N_RSIGNSINMAXL;
49          double ZERO_LEVEL;
50          int N_READ_SAMPLES;
51          int BEGINNING; // Starting sample index
52          double SHORTSIZE;
53          double PSD_MIN; // Minimal relevant value of PSD
54          double PDB_MIN; // Minimal relevant value of PSD in decibels
55          int VX_LIMIT_INDEX; // Index of maximal sample of PSD we would like to work with.
                372 was chosen in original Matlab implementation, because we have chosen
                velocity 63.1559 km/h as maximal. This could be easily changed, however, all
                 neural networks used here are trained for the number of samples, derived
                from this value.
56          char* FILENAME;
57          string FN, FILENAME_W_INPUT_low, FILENAME_W_INPUT_mid, FILENAME_W_HIDDEN_low,
                FILENAME_W_HIDDEN_mid, FILENAME_B_INPUT_low, FILENAME_B_INPUT_mid,
                FILENAME_B_HIDDEN_low, FILENAME_B_HIDDEN_mid, FILENAME_inp_row_max_mid,
                FILENAME_inp_row_min_mid, FILENAME_inp_row_max_low, FILENAME_inp_row_min_low
                ;
58
59          int N_HIDDEN_low, N_HIDDEN_mid; // Number of hidden neurons for each used type of
                neural network.
60
61          ddpair_t initpair;
62  };
63
64
65  /*--------------------------------------------------------------------------
66   *  Member functions
67   *-------------------------------------------------------------------------*/
68
69  Constants_datastore::Constants_datastore() {
70      N_FFT = 8192; // FFT window width
71      NOVERLAP = 7168; // Negative overlap
72      N_DIRBUFF = 8000;
73      N_RSIGNSINMAXL = 90000;
74      ZERO_LEVEL = 1e-30;
75
76      N_READ_SAMPLES = INT_MAX; // INT_MAX indicates "as many as the file contains".
77  //  N_READ_SAMPLES = 100000;
78
79  //  BEGINNING = 205334;
80      BEGINNING = 0;
81
82      SHORTSIZE = 32768.0;
83      VX_LIMIT_INDEX = 372;
84      PSD_MIN = 1e-15;
85      PDB_MIN = 10 * log10(PSD_MIN);
86
87      N_HIDDEN_mid = 20;
88      N_HIDDEN_low = 20;
89
90      FN = "<path_to_recording.wav>";
91
92      FILENAME_W_INPUT_low = "<path_to_net_low_input_layer_weights_matrix_file.extension>";
93      FILENAME_W_INPUT_mid = "<path_to_net_mid_input_layer_weights_matrix_file.extension>";
```

```
94      FILENAME_W_HIDDEN_low = "<path_to_net_low_hidden_layer_weights_matrix_file.extension>"
            ;
95      FILENAME_W_HIDDEN_mid = "<path_to_net_mid_hidden_layer_weights_matrix_file.extension>"
            ;
96      FILENAME_B_INPUT_low = "<path_to_net_low_input_layer_bias_vector_file.extension>";
97      FILENAME_B_INPUT_mid = "<path_to_net_mid_input_layer_bias_vector_file.extension>";
98      FILENAME_B_HIDDEN_low = "<path_to_net_low_hidden_layer_bias_vector_file.extension>";
99      FILENAME_B_HIDDEN_mid = "<path_to_net_mid_hidden_layer_bias_vector_file.extension>";
100     // Row extrema of training matrices
101     FILENAME_inp_row_max_mid = "<path_to_net_mid_training_matrix_row_maxima_file.extension
            >";
102     FILENAME_inp_row_min_mid = "<path_to_net_mid_training_matrix_row_minima_file.extension
            >";
103     FILENAME_inp_row_max_low = "<path_to_net_low_training_matrix_row_maxima_file.extension
            >";
104     FILENAME_inp_row_min_low = "<path_to_net_low_training_matrix_row_minima_file.extension
            >";
105
106     // Filling character array by characters from string
107     FILENAME = (char*) malloc(FN.length() * sizeof(char) + 1);
108     for (int i = 0; i < FN.length(); i++) {
109         FILENAME[i] = FN[i];
110     }
111     FILENAME[FN.length()] = '\0';
112
113     // Zero pair
114     initpair.first = 0.0;
115     initpair.second = 0.0;
116 }
117
118 Constants_datastore::~Constants_datastore() {
119     free(FILENAME);
120 }
121
122 int Constants_datastore::get_N_DIRBUFF() {
123     return N_DIRBUFF;
124 }
125
126 int Constants_datastore::get_N_RSIGNSINMAXL() {
127     return N_RSIGNSINMAXL;
128 }
129
130 double Constants_datastore::get_ZERO_LEVEL() {
131     return ZERO_LEVEL;
132 }
133
134 int Constants_datastore::get_N_READ_SAMPLES() {
135     return N_READ_SAMPLES;
136 }
137
138 int Constants_datastore::get_BEGINNING() {
139     return BEGINNING;
140 }
141
142 int Constants_datastore::get_VX_LIMIT_INDEX() {
143     return VX_LIMIT_INDEX;
144 }
145
146 double Constants_datastore::get_PDB_MIN() {
147     return PDB_MIN;
148 }
149
150 double Constants_datastore::get_PSD_MIN() {
151     return PSD_MIN;
152 }
153
154 int Constants_datastore::get_N_FFT() {
155     return N_FFT;
156 }
157
158 int Constants_datastore::get_NOVERLAP() {
159     return NOVERLAP;
160 }
161
```

```
162 int Constants_datastore::get_N_HIDDEN_mid() {
163     return N_HIDDEN_mid;
164 }
165
166 int Constants_datastore::get_N_HIDDEN_low() {
167     return N_HIDDEN_low;
168 }
169
170 void Constants_datastore::set_N_READ_SAMPLES(int n) {
171     N_READ_SAMPLES = n;
172 }
173
174 ddpair_t Constants_datastore::get_initpair() {
175     return initpair;
176 }
177
178 // Data filenames
179 string Constants_datastore::get_FILENAME_W_INPUT_low() {
180     return FILENAME_W_INPUT_low;
181 }
182
183 string Constants_datastore::get_FILENAME_W_INPUT_mid() {
184     return FILENAME_W_INPUT_mid;
185 }
186
187 string Constants_datastore::get_FILENAME_W_HIDDEN_low() {
188     return FILENAME_W_HIDDEN_low;
189 }
190
191 string Constants_datastore::get_FILENAME_W_HIDDEN_mid() {
192     return FILENAME_W_HIDDEN_mid;
193 }
194
195 string Constants_datastore::get_FILENAME_B_INPUT_low() {
196     return FILENAME_B_INPUT_low;
197 }
198
199 string Constants_datastore::get_FILENAME_B_INPUT_mid() {
200     return FILENAME_B_INPUT_mid;
201 }
202
203 string Constants_datastore::get_FILENAME_B_HIDDEN_low() {
204     return FILENAME_B_HIDDEN_low;
205 }
206
207 string Constants_datastore::get_FILENAME_B_HIDDEN_mid() {
208     return FILENAME_B_HIDDEN_mid;
209 }
210
211 string Constants_datastore::get_FILENAME_inp_row_max_mid() {
212     return FILENAME_inp_row_max_mid;
213 }
214
215 string Constants_datastore::get_FILENAME_inp_row_max_low() {
216     return FILENAME_inp_row_max_low;
217 }
218
219 string Constants_datastore::get_FILENAME_inp_row_min_low() {
220     return FILENAME_inp_row_min_low;
221 }
222
223 string Constants_datastore::get_FILENAME_inp_row_min_mid() {
224     return FILENAME_inp_row_min_mid;
225 }
226
227 void Constants_datastore::set_BEGINNING() {
228     BEGINNING = 0;
229 }
230
231 double Constants_datastore::get_SHORTSIZE() {
232     return SHORTSIZE;
233 } /* -----  end of class Constants_datastore  ----- */
```

Kód E.22: Objekt, uchovávajúci väčšinu konštánt, potrebných pri behu výpočtu.

```
/* class Wavreader
 * ========================================================================================
 *        Class:  Wavreader
 *  Description:  Operates with WAVE soundfile
 * ========================================================================================
 */
class Wavreader {
    private:
        short* data;
        int size;
        int chunkSize;
        int format;
        int subchunk1ID;
        int subchunk1Size;
        int sampleRate;
        int byteRate;
        int subchunk2ID;
        int subchunk2Size;
        char chunkID[5];
        short audioFormat;
        short numChannels;
        short blockAlign;
        short bitsPerSample;
        Constants_datastore* constants;
        FILE* filehandle;
        char* filename;
    public:
        Wavreader(Constants_datastore*);
        void read_file(char*);
        short* get_data();
        int get_n_read_samples();
};

/*-------------------------------------------------------------------------------
 *  Member functions
 *-------------------------------------------------------------------------------*/

Wavreader::Wavreader(Constants_datastore* constantsptr) {
    data = NULL;
    size = 0;
    constants = constantsptr;
    filehandle = NULL;
}


void Wavreader::read_file(char* filename) {
    filehandle = fopen(filename, "rb");

    if (filehandle) {
        fread(chunkID, sizeof(char), 4, filehandle);
        fread(&chunkSize, sizeof(int), 1, filehandle);
        fread(chunkID, sizeof(char), 4, filehandle);
        fread(chunkID, sizeof(char), 4, filehandle);
        fread(&format, sizeof(int), 1, filehandle);
        fread(&audioFormat, sizeof(short), 1, filehandle);
        fread(&numChannels, sizeof(short), 1, filehandle);
        fread(&sampleRate, sizeof(int), 1, filehandle);
        fread(&byteRate, sizeof(int), 1, filehandle);
        fread(&blockAlign, sizeof(short), 1, filehandle);
        fread(&bitsPerSample, sizeof(short), 1, filehandle);
        fread(chunkID, sizeof(char), 4, filehandle);
        fread(&subchunk2Size, sizeof(int), 1, filehandle);

        size = subchunk2Size / sizeof(short);
        data = (short*)malloc(subchunk2Size);
        fread(data, sizeof(short), size * numChannels, filehandle);

        fclose(filehandle);
```

```
69      }
70      else {
71          cout << "Error: Given WAVE file does not exist!" << endl;
72          exit(1);
73      }
74
75  }
76
77  short* Wavreader::get_data() {
78      return data;
79  }
80
81  int Wavreader::get_n_read_samples() {
82      return (size / 2);
83  }
```

Kód E.23: Súbor s deklaráciou a definíciou triedy `Wavreader`, slúžiacej na načítavanie vzoriek zo súboru vo WAVE formáte.

```
1   /* class Sample_source
2    * ================================================================================
3    *        Class:  Sample_source
4    *  Description:  Simulation of sample-shooting
5    * ================================================================================
6    */
7   class Sample_source {
8       private:
9           double* datasource;
10          int sample_index;
11          ddpair_t actual_sample;
12          int n_datasource;
13          fftw_complex* frame;
14          Constants_datastore* constants;
15          Wavreader* wav;
16      public:
17          Sample_source(double*, Constants_datastore*, fftw_complex*);
18          ddpair_t shoot_sample();
19          void set_sample_index(int);
20          void initialize_frame();
21  };
22
23  /*--------------------------------------------------------------------------------
24   *  Member functions
25   *-------------------------------------------------------------------------------*/
26
27  Sample_source::Sample_source(double* data, Constants_datastore* constantsptr, fftw_complex
        * arr) {
28      constants = constantsptr;
29      datasource = data;
30      sample_index = 0;
31      n_datasource = 2 * constants->get_N_READ_SAMPLES();
32      frame = arr;
33  }
34
35  ddpair_t Sample_source::shoot_sample() {
36      if (sample_index <= (n_datasource / 2 - 1)) {
37          actual_sample.first = datasource[2 * sample_index]; // Real part
38          actual_sample.second = datasource[2 * sample_index + 1]; // Imaginary part
39          sample_index++;
40      }
41      else {
42          cout << "Error in function Sample_source::shoot_sample(): index exceeds datastore
                length!" << endl;
43          // TODO The text output is only for testing purposes.
44          // exit(EXIT_FAILURE);
45      }
46
47      return actual_sample;
48  }
49
50  void Sample_source::initialize_frame() { // Fills the frame with first (8191) samples
51      for (int i = 0; i < constants->get_N_FFT() - 1; i++) {
52          actual_sample = this->shoot_sample();
```

```
53          frame[i][0] = actual_sample.first;
54          frame[i][1] = actual_sample.second;
55      }
56  }
57
58  void Sample_source::set_sample_index(int index) {
59      sample_index = index;
60  }
```

Kód E.24: Zdrojový súbor triedy `Sample_source`.

```
1   /* class Frame_operator
2    * ================================================================================
3    *        Class:  Frame_operator
4    *  Description:  Operates with signal-samples-buffer.
5    * ================================================================================
6    */
7   class Frame_operator {
8       private:
9           fftw_complex* frame;
10          fftw_complex* spectrum;
11          Constants_datastore* constants;
12          double* hamming_window;
13      public:
14          Frame_operator(fftw_complex*, fftw_complex*, Constants_datastore*);
15          ~Frame_operator();
16          void add_new_sample(ddpair_t*);
17          void hamming_the_frame();
18          void revert_hamming_the_frame();
19          void psd();
20          void psd2dB();
21          int direction_detector();
22  };
23
24  /*--------------------------------------------------------------------------
25   * Member functions
26   *-------------------------------------------------------------------------*/
27
28  Frame_operator::Frame_operator(fftw_complex* frameptr, fftw_complex* spectrumptr,
        Constants_datastore* constantsptr) {
29      frame = frameptr;
30      spectrum = spectrumptr;
31      constants = constantsptr;
32      hamming_window = (double*)malloc(constants->get_N_FFT() * sizeof(double));
33      for (int i = 0; i < constants->get_N_FFT(); i++) {
34          hamming_window[i] = .54 - .46 * cos(2 * M_PI * i / (constants->get_N_FFT() - 1));
35      }
36  }
37
38  Frame_operator::~Frame_operator() {
39      free(hamming_window);
40  }
41
42  void Frame_operator::hamming_the_frame() {
43  // Applies Hamming window to frame
44      double hamming_window_coefficient;
45      for (int i = 0; i < constants->get_N_FFT(); i++) {
46          frame[i][0] *= hamming_window[i];
47          frame[i][1] *= hamming_window[i];
48      }
49  }
50
51  void Frame_operator::revert_hamming_the_frame() {
52  // Reverts windowing the frame by Hamming window
53      double hamming_window_coefficient;
54      for (int i = 0; i < constants->get_N_FFT(); i++) {
55          frame[i][0] /= hamming_window[i];
56          frame[i][1] /= hamming_window[i];
57      }
58  }
59
60  void Frame_operator::add_new_sample(ddpair_t *sample) {
61      for (int i = 0; i < constants->get_N_FFT() - 1; i++) {
```

```
62          frame[i][0] = frame[i + 1][0];
63          frame[i][1] = frame[i + 1][1];
64      }
65      frame[constants->get_N_FFT() - 1][0] = (*sample).first;
66      frame[constants->get_N_FFT() - 1][1] = (*sample).second;
67  }
68
69  void Frame_operator::psd() { // Computes normalized power spectral density from standard
        frequency spectrum. PSD is real. We save its value to real part of spectrum complex
        array and set the imaginary part to zero. We do not compute the PSD for whole buffer
        , because it is not necessary.
70      for (int i = 0; i < constants->get_VX_LIMIT_INDEX(); i++) {
71          spectrum[i][0] = (1 / (2 * M_PI * constants->get_N_FFT())) * (spectrum[i][0] *
                spectrum[i][0] + spectrum[i][1] * spectrum[i][1]);
72          spectrum[i][1] = 0.0;
73          spectrum[constants->get_N_FFT() - 1 - i][0] = (1 / (2 * M_PI * constants->
                get_N_FFT())) * (spectrum[constants->get_N_FFT() - 1 - i][0] * spectrum[
                constants->get_N_FFT() - 1 - i][0] + spectrum[constants->get_N_FFT() - 1 - i
                ][1] * spectrum[constants->get_N_FFT() - 1 - i][1]);
74          spectrum[constants->get_N_FFT() - 1 - i][1] = 0.0;
75      }
76  }
77
78  int Frame_operator::direction_detector() { // Determining direction of movement (+1 ->
        forward, -1 -> backward), if the movement is present. Otherwise, returns zero.
79      int sign; // Sign of difference between sum of psd in "upper half" and sum of psd in "
            lower half".
80      double sum_upper = 0.0, sum_lower = 0.0; // Sums of psd in "sides".
81
82      for(int i = 0; i < (constants->get_VX_LIMIT_INDEX() - 1); i++) {
83          sum_upper += spectrum[constants->get_N_FFT() - constants->get_VX_LIMIT_INDEX() + 1
                + i][0];
84          sum_lower += spectrum[i][0];
85      }
86      sum_upper += spectrum[0][0];
87      sum_lower += spectrum[constants->get_VX_LIMIT_INDEX() - 1][0];
88
89      sign = (0.0 < (sum_upper - sum_lower)) - (0.0 > (sum_upper - sum_lower)); // Signum.
            Returns 1, 0 or -1.
90
91      return sign;
92  }
```

Kód E.25: Zdrojový súbor triedy `Frame_operator`.

```
1   /* class Neural_network
2    * ================================================================================
3    *        Class:  Neural_network
4    *  Description:  Neural network, trained in Matlab environment. This network has one
         input and one hidden layer.
5    * ================================================================================
6    */
7
8   class Neural_network
9   {
10      public:
11          // ----------- Constructors and destructors --------------
12          Neural_network(Constants_datastore*, nn_enum, fftw_complex*);
13          ~Neural_network();
14          // --------------------- Variables ----------------------
15          // --------------------- Functions ----------------------
16          void load();
17          void net_walkthrough(); // Normalization -> input layer -> tansig -> hidden layer
                -> softmax -> output
18          ddpair_t get_result();
19      private:
20          // --------------------- Variables ----------------------
21          Constants_datastore* constants;
22          double* input; // Input data
23          fftw_complex* spectrum; // Pointer to working array
24          ddpair_t output; // Output from last network step
25          dvec_t w_input, w_hidden, b_input, b_hidden; // Weights and biases
26          dvec_t input_row_min, input_row_max; // Extrema of weights-matrices' rows
```

```
27          nn_enum which_nn; // Indication of scope of neural network
28          int* reshape_ind; // Map of 'input' elements to network-input-friendly array size
29          int N_HIDDEN; // Number of hidden neurons
30          dvec_t input_layer_result; // output from input layer
31          ddpair_t result; // output from hidden layer
32          // ---------------------- Functions ----------------------
33          void mapminmax(); // Normalization of input vector (coefficients derived from
                training data)
34          void input_layer(); // Network layers
35          void hidden_layer(); // Network layers
36          void refresh(); // Adding new data to working array
37          void tansig(); // Tangent sigmoid
38          void softmax(); // Softmax function
39          void reconstruct_pattern_max(); // Reconstruction of pattern from output from
                softmax layer
40          void reconstruct_pattern_half(); // Reconstruction of pattern from output from
                softmax layer -- another possibility
41  };
42
43
44  /*=========================================================================
45   *  Member functions
46   *=========================================================================*/
47
48  /*-------------------------------------------------------------------------
49   *  Constructors and destructors
50   *-------------------------------------------------------------------------*/
51
52  Neural_network::Neural_network(Constants_datastore* constantsptr, nn_enum which_nn_val,
         fftw_complex* inputptr) {
53      constants = constantsptr;
54      which_nn = which_nn_val;
55      output.first = 0.0;
56      output.second = 0.0;
57
58      // ------------------------ Filling indices array
            ----------------------------------------
59      reshape_ind = (int*)malloc(2 * constants->get_VX_LIMIT_INDEX() * sizeof(int));
60      for (int i = 0; i < (constants->get_VX_LIMIT_INDEX() - 1); i++) {
61          reshape_ind[i] = constants->get_N_FFT() - constants->get_VX_LIMIT_INDEX() + 1 + i;
62          reshape_ind[constants->get_VX_LIMIT_INDEX() + i] = i;
63      }
64      reshape_ind[constants->get_VX_LIMIT_INDEX() - 1] = 0;
65      reshape_ind[2 * constants->get_VX_LIMIT_INDEX() - 1] = constants->get_VX_LIMIT_INDEX()
            - 1;
66      //
            ----------------------------------------------------------------------------------------
67
68      spectrum = inputptr;
69
70      // Allocating memory for copy of relevant elements of array, which inputptr points to.
            We will do changes on input array and we don't want to destroy its original
            values because we have to use them as an input for another neural network. For
            this reason, we have to copy elements.
71      input = (double*)malloc(2 * constants->get_VX_LIMIT_INDEX() * sizeof(double));
72
73      // Switch assigning number of hidden neurons
74      switch ( which_nn_val ) {
75          case abs_low:
76              N_HIDDEN = constants->get_N_HIDDEN_low();
77              break;
78
79          case abs_mid:
80              N_HIDDEN = constants->get_N_HIDDEN_mid();
81              break;
82          default:
83              N_HIDDEN = 0;
84              break;
85      }               /* -----  end switch  ----- */
86
87      for (int i = 0; i < N_HIDDEN; i++) { // Filling input layer result vector with zeros
88          input_layer_result.push_back(0.0);
89      }
```

```
90
91      result.first = 0.0;
92      result.second = 0.0;
93  }
94
95
96  Neural_network::~Neural_network() {
97      free(input);
98      free(reshape_ind);
99  }
100
101 /*------------------------------------------------------------------------------
102  *  Data loading
103  *-----------------------------------------------------------------------------*/
104
105 void Neural_network::load() { // Loads matrices to workspace
106     double actual_number = 0.0;
107
108     if (which_nn == abs_low) {
109         ifstream datafile;
110         datafile.open(constants->get_FILENAME_W_INPUT_low(), ifstream::in);
111         if (datafile.is_open()) {
112             while (datafile >> actual_number) {
113                 w_input.push_back(actual_number);
114             }
115         }
116         datafile.close();
117
118         datafile.open(constants->get_FILENAME_W_HIDDEN_low(), ifstream::in);
119         if (datafile.is_open()) {
120             while (datafile >> actual_number) {
121                 w_hidden.push_back(actual_number);
122             }
123         }
124         datafile.close();
125
126         datafile.open(constants->get_FILENAME_B_INPUT_low(), ifstream::in);
127         if (datafile.is_open()) {
128             while (datafile >> actual_number) {
129                 b_input.push_back(actual_number);
130             }
131         }
132         datafile.close();
133
134         datafile.open(constants->get_FILENAME_B_HIDDEN_low(), ifstream::in);
135         if (datafile.is_open()) {
136             while (datafile >> actual_number) {
137                 b_hidden.push_back(actual_number);
138             }
139         }
140         datafile.close();
141
142         datafile.open(constants->get_FILENAME_inp_row_max_low(), ifstream::in);
143         if (datafile.is_open()) {
144             while (datafile >> actual_number) {
145                 input_row_max.push_back(actual_number);
146             }
147         }
148         datafile.close();
149
150         datafile.open(constants->get_FILENAME_inp_row_min_low(), ifstream::in);
151         if (datafile.is_open()) {
152             while (datafile >> actual_number) {
153                 input_row_min.push_back(actual_number);
154             }
155         }
156         datafile.close();
157     }
158     else if (which_nn == abs_mid) {
159         ifstream datafile;
160         datafile.open(constants->get_FILENAME_W_INPUT_mid(), ifstream::in);
161         if (datafile.is_open()) {
162             while (datafile >> actual_number) {
163                 w_input.push_back(actual_number);
```

```
164             }
165         }
166         datafile.close();
167 
168         datafile.open(constants->get_FILENAME_W_HIDDEN_mid(), ifstream::in);
169         if (datafile.is_open()) {
170             while (datafile >> actual_number) {
171                 w_hidden.push_back(actual_number);
172             }
173         }
174         datafile.close();
175 
176         datafile.open(constants->get_FILENAME_B_INPUT_mid(), ifstream::in);
177         if (datafile.is_open()) {
178             while (datafile >> actual_number) {
179                 b_input.push_back(actual_number);
180             }
181         }
182         datafile.close();
183 
184         datafile.open(constants->get_FILENAME_B_HIDDEN_mid(), ifstream::in);
185         if (datafile.is_open()) {
186             while (datafile >> actual_number) {
187                 b_hidden.push_back(actual_number);
188             }
189         }
190         datafile.close();
191 
192         datafile.open(constants->get_FILENAME_inp_row_max_mid(), ifstream::in);
193         if (datafile.is_open()) {
194             while (datafile >> actual_number) {
195                 input_row_max.push_back(actual_number);
196             }
197         }
198         datafile.close();
199 
200         datafile.open(constants->get_FILENAME_inp_row_min_mid(), ifstream::in);
201         if (datafile.is_open()) {
202             while (datafile >> actual_number) {
203                 input_row_min.push_back(actual_number);
204             }
205         }
206         datafile.close();
207     }
208     else {
209         cout << "Error in function Neural_network::load_wb(nn_enum): unrecognized input"
                 << endl;
210         // TODO Another error indication for real application
211         // exit(EXIT_FAILURE);
212     }
213 }
214 
215 /*-----------------------------------------------------------------------------
216  *  Network-walkthrough
217  *-----------------------------------------------------------------------------*/
218 
219 void Neural_network::mapminmax() { // Normalizing input array values by training matrix
        row minima and maxima
220     if ((input_row_max.size() == input_row_min.size()) && (input_row_min.size() == (2 *
             constants->get_VX_LIMIT_INDEX()))) {
221         for (int i = 0; i < input_row_min.size(); i++) {
222             input[i] = 2 * ((input[i] - input_row_min[i]) / (input_row_max[i] -
                     input_row_min[i])) - 1;
223         }
224     }
225     else {
226         cout << "Error in function Neural_network::mapminmax()" << endl;
227     }
228 }
229 
230 
231 void Neural_network::input_layer() {
232 
233     fill(input_layer_result.begin(), input_layer_result.end(), 0.0); // Resetting input
```

```
          layer vector to zeros
234     for(int i = 0; i < N_HIDDEN; i++) { // Multiplying input weights by input
235         for (int j = 0; j < (2 * constants->get_VX_LIMIT_INDEX()); j++) {
236             input_layer_result[i] += (input[j] * w_input[i * (2 * constants->
                    get_VX_LIMIT_INDEX()) + j]);
237         }
238     }
239
240     for(int i = 0; i < N_HIDDEN; i++) { // Adding biases
241         input_layer_result[i] += b_input[i];
242     }
243
244 }
245
246
247 void Neural_network::hidden_layer() {
248     result.first = 0.0;
249     result.second = 0.0; // Resetting result pair to zeros
250
251     for(int i = 0; i < N_HIDDEN; i++) { // Multiplying input weights by input
252         result.first += (input_layer_result[i] * w_hidden[i]);
253         result.second += (input_layer_result[i] * w_hidden[N_HIDDEN + i]);
254     }
255
256     result.first += b_hidden[0];
257     result.second += b_hidden[1]; // Adding biases
258 }
259
260
261 void Neural_network::refresh() { // Copy relevant elements of psd (spectrum) array to
      working array 'input'.
262     for (int i = 0; i < (2 * constants->get_VX_LIMIT_INDEX()); i++) {
263         input[i] = spectrum[reshape_ind[i]][0];
264     }
265 }
266
267
268 void Neural_network::tansig() { // Tangent sigmoid function applied to result of input
      layer coefficients multiplication
269     for (int i = 0; i < N_HIDDEN; i++) {
270         input_layer_result[i] = 2 / (1 + exp(-2 * input_layer_result[i])) - 1;
271     }
272 }
273
274
275 void Neural_network::softmax() { // Softmax function application to output from hidden
      layer
276     double sumexp = 0.0;
277     sumexp = exp(result.first) + exp(result.second);
278     result.first = exp(result.first) / sumexp;
279     result.second = exp(result.second) / sumexp;
280 }
281
282
283 void Neural_network::reconstruct_pattern_max() { // Reconstructs pattern simply by
      selecting its maximum and assigning 1 to its position. This is useful only in cases
      when there is only one non-zero point (one) in all possible patters. However, that
      is our case.
284     bool first;
285     (result.first > result.second) ? (first = true) : (first = false);
286     result.first = (double)first;
287     result.second = (double)(!first);
288 }
289
290
291 void Neural_network::reconstruct_pattern_half() { // Assigns 1 to every value of input,
      which value is higher than 0.5
292     (result.first > .5) ? (result.first = 1.0) : (result.first = 0.0);
293     (result.second > .5) ? (result.second = 1.0) : (result.second = 0.0);
294 }
295
296
297 void Neural_network::net_walkthrough() { // Just an more human alias :-)
298     this->refresh();
```

```
299     this->mapminmax();
300     this->input_layer();
301     this->tansig();
302     this->hidden_layer();
303     this->softmax();
304     this->reconstruct_pattern_max();
305 }
306
307
308 /*--------------------------------------------------------------------------
309  *  Getters and setters
310  *-------------------------------------------------------------------------*/
311
312 ddpair_t Neural_network::get_result() {
313     return result;
314 }
```

Kód E.26: Zdrojový súbor triedy `Neural_network`.

```
1  /* class Spectrum_operator
2   * ============================================================================
3   *        Class:  Spectrum_operator
4   *  Description:  Operates with spectrum buffer. More precisely, contains tools for
        spectral centroid computation.
5   * ============================================================================
6   */
7  class Spectrum_operator {
8      private:
9          fftw_complex* spectrum;
10         Constants_datastore* constants;
11         double sum_s(int, int);
12         double sum_si(int, int);
13     public:
14         Spectrum_operator(fftw_complex*, Constants_datastore*);
15         int sc(sc_enum);
16 };
17
18
19 /*--------------------------------------------------------------------------
20  *  Member functions
21  *-------------------------------------------------------------------------*/
22
23 Spectrum_operator::Spectrum_operator(fftw_complex* spectrumptr, Constants_datastore*
       constantsptr) {
24     spectrum = spectrumptr;
25     constants = constantsptr;
26 }
27
28
29 double Spectrum_operator::sum_s(int start, int stop) { // Returns sum of values on indices
        start:stop (inclusively) of spectrum's "real part". When this function is being
       called, there should be power spectral density stored in spectrum array. Power
       spectral density is real quantity, however, the array is still complex with zero
       values on its imaginary part positions.
30     double result = 0.0;
31     for (int i = start; i < (stop + 1); i++) {
32         result += spectrum[i][0];
33     }
34
35     return result;
36 }
37
38
39 double Spectrum_operator::sum_si(int start, int stop) { // Returns sum of values on
       indices start:stop (inclusively) of spectrum's "real part" multiplied by its index.
       When this function is being called, there should be power spectral density stored in
        spectrum array. Power spectral density is real quantity, however, the array is
       still complex with zero values on its imaginary part positions.
40     double result = 0.0;
41     for (int i = start; i < (stop + 1); i++) {
42         result += spectrum[i][0] * i;
43     }
44
```

```
45      return result;
46  }
47
48
49  int Spectrum_operator::sc(sc_enum which_sc) {
50      int sc_result = INT_MIN;
51      switch (which_sc) {
52          case low:
53              sc_result = (this->sum_si(0, 7) / this->sum_s(0, 7));
54              if (sc_result < 0) sc_result = 0;
55              break;
56
57          case mid:
58              sc_result = (this->sum_si(4, 22) / this->sum_s(4, 22));
59              if (sc_result < 0) sc_result = 4;
60              break;
61
62          case all:
63              sc_result = (this->sum_si(0, constants->get_VX_LIMIT_INDEX() - 1) / this->
                    sum_s(0, constants->get_VX_LIMIT_INDEX() - 1));
64              if (sc_result < 0) sc_result = 0;
65              break;
66
67          default:
68              break;
69      }   /* -----  end switch  ----- */
70
71      return sc_result;
72  }
```

Kód E.27: Zdrojový súbor triedy `Spectrum_operator`.

```
1   /*
2    * ========================================================================================
3    *
4    *       Filename:  types.h
5    *
6    *    Description:  Header file with type definitions
7    *
8    *        Version:  1.0
9    *        Created:  04/17/2018 12:59:01 PM
10   *       Revision:  none
11   *       Compiler:  g++
12   *
13   *         Author:  Matej Oravec (), matoravec@gmail.com
14   *   Organization:  CTU in Prague
15   *
16   * ========================================================================================
17   */
18
19  typedef pair<double, double> ddpair_t; // Pair of integer and double
20
21  typedef vector<double> dvec_t; // Vector of doubles
22  typedef vector<int> ivec_t; // Vector of doubles
23
24  typedef vector<dvec_t> dmatrix_t; // Matrix of doubles
25
26  enum nn_enum {abs_low, abs_mid};
27
28  enum sc_enum {low, mid, all};
```

Kód E.28: Hlavičkový súbor `types.h`.

```
1   class Mode_filter {
2       private:
3           int window_width_dir; // Mode filter window width for direction
4           int window_width_low; // Mode filter window width for absolute values of low
                velocities
5           int window_width_mid; // Mode filter window width for absolute values of middle
                velocities
6           int* window; // Window, which is modus computed from
7           int n_samples_inside; // Number of samples already stored in window
```

```
8         bool initialized; // Initialization progress flag. Turns to 'true' when amount oq
              samples equal to half of the direction window width is loaded.
9         int howmany[3][3]; // Stores how many minus ones, zeros and ones are in subwindows
              (subwindow are rows of 'window' -- first row is connected with low absolute
              values, second with middle absolute values and the third with direction).
              Naturally, first two subwindows (rows) do not contain minus ones.
              Nevertheless used algorithm will get over it.
10        int modes[3]; // Results -- modes of 'window''s rows
11        int UNINITIALIZED_RETURN_VALUE[3]; // Value, which is returned from mode()
              function when the computation is not initialized.
12        int max; // Temporary maximum
13    public:
14        Mode_filter();
15        ~Mode_filter();
16        void add_values(int, int, int); // Adds three values to window; Each one to
              separate "row". First one to first, second one to second and the third to
              third one.
17        int* mode(); // Returns modes of rows of 'window'.
18 };
19
20 /*==========================================================================
21  * Member functions
22  *=========================================================================*/
23
24 /*--------------------------------------------------------------------------
25  *  Constructors & destructors
26  *-------------------------------------------------------------------------*/
27
28 Mode_filter::Mode_filter() {
29     window_width_dir = 36, window_width_low = 10; window_width_mid = 10;
30     window = (int*)malloc(window_width_dir * 3 * sizeof(int));
31     for (int i = 0; i < 3 * window_width_dir; i++) { // Initialization with zeros
32         window[i] = 0;
33     }
34     n_samples_inside = 0;
35     initialized = false;
36     UNINITIALIZED_RETURN_VALUE[0] = INT_MIN; UNINITIALIZED_RETURN_VALUE[1] = INT_MIN;
           UNINITIALIZED_RETURN_VALUE[2] = INT_MIN;
37 }
38
39
40 Mode_filter::~Mode_filter() {
41     free(window);
42 }
43
44 /*--------------------------------------------------------------------------
45  *  Other member functions
46  *-------------------------------------------------------------------------*/
47
48 void Mode_filter::add_values(int val_low, int val_mid, int val_dir) {
49     for (int i = window_width_dir - 1; i > 0; i--) {
50         window[i] = window[i - 1];
51         window[window_width_dir + i] = window[window_width_dir + i - 1];
52         window[2 * window_width_dir + i] = window[2 * window_width_dir + i - 1];
53         window[0] = val_low;
54         window[window_width_dir] = val_mid;
55         window[2 * window_width_dir] = val_dir;
56     }
57     if (!initialized) {
58         n_samples_inside++;
59         if (n_samples_inside == window_width_dir / 2) initialized = true;
60     }
61 }
62
63
64 int* Mode_filter::mode() {
65     if (initialized) {
66         for(int i = 0; i < 3; i++) { // Resetting counting array
67             for(int j = 0; j < 3; j++) {
68                 howmany[i][j] = 0;
69             }
70         }
71
72         for(int i = 13; i < (13 + 10); i++) {
```

```
73              if (window[i] == -1) (howmany[0][0])++;
74              else if (window[i] == 0) (howmany[0][1])++;
75              else if (window[i] == 1) (howmany[0][2])++;
76              else cout << "Error in Mode_filter::mode()!" << endl;
77          }
78          for(int i = window_width_dir + 13; i < (window_width_dir + 13 + 10); i++) {
79              if (window[i] == -1) (howmany[1][0])++;
80              else if (window[i] == 0) (howmany[1][1])++;
81              else if (window[i] == 1) (howmany[1][2])++;
82              else cout << "Error in Mode_filter::mode()!" << endl;
83          }
84          for(int i = 2 * window_width_dir; i < (3 * window_width_dir); i++) {
85              if (window[i] == -1) (howmany[2][0])++;
86              else if (window[i] == 0) (howmany[2][1])++;
87              else if (window[i] == 1) (howmany[2][2])++;
88              else cout << "Error in Mode_filter::mode()!" << endl;
89          }
90
91          for(int i = 0; i < 3; i++) {
92              max = INT_MIN;
93              for(int j = 0; j < 3; j++) {
94                  if(howmany[i][j] > max) {
95                      max = howmany[i][j];
96                      modes[i] = j - 1; // 2-1=1, 1-1=0, 0-1=-1.
97                  }
98              }
99          }
100         return modes;
101     }
102     else return UNINITIALIZED_RETURN_VALUE;
103
104 }
```

Kód E.29: Zdrojový súbor triedy `Mode_filter`.

```
1   // template of class Delay_line
        ------------------------------------------------------------------------------
2   /* This template is a FIFO structure with fixed length; Alternatively, it could be called
        cyclic buffer. */
3   //
        ------------------------------------------------------------------------------------------------------------

4   template <class T>
5   class Delay_line {
6       private:
7           int delay;
8           T* buffer;
9           int n_rows;
10          T* first;
11      public:
12          Delay_line(int);
13          Delay_line(int, int);
14          ~Delay_line();
15          void push_back(T);
16          void push_back(T*);
17          T* get_first();
18  };
19
20  /*============================================================================
21   * Member functions
22   *============================================================================*/
23
24  /*----------------------------------------------------------------------------
25   *  Constructors & destructors
26   *----------------------------------------------------------------------------*/
27
28  template <class T>
29  Delay_line<T>::Delay_line(int delayval) {
30      delay = delayval;
31      n_rows = 1;
32      buffer = (T*)malloc(n_rows * delay * sizeof(T));
33      for (int i = 0; i < n_rows * delay; i++) {
34          buffer[i] = numeric_limits<T>::min();
```

```
35      }
36      first = (T*)malloc(sizeof(T));
37  }
38
39  template <class T>
40  Delay_line<T>::Delay_line(int delayval, int n_rowsval) {
41      delay = delayval;
42      n_rows = n_rowsval;
43      buffer = (T*)malloc(n_rows * delay * sizeof(T));
44      for (int i = 0; i < n_rows * delay; i++) {
45          buffer[i] = numeric_limits<T>::min();
46      }
47      first = (T*)malloc(n_rows * sizeof(T));
48  }
49
50  template <class T>
51  Delay_line<T>::~Delay_line() {
52      free(buffer);
53      free(first);
54  }
55
56  /*--------------------------------------------------------------------------
57   *  Other member functions
58   *-------------------------------------------------------------------------*/
59
60  template <class T>
61  void Delay_line<T>::push_back(T val) {
62      if (n_rows == 1) {
63          for(int i = 0; i < (delay - 1); i++) {
64              buffer[i] = buffer[i + 1];
65          }
66          buffer[delay - 1] = val;
67      }
68      else { // Add the same value to the end of every row
69          for(int i = 0; i < n_rows; i++) {
70              for(int j = 0; j < (delay - 1); j++) {
71                  buffer[i * delay + j] = buffer[i * delay + j + 1];
72              }
73              buffer[(i + 1) * delay - 1] = val;
74          }
75      }
76  }
77
78  template <class T>
79  void Delay_line<T>::push_back(T* val) {
80      for(int i = 0; i < n_rows; i++) {
81          for(int j = 0; j < (delay - 1); j++) {
82              buffer[i * delay + j] = buffer[i * delay + j + 1];
83          }
84          buffer[(i + 1) * delay - 1] = val[i];
85      }
86  }
87
88  template <class T>
89  T* Delay_line<T>::get_first() {
90      for(int i = 0; i < n_rows; i++) {
91          first[i] = buffer[i * delay];
92      }
93      return first;
94  }
```

Kód E.30: Šablóna `Delay_line`.

Literatúra

[1] MATĚJKA, Štěpán. *Radarová odometrie – Určení brzdné dráhy vozidla pomocí dopplerovského radaru.* Praha, 2017

[2] RICHARDS, Mark A. *Fundamentals of Radar Signal Processing.* 2nd ed. New York: McGraw – Hill, c2014. ISBN 978-0-07-179832-7.

[3] DOKE, Jiro. *Grabit.* Dokumentácia. Online. Dostupné z `https://www.mathworks.com/matlabcentral/fileexchange/7173-grabit` [27. 6. 2017]

[4] FRIGO, Matteo a Steven JOHNSON. *FFTW.* Dokumentácia. Online. Dostupné z `http://www.fftw.org/fftw3.pdf` [9. 9. 2017]

[5] MATOUŠEK, Rostislav. *CW radarový senzor* 11 GHz *pro měření rychlosti automobilů.* Diplomová práca. Praha: ČVUT, 2015.

[6] STILL, Michael. *The definitive guide to imagemagick.* Berkeley: Apress, c2006. ISBN 1-59059-590-4.

[7] SAPP, Craig S. *WAVE PCM soundfile format.* Dokumentácia C++ knižnice. Online. Dostupné z `http://soundfile.sapp.org/doc/WaveFormat/` [9. 9. 2017]

[8] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep Learning.* Cambridge, Massachusetts: The MIT Press, 2016. ISBN 978-0-262-03561-3.

[9] LIBERTY, Jesse. *Naučte se C++ za 21 dní.* 2. aktualiz. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1583-1.

[10] InnoSenT. *Data Sheet IVS-167.* Dokumentácia radarovného senzoru. Online. Dostupné z `http://www.innosent.de/fileadmin/media/dokumente/DATASHEETS_2016/Datenblatt_IVS-167_V3.1.pdf` [10. 10. 2017]

[11] ELLINGSON, Steven W. *Correcting I-Q Imbalance in Direct Conversion Receivers.* Online. Dostupné z `http://www.faculty.ece.vt.edu/swe/argus/iqbal.pdf` [13. 11. 2017]

[12] MASSAR, Melody L. a Matthew FICKUS a Erik Bryan a kolektív. *Fast computation of spectral centroids.* Advances in Computational Mathematics 35:83. Online. Dostupné z `https://doi.org/10.1007/s10444-010-9167-y` [21. 11. 2017].

[13] OPPENHEIM, Alan V., Ronald W. SCHAFER a John R. BUCK *Discrete time signal processing*. 2nd ed. Upper Saddle River: Prentice Hall, c1999. ISBN 0137549202

[14] WOLFF, Christian. *Radar Basics: Duplexer*. Online. Dostupné z `www.radartutorial.eu/06.antennas/Duplexer.en.html` [26. 3. 2018].

[15] PECHAČ, Pavel a Stanislav ZVÁNOVEC. *Základy šíření vln pro plánování pozemních rádiových spojů*. Praha: BEN – technická literatura, 2007. ISBN 978-80-7300-223-7.

[16] STALLMAN, Richard M. a Roland MCGRATH. *GNU make: a program for directed compilation*. Boston: GNU, c2002. ISBN 1-882114-82-5.

[17] BELLIS, Mary. *RADAR and Doppler RADAR: Invention and History*. Blogový príspevok. Online. Dostupné z `https://www.thoughtco.com/radar-and-doppler-history-4070020`.

[18] SIMON, Dan. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. Hoboken: Wiley, 2006. ISBN 978-0-470-04534-3.

[19] LABBE, Roger R. Jr. *Kalman and Bayesian Filters in Python*. Online. Dostupné z `https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python` [1. 5. 2018]

[20] KAPLAN, Elliot D. a C. HEGARTY. *Understanding GPS: principles and applications*. 2nd ed. Boston: Artech House, c2006. ISBN 1-58053-894-0.

Zoznam príloh

nahravky.tar.gz
prepare_for_classification.tar.gz
patchwork_generator.tar.gz
priemgen_final.tar.gz
velocity_cpp.tar.gz

Register