KONTINUÁLNE SPRACOVANIE ZVUKU V MATLABE

Ing. Stanislav Števo, PhD.[1], Ing. Marek Vnuk[1], Ing. Jakub Osuský, PhD.[2]

[1]Ústav robotiky a kybernetiky, [2]Ústav aplikovanej mechatroniky
Fakulta elektrotechniky a informatiky, ilkovičova 3, 812 19 Bratislava
Slovenská technická univerzita

Abstrakt
Článok sa zaoberá možnosťami súčasného nahrávania a prehrávania zvuku v softvérovom prostredí Matlab. Jednotlivé príkazy so stručným popisom tak tvoria jednoduchý a prehľadný manuál, tvoriaci základný východiskový bod pre zvládnutie obdobných aplikácií. Článok opisuje aj problémy a riešenia, ktoré vyvstanú pri potrebe kontinuálneho spracovania zvuku. Aplikačnou doménou prípadovej štúdie je spracovanie nahratých zvukových stôp z vibrometra. Pozornosť zameraná najmä na súbežné prehrávanie a nahrávanie zvukovej stopy ako aj jej súčasného spracovávania.

Kľúčové slová: kontinuálne spracovanie zvuku, Matlab, detekcia zlyhania základných životných funkcií, vibrometer

Úvod

Či už dýchanie, alebo samotný tlkot srdca vyvoláva merateľné vibrácie na povrchu tela. Zaznamenaním a vyhodnotením týchto vibrácii v reálnom čase vieme diagnostikovať aktuálne poruchy činnosti základných životných funkcií alebo ich priame zlyhanie. Za týmto účelom bol zhotovený cenovo dostupný, ale dostatočne citlivý vibrometer. Je založený na triangulačnej vibrometrii spojením modulovaného lasera a PSD detektora. Na tvorbu modulačného signálu pre laser a zaznamenávanie signálu z PSD detektora sa využíva zvuková karta ako lacný digitálno-analogový a analógovo-digitálny prevodník. Spracovanie dát (softvérová časť) je realizované v prostredí Matlab. V ňom bola pozornosť zameraná na súbežné prehrávanie aj nahrávanie zvukovej stopy a jej súčasného spracovávania.

Ovládací program pre spracovanie zvuku

Vývoj ovládacieho softwaru vibrometra bol realizovaný v prostredí Matlab 2012b (v 32 bitovej verzii, keďže využívaný Data Acquisition Toolbox na 64 bitovej verzii nefunguje). Riešenie spracovania zvuku bolo rozdelené do 4 logických vývojových častí, ktoré boli následne implementované do vytvoreného GUI.

Prvá časť sa zaoberá prehrávaním kontinuálneho sínusového signálu cez externú zvukovú kartu pre moduláciu lasera, ktorá musí fungovať časovo neobmedzene. Táto časť predstavovala najväčšie problémy, pretože bolo potrebné zaručiť kontinualitu prehrávania bez prerušenia. To sa bežnými "Matlabovskými" prehrávačmi zvuku nedalo zabezpečiť.

V druhej časti sa pozornosť zameriava na následné kontinuálne nahrávanie stereo signálu z externej zvukovej karty. Základom je vytvorenie algoritmu nahrávania a následného zberu dát počas samotného nahrávania bez následného pretečenia buffera. Zároveň táto časť musí v pravidelných intervaloch spúšťať časť programu na spracovanie samotného signálu.

Tretia časť je zameraná na spracovanie nahratých signálov. Signál musí byť najskôr orezaný na pásmo v okolí modulačnej frekvencie, následne je nutné vyfiltrovať modulačnú frekvenciu a vypočítať aktuálnu výchylku. Základom je preto vhodné nastavenie pásmovej priepuste pre odfiltrovanie všetkých nežiaducich rušení.

Prehrávanie

Cieľom tejto časti je vytvorenie algoritmu pre prehrávanie kontinuálneho sínusu pomocou externej zvukovej karty pre modulovanie laseru. Modulovanie je nutné kvôli odstráneniu šumov pomocou amplitúdovej modulácie, pri podmienke zaručenia kontinuality prenosu bez výpadkov. Prerušenia by mohli viesť k problémom pri demodulácii a nepresnostiam pri výpočte polohy dopadajúceho laserového lúču na PSD detektor. Najskôr je nutné vytvoriť vektor dát pre prehrávanie:

```
fs=48000;                    %vzorkovacia frekvencia
f=12000;                     %modulačná frekvencia
a=0;                        %posunutie pre zosilnenie lasera
d = 1.0;                     %dĺžka (s)
n = fs * d;                  %počet vzoriek
s = (1:n) / fs;              %časy v jednotlivých vzorkách
y=sin(2*pi*f*s)+a;           %vektor vzoriek
```

Následne je možné spustiť prehrávanie:

```
player = audioplayer(y, Fs);     %vytvorenie objektu audioplayer
play(player);                    %spustenie prehrávania
```

Problém nastáva pri snahe obnovovať prehrávanie za účelom vytvorenia kontinuálneho signálu. Funkcia „audioplayer“ neobsahuje automatický repeat. Obsahuje síce možnosť nastavenia „TimerFcn“, teda funkcie, ktorá sa v pravidelnom cykle počas prehrávania spúšťa. V nej je však možné robiť iba obslužné výpočty a nie priamo zasahovať do prehrávania. Je to spôsobené tým, že funkcia „audioplayer“ nepodporuje zásah do prehrávacej fronty a pre opätovné spustenie prehrávania je nutné, aby pôvodné prehrávanie bolo ukončené. Z tohto dôvodu, dochádza pri tomto type prehrávania k nutným - nežiadúcim výpadkom medzi dvoma spusteniami funkcie. Tento problém je možné vyriešiť vytvorením dostatočne dlhého vektora vzoriek. Pri kontrole skontrolujeme stavu pamäte (uvedený príklad PC s 4GB pamäte):

```
>> memory
Maximum possible array:   2046 MB (2.145e+09 bytes)
Memory available for all arrays:   3360 MB (3.524e+09 bytes)
Memory used by MATLAB:    385 MB (4.033e+08 bytes)
Physical Memory (RAM):   4079 MB (4.277e+09 bytes)
```

zistíme, že maximálna veľkosť použiteľnej pamäte je $2{,}145*10^9$ bitov. Čo zodpovedá (pri 16-bitovej dĺžke jedného čísla a použitej vzorkovacej frekvencii 48kHz) približne 2793 sekundám prehrávaného záznamu. To je približne 46.54 minút. Tento čas je pre dlhodobejšie merania prikrátky a výsledný vektor má 134088000 prvkov. Tento vektor je príliš veľký a zaberá v podstate všetku voľnú pamäť pre ďalšie spracovanie. Z týchto dôvodov je nutné voliť cestu opakovaného prehrávania kratších vektorov vzoriek.

Tento problém je možné vyriešiť pomocou Data Acquisition Toolbox-u, ktorý je síce primárne zameraný na prácu s meracími kartami National Instruments, ale podporuje aj zvukové karty. Jedinou podmienkou je inštalácia 32-bitovej verzie Matlabu. Na 64-bitovej verzii nie sú podporované funkcie na prehrávanie a nahrávanie zvuku. Na prehrávanie signálu je potrebné najprv zistiť, pomocou funkcie „audiodevinfo“, identifikačné číslo príslušného výstupu na externej zvukovej karte.

```
ID = audiodevinfo(0,'Speakers (USB Multi-Channel Audio Device)
(Windows DirectSound)');
     %nula hovori ze ide o vystupne zariadenie,
     %druhy je nazov zariadenia
```

Pri vytváraní objektu pre analógový výstup zvolíme názov ovládača hardwarového adaptéra na „winsound“ pre audio výstup.

```
ao=analogoutput('winsound',ID);
```

Predpokladáme vysielanie stereo signálu, t.j. rovnaký pre ľavý a pravý kanál. Ľavý a pravý kanál sa nastavuje vektorom číselného vyjadrenia konkrétneho kanálu. Zároveň zvolíme požadovanú vzorkovaciu frekvenciu, v našom prípade 48kHz.

```
addchannel(ao, [1 2]);           %vytvorenie stereo výstupu
set(ao,'SampleRate',fs);         %nastavenie vzorkovacej frekvencie
```

Nastavíme trigger (spúšťač prehrávania) buď na hodnotu „Immediate“, kedy sa spustí spúšťač hneď po spustení prehrávania funkciou „start“, alebo na hodnotu „Manual“ kedy sa spúšťač rozbieha nezávisle od štartu objektu funkciou „trigger“:

```
set(ao,'TriggerType','Immediate');     %nastavenie trigger-a
```

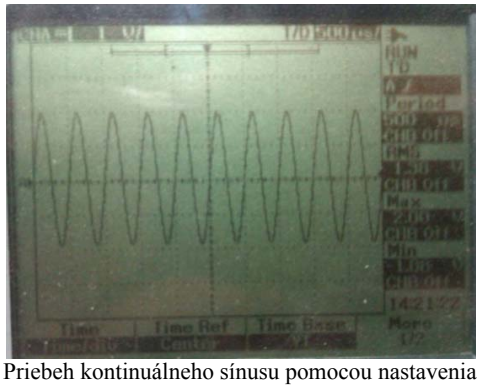
alebo

```
set(ao, 'TriggerType', 'Manual');      %nastavenie trigger-a
trigger(ao)                            %spustenie trigger-a
```

Následne sa dá postupovať dvoma rôznymi spôsobmi. Buď realizovať kontinualitu prehrávania pomocou nastavenia „RepeatOutput“ .Najväčšou nevýhodou je nutnosť preddefinovania počtu požadovaných opakovaní, čo niekedy vopred nemusíme vedieť:

```
set(ao, 'RepeatOutput', 4); %4x opakovanie, teda celkove 5 behov
```

Druhá možnosť je nastavením funkcie „TimerFcn“, ktorá zavolá funkciu pre naplnenie fronty prehrávania. Pri použití opakovaného prehrávania je nutné nastaviť trigger na hodnotu „Manual“. Pre nutnosť zadefinovania počtu opakovaní vopred, prípadnému následnému dodefinovaniu počtu opakovaní sme zvolili obnovovanie dát pomocou „TimerFcn“.Realizovať podobný výsledok môžeme aj nastavením vlastností "Samples Output FcnCount" analógového výstupného zariadenia. Nastavením volania funkcie na rovnakú dĺžku (ako tých, ktoré sú vo fronte prehrávania,) je takmer zaručené, že sa vyskytnú„presluchy“ vo výstupe.



Obrázok číslo 1: Priebeh kontinuálneho sínusu pomocou nastavenia „RepeatOutput“.

Jednou z možností je nastaviť "SamplesOutputFcnCount" na kratšiu dĺžku ako je dĺžka dát, ktoré sú vo fronte prehrávania, a to tak, že údaje treba vložiť do analógového frontu výstupného zariadenia pred tým ako sa front vyprázdni. To je možné urobiť nasledujúcim spôsobom:

```
set(ao,'SamplesOutputFcnCount',0.9*n);%0,9*počet vzoriek
```

Spoľahlivejšou možnosťou je použiť volanie nastavenia "TimerFcn". Podstata spätného volania „TimerFcn“ je, že len jeden z vektora dát je v danom okamihu vo fronte prehrávania. Kým volania nastavenia „SamplesOutputFcnCount“ zaručuje pri chode, že samotné prehrávanie bude trvať dlhšie, ako je frekvencia, s akou je volaná príslušná funkcia a nakoniec budú dáta vo fronte prehrávania čakať a spomaľovať samotné volanie, ale „TimerFcn“ je na naopak vyradená v prípade, že je aspoň jeden vektor dát v fronte prehrávania. Následne vďaka tomu môže „TimerFcn“ dosiahnuť trvalý analógový výstup bez prerušenia alebo pretečenia:

```
set(ao,'TimerFcn',{'data_pre_timerFcn',ao,data}); %zadefinovaniefunkcie
set(ao,'TimerPeriod',(0.9*f*d*(fs/f)));   %volanie po 90% periódach sínu
```

Treba poznamenať, že spätné volanie je spustené krátko pred vyprázdnením pamäte dát. Konkrétne po skončení 90% períód sínusu v danom definovanom čase. Je veľmi dôležité, aby bežiaci sa objekt „analogoutput“ nevyprázdnil predtým ako sa zastaví a zistí diskontinuitu medzi každým dátovým výstupom (ako je tomu v prípade „SamplesOutputFcn“). Príklad kódu ukazuje fungovanie funkcie pre naplnenie fronty dátami „data_pre_timerFcn“ (volajúcej v kóde vyššie):

```
if (ao.SamplesAvailable + length(data)) >= ao.MaxSamplesQueued
      return
end

putdata(ao, data);      %vloží dáta

if strcmp(ao.Running, 'Off')
      start(ao)         %reštart pri výpadku
end
```

V prvom kroku zistíme koľko vzoriek je stále vo fronte a koľko sme požadovali pridať do fronty. Ak je táto suma vyššia, než to, čo potrebujeme, vrátime sa bez viacerých dát vo fronte. Všimnite si, že s druhým prístupom, keď bude stáť vo fronte viac dát než povoľuje parameter „MaxSamplesQueued“, ktorý hovorí o maximálnej možnej dĺžke dát, ktorá môže byť vo fronte prehrávania, spôsobí funkcia „putdata“ blokovanie príkazového riadku. Preto, ak je priestor vo fronte, presúvame nové dáta do fronty a potom sa reštartuje objekt iba v prípade, že sa zastaví.

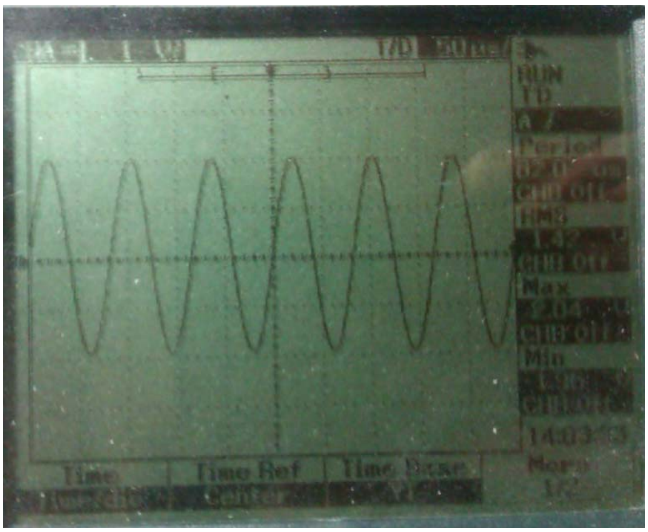
Po naparametrizovaní všetkých nastavení je nutné ešte pred spustením prehrávania vložiť do fronty prehrávania prvý vektor dát. Následne spustíme prehrávanie:

```
putdata(ao,data);       %vloženie dát
start(ao)               %spustenie
```

Po skončení potrebného prehrávania vypneme prehrávanie objektu „analogoutput“ a vymažeme ho:

```
stop(ao)          %zastavenie
delete(ao)        %vymazanie
```

Pomocou tohto programu je možné dosiahnuť kontinuálne prehrávanie vektora dát, sínusu, v ľubovoľnej dĺžke bez nutnosti ďalších zásahov do behu programu. Zároveň jeho nasledovná implementácia do grafického rozhrania nie je zložitá. Správnosť prehrávaných dát bola overená aj na osciloskope pripojenom na výstupe zo zvukovej karty. Presluchy neboli už ani počuteľné, ani viditeľné na osciloskope ako v predošlých prípadoch.



Obrázok číslo 2: Priebeh kontinuálneho sínusu pomocou nastavenia „TimerFcn“.

Nahrávanie

Tak ako pri prehrávaní signálu, aj pri nahrávaní môžeme využiť funkciu „audiorecorder“. Avšak hneď na začiatku narazíme na obmedzenia tejto funkcie. Najväčšou slabinou je nemožnosť získavania nahratých dát počas samotného nahrávania, čím sa vlastne znemožňuje spracovanie signálu počas nahrávania a teda samotný on-line beh programu:

```
r = audiorecorder(48000, 16, 2);      %vytvorenie objektu
record(r);                            % spustenie nahravania
for i=1:4
     pause(1);                        %nahratie 4 sekund
end
stop(r);                              %zastavenie nahravania
s = getaudiodata(r);                  %vybratie nahratych dat
```

Problém znovu rieši Data Acquisition Toolbox s jeho možnosťami. Na prehrávanie signálu zistíme pomocou funkcie „audiodevinfo“ identifikačné číslo príslušného vstupu na externej

zvukovej karte, rovnako ako pri prehrávaní. V zariadení môžeme využiť štandardný vstup Line In audiosystému, ktorého názov nájdeme v štruktúre vstupov a pred definovaním objektu pre nahrávanie overíme jeho identifikačné číslo. Do parametrov pre funkciu je nutné taktiež zadať číslo 1, ktoré určuje, že sa jedná o audio vstup:

```
id=audiodevinfo(1,'Line In (USB Multi-Channel Audio Device) (Windows
DirectSound)')
```

Pri vytváraní objektu pre analógový vstup zvolíme názov ovládača hardwarového adaptéra na „winsound“ pre audio výstup, rovnako ako pri výstupe.

```
ai = analoginput('winsound',id);
```

Predpokladajme príjem stereo signálu, t.j. rozdielny pre ľavý a pravý kanál. Ľavý a pravý kanál sa nastavuje vektorom číselného vyjadrenia konkrétneho kanálu. Ľavý kanál nám poskytuje (v nami skúmanom prípade) informáciu o rozdielovej zložke signálov. Pravý kanál zase vypovedá o invertovanej súčtovej zložke signálov. Zároveň zvolíme požadovanú vzorkovaciu frekvenciu, pre naše účely sme zvolili frekvenciu 48kHz:

```
addchannel(ai, [1 2]);
set(ai, 'SampleRate', fs);
```

Následne nastavímeatribút „SamplesPerTrigger“, ktorý určuje počet vzoriek, ktorý sa získa pre každý analógový vstupný kanál skupiny pre každú aktivačnú udalosť, ktorá nastane. Ak je nastavená „SamplesPerTrigger“ na parameter „Inf“, potom analógový vstupný objekt neustále získava dáta z nahrávacej fronty, kým nie je funkcia zastavená alebo pokiaľ nedôjde k chybe, čo je pre náš neustály záznam najnevhodnejšie:

```
set(ai,'SamplesPerTrigger',Inf); %nastavenie na nekonečné nahrávanie
```

Nastavením vlastnosti „SamplesAcquiredFcn“ analógového vstupného objektu dosiahneme možnosť pravidelne sledovať údaje vykonaním definovanej funkcie. Priradíme funkciu vlastnosti „SamplesAcquiredFcn“ ako parameter, ktorý sa bude vykonávať zakaždým, keď uplynie doba stanovená v parametri „SamplesAcquiredFcnCount“. V tomto prípade nechávame možnosť nastaviť „SamplesAcquiredFcnCount“ podľa aktuálnych požiadaviek ako súčin požadovanej periódy opakovania a vzorkovacej frekvencie, automaticky na 48000 vzoriek. Samotné volanie parametra „SamplesAcquiredFcn“ je nastavené na funkciu „volanie_timerFc“, ktorá spravuje samotné čítanie signálov a prípadné ďalšie spracovanie nahratých signálov:

```
set(ai,'SamplesAcquiredFcnCount',n);
set(ai,'SamplesAcquiredFcn',{@volanie_timerFcn,handles});
```

Táto funkcia má za úlohu získať dáta z nahrávacej fronty nahrávania, prípadne uložiť dáta do súboru. Za účelom ďalšieho spracovania dát, ich ukladá do štruktúry tvorenej časom, ľavým a pravým kanálom, ktoré získa pomocou funkcie „getdata“. Výsledný vektor dát je potom druhotne spracovávaný. Zároveň sa vypočíta maximálna možná dĺžka vektora dát a kontroluje sa, či nebude prekročená. Vlastnosť „SamplesAvailable“ vracia počet zaznamenaných vzoriek od posledného volania funkcie „getdata“:

```
persistent totalData;            %premenná zostáva v pamäti

[uV sV] = memory;
```

```
maxLengthArray=uV.MaxPossibleArrayBytes/16; %maximálna dĺžka vektora

[data,time] =getdata(ai, ai.SamplesAvailable); %uloženie dát

if(isempty(totalData) || ((3*d*48000)>maxLengthArray))
    totalData.time = time;
    totalData.data =data;
else
    totalData.time = [totalData.time;time];
    totalData.data = [totalData.data;data];
end
```

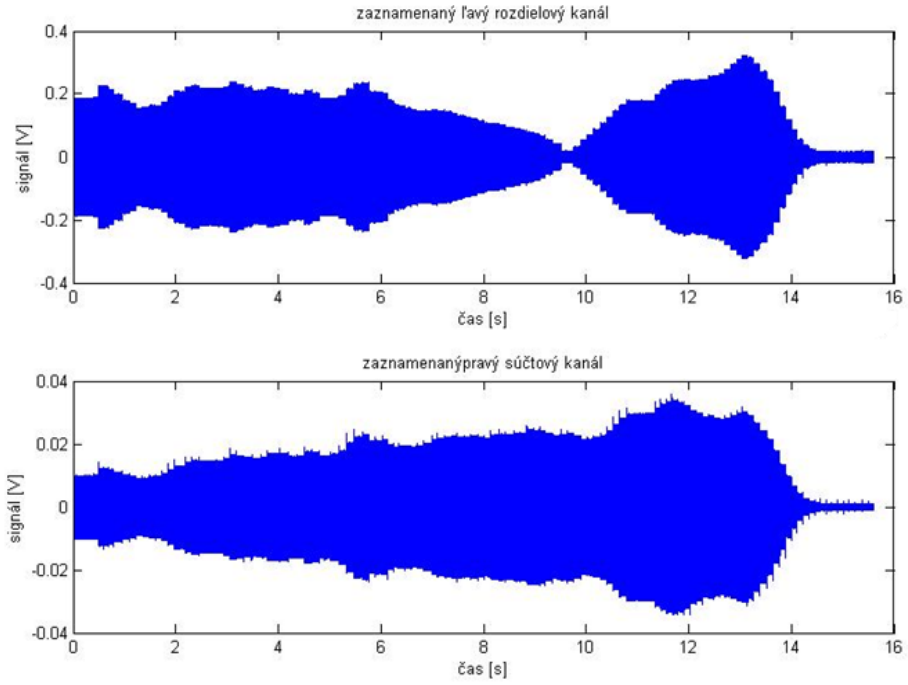
Po nastavení všetkých parametrov a zadefinovaní funkcie na spracovanie dát sa môže spustiť samotné nahrávanie signálu.

```
start(ai)              %spustenie
```

Po skončení potrebného nahrávania vypneme nahrávanie objektu „analoginput“ a vymažeme ho:

```
stop(ai)          %zastavenie
delete(ai)        %vymazanie
```

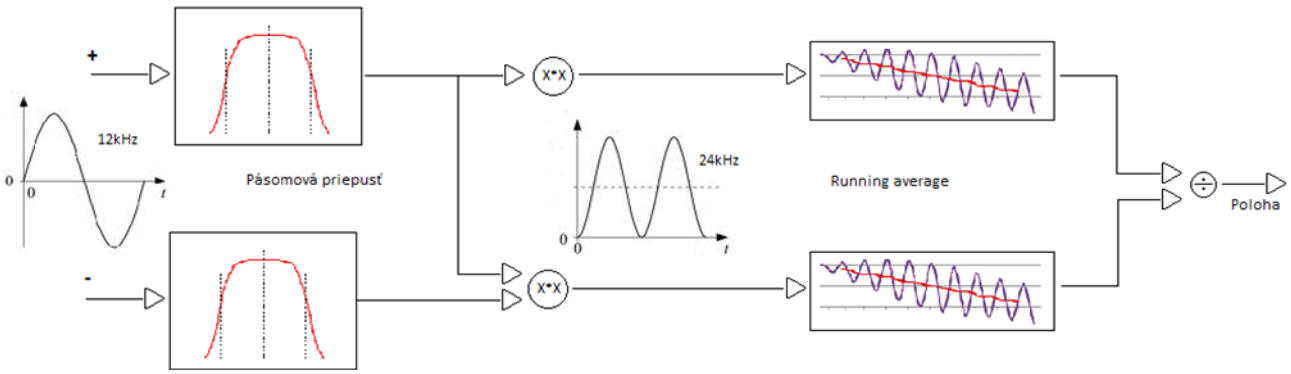
Takto zadefinovaným nahrávaním signálu z PSD detektora, (vo všeobecnosti iného detektora zvuku) pomocou externej zvukovej karty, dostávame kompaktný vektor dát k ďalšiemu spracovaniu. Jediným obmedzením je veľkosť pamäte, ktorá v našom prípade dovolovala nahrať záznam dlhý približne 15,51 minúty. Tento čas sa dá prípadne predĺžiť ukladaním dát do súboru. Program v opačnom prípade po dosiahnutí maximálnej dĺžky vektora, začne dáta automaticky zapisovať odznova.



Obrázok číslo 3: Nahraté signály zo zvukovej karty pri prechode celým rozsahom detektora.

Spracovanie dát

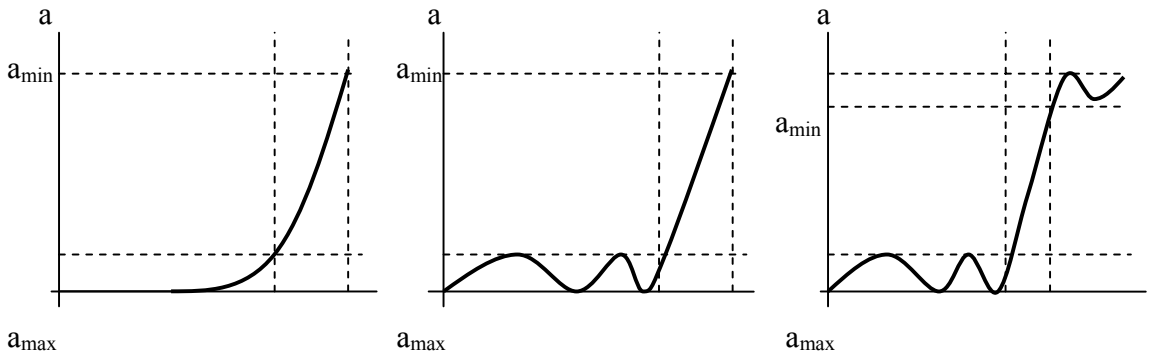
Pod spracovaním dát sa myslí spracovanie nahratých signálov na výslednú polohu dopadajúceho svetelného lúča na PSD detektor. V prvom kroku je nutné oba signály, či už od súčtovej zložky, alebo rozdielovej, demodulovať. Následne môžeme vyjadriť polohu lúča. Nie však ako je uvádzaný všeobecný vzťah pre výpočet polohy dopadajucého lúča na PSD detektor, pretože v nami navrhnutej elektronickej časti zariadenia pre spracovanie signálu sú odlišné zosilnenia ako pri všeobecnom návrhu. V tomto prípade sa jedná iba o podiel samotnej demodulovanej rozdielovej a súčtovej zložky.



Obrázok číslo 4: Schéma spracovacej časti signálu v programe.

Samotný výpočet demodulácie sa realizuje v troch krokoch. V prvom kroku do výpočtu vstupujú dva signály, rozdielový a súčtový, ktoré sú modulované sínusom o frekvencii 12kHz a na ktorom sú naakumulované samotné signály. Tieto vstupné signály sú následne filtrované cez pásmovú priepusť.

Pásmová priepusť je typ filtra, ktorý prepúšťa zložky signálov medzi dolnou medznou frekvenciou f_L a hornou medznou frekvenciou f_H. To znamená, že všetky ostané signály s frekvenciami nižšími ako f_L a vyššími ako f_H sa prenášajú s požadovaným útlmom . V ideálnom prípade je útlm čo najväčší a prenos sa preto rovná nule. V princípe ide o spojenie dolno a horno priepustného filtra. Súčiastky sú vyberané tak, aby medzná frekvencia hornopriepustného filtra bola nižšia ako medzná frekvencia dolnopriepustného filtra. Frekvencia $f_1<f_H<f_L$ môže prejsť cez dolnopriepustný filter, ale je potlačená hornopriepustným filtrom. Frekvencia $f_2>f_L>f_H$ môže prejsť hornopriepustným filtrom, ale nedokáže prejsť cez oblasť s charakteristikou dolného priepustu. Frekvencia f_{center}, ktorá je skoro v strede, prejde oboma filtrami bez väčších zmien. Samotné filtre sú prvky, ktoré v určitom frekvenčnom pásme prepúšťajú signál s minimálnym útlmom, zatiaľ čo v inom frekvenčnom pásme majú veľký útlm a signál neprepúšťajú. Podľa tvaru útlmovej charakteristiky rozoznávame filtre s maximálne plochou útlmovou charakteristikou, (Butterwortove filtre, obr.5a), s izoextremálnou útlmovou charakteristikou v priepustnom pásme (Čebyševove filtre, obr.5b), s izoextremálnou útlmovou charakteristikou v priepustnom i v nepriepustnom pásme (eliptické filtre, obr.5c). [1 ,2]



Obrázok číslo 5: Útlmové charakteristiky filtrov. [1]

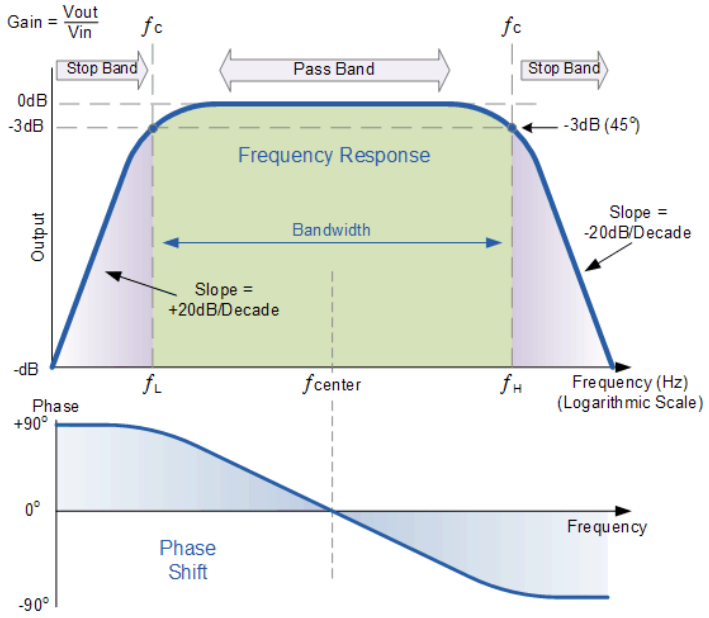
Základné delenie filtrov je na filtre typu FIR a IIR. FIR filtre sú číslicové filtre s konečnou impulzovou charakteristikou. Nemajú obdobu v analógovej technike, sú produktom číslicového spracovania signálov. Prenosovú funkciu $H(z)$ lineárneho diskrétneho FIR filtra N-tého rádu môžeme zapísať vo všeobecnom tvare:

$$H(z) = b_0 + b_1 z^{-1} + \ldots + b_N z^{-N} \qquad (1.1)$$

FIR filtre je možné vždy realizovať nerekurzívnou štruktúrou, v niektorých špecifických prípadoch však zavedenie spätnej väzby zjednodušuje výpočet. Prednosťou FIR filtrov je možnosť realizácie lineárnej fázovej frekvenčnej charakteristiky a stabilita. Nevýhodou FIR filtrov môže byť vysoký rád filtra potrebný na splnenie frekvenčnej špecifikácie, s čím súvisí nárast výpočtovej zložitosti a oneskorenia výstupného signálu. IIR filtre sú filtre s nekonečnou impulzovou charakteristikou. Pre ich realizáciu je nutné použiť štruktúry so spätnou väzbou. Prenosovú funkciu $H(z)$ lineárneho diskrétneho IIR filtra M-tého môžeme zapísať vo všeobecnom tvare:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + \ldots + a_M z^{-M}} \qquad (1.2)$$

kde platí $M > 0$ a $a_M \neq 0$. Z hľadiska výpočtovej zložitosti sú filtre IIR podstatne výhodnejšie než FIR, pretože požiadavky na frekvenčnú charakteristiku možno splniť pri nižších rádoch filtra. Nevýhodou je však možnosť nestability a nelinearita fázovej frekvenčnej charakteristiky. Návrh číslicových IIR filtrov je obvykle založený na technike návrhu analógových filtrov a ich následnej transformácie na diskrétnu sústavu. Prevod do diskrétnej časovej oblasti možno uskutočniť viacerými metódami, medzi ktoré patria invariancia impulzovej charakteristiky, metóda spätných alebo dopredných diferencií, bilineárna transformácia. Z uvedených metód sa najčastejšie používa metóda bilineárnej transformácie, ktorá zaručuje zachovanie stability a imaginárna os p-roviny je jednoznačne mapovaná na jednotkovú kružnicu v z-rovine. Pri návrhu sa zvyčajne obmedzujeme na štyri základné typy filtrov – DP, HP, PP a PZ filtre, pričom je potrebné zohľadniť nelineárne mapovanie medzi frekvenciami analógovej a diskrétnej sústavy. Kauzálny stabilný IIR filter nemôže mať lineárnu fázovú charakteristiku, čo je nevýhodné pre spracovanie biosignálov vyžadujúcich zachovanie tvaru (napr. EKG, evokované potenciály).



Obrázok číslo 6: Pásmová priepusť[3]

Ak nevyžadujeme aby spracovanie signálu prebiehalo v reálnom čase, môžeme dosiahnuť nulovú fázovú charakteristiku prostredníctvom priameho a spätného behu IIR filtra. Tento tzv. obojsmerný IIR filter je možné realizovať pomocou dvoch kauzálnych filtrov, pričom signál pred vstupom do druhého otočíme v čase. Ekvivalentná frekvenčná prenosová funkcia je potom kvadrátom modulovej charakteristiky jedného filtra, fázové charakteristiky sa v dôsledku časovej reverzie signálu vykompenzujú. Obojsmerný Butterworthov PP filter je napr. štandardne odporúčaný pri spracovaní spriemerovaného EKG signálu za účelom detekcie oneskorených komorových potenciálov. [4, 5]

Z tohto dôvodu budeme voliť Butterworthov filter, zároveň pôjde o filter typu IIR pre jeho menšiu výpočtovú náročnosť. Pri parametrizovaní pásmovej priepusti v Matlabe treba myslieť na to či sa jedná o návrh IIR alebo FIR filtra. Podľa toho sa vyberajú konkrétne parametre, ktoré sa nastavujú. Ako prvý podstatný parameter je stupeň filtra. Čím väčší je zvolený stupeň filtra, tým viac sa jeho charakteristika podobá ideálnej pásmovej priepusti obdĺžnikového tvaru. Samozrejme so zvyšujúcim sa stupňom sa zväčšuje aj výpočtová náročnosť a teda aj čas spracovávania. Najviac sa osvedčil filter desiateho rádu. Poskytoval dobrý pomer kvality filtrovania a rýchlosti. S narastajúcim stupňom sa už čas predlžoval neprimerane k získanému zlepšeniu kvality filtrácie. Ďalšími dôležitými parametrami je ohraničenie prepúšťaných frekvencií pri ktorých dochádza k potlačeniu signálu o medzný útlm 3dB. Volili sme symetrické orezanie dát okolo nosnej frekvencie 12kHz s prednastaveným pásmom ± 500Hz. Ako posledná hodnota sa nastavuje vzorkovacia frekvencia. Ostatné nastavenia možné pre IIR filter ako nezávislý rád menovateľa a čitateľa prenosovej funkcie filtra alebo spodná frekvencia hrany ohraničenia priepusti, nám pri pokusoch s týmito nastaveniami neprinášali žiadne ďalšie zlepšenie filtrovaného signálu. Následné definujeme pásmovú priepusť v Matlabe ako:

```
fs=48000;
fz=12000;
fpas=500;
st = 10;
d = fdesign.bandpass('N,F3dB1,F3dB2',st,(fz-fpas),(fz+fpas),fs);
      %priepust
```

Ďalej vytvoríme samotný Butterworthov filter.

```
Hd1 = design(d,'butter');  %tvorba filtra s definovanými parametrami
```

Ďalej musíme samotný signál prefiltrovať pomocou vytvoreného filtra.

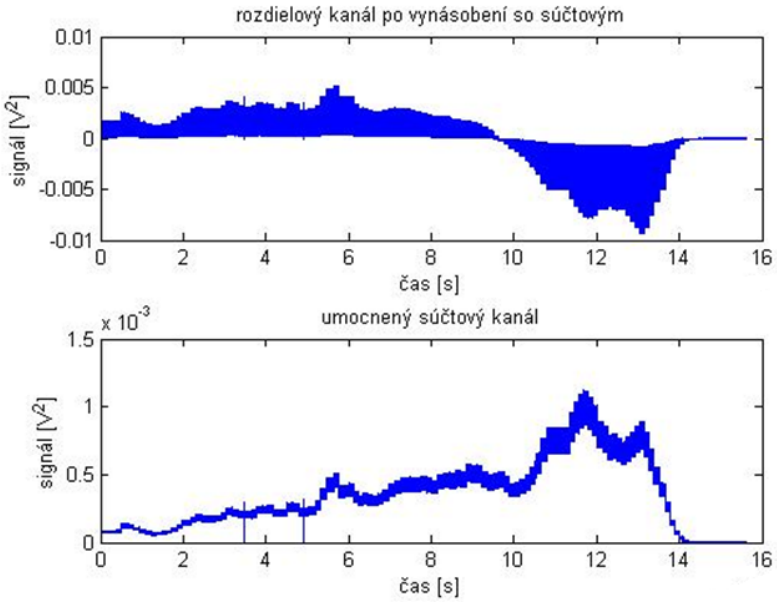
```
signal2 = filter(Hd1, signal1);  %filtrácia prvého signálu
```

Po prefiltrovaní nahratého signálu dostávame signál, ktorý je orezaný od "parazitných" šumov vyšších a nižších frekvencií a tak dostávame vyčistený signál nosnej frekvencie so zaznamenanými dátami v medziach priepustnej frekvencie s určitým šumom.

V ďalšom kroku spracovania signálu je nutné orezať nepotrebnú symetrickú časť signálu. Vstupuje nám už orezaný filtrovaný signál, ktorý následným prenásobením orežeme. Ako je vidno z obrázku č.3, signál pre pravý aj ľavý kanál je symetrický, čo je spôsobené samotným nahrávaním. Následným roznásobením dosiahneme odstránenie symetrických častí signálu a zostanú tak iba časti, ktoré sú nosičmi podstatnej (cieľovej) informácie. Pre súčtový kanál stačí, keď ho umocníme na druhú, teda vynásobíme súčtový signál so samím sebou. Pri rozdielovej zložke to už také jednoduché nie je. Dochádza pri nej totiž k zmene znamienka pri prechode nulou z jednej strany PSD detektora na druhú. Samotným umocnením by sme stratili túto znamienkovú zmenu. Z tohto dôvodu budeme záporný kanál násobiť so súčtovým. Vďaka tomu neprídeme o znamienkovú zmenu a násobenie informácie o polohe s informáciou o intenzite nepokazí samotný výsledný signál. Táto úprava sa aj tak pri neskoršom vzájomnom delení súčtového a rozdielového kanála odstráni.

```
signal11=signal1.*signal1;  %súčtový
signal12=signal1.*signal2;  %rozdielový
```

Výsledkom tejto úpravy je orezaný signál, so správnou znamienkovou hodnotu s nosnou frekvenciou. Vrcholy, ktoré je možné pozorovať v približne 3,5s a 5s sú spôsobené výpadkom laserového lúča. Vďaka tomu v týchto bodoch bolo zariadenie bez signálu a pôsobil iba šum, čo viedlo k vzniku tejto chyby merania.



Obrázok číslo 7: Prenásobený signál.

V nasledujúcom kroku spracovania signálu demodulujeme samotný nosný modulačný sínusový signál, ktorý má teraz už po prenásobení signálov frekvenciu 24kHz. Na odstránenie použijeme posuvný priemer „running average“. Ide o priemer s plávajúcim oknom definovaným určitým počtom vzoriek. To znamená, že tento priemer si vždy zoberie určitý, definovaný počet vzoriek, z nich vytvorí priemer a posunie sa zase ďalej. Napríklad pre dĺžku okna 5 bude prvých pár vzoriek vyzerať:

$$yy(1) = y(1)$$
$$yy(2) = (y(1) + y(2) + y(3))/3$$
$$yy(3) = (y(1) + y(2) + y(3) + y(4) + y(5))/5$$
$$yy(4) = (y(2) + y(3) + y(4) + y(5) + y(6))/5$$
$$...$$

Tento princíp odstraňovania nosnej frekvencie využíva fakt, že stredný priemer určitého množstva celých periód sínusu je vždy nulový. Je to dané jeho symetrickosťou pol periód, ktoré sú rovnako veľké, len s opačným znamienkom, čo spôsobuje ich vlastnú kompenzáciu v priemere. Samotný posuvný priemer je v Matlabe možné realizovať dvoma spôsobmi. Buď pomocou funkcie „smooth“ alebo vytvorením filtra. Pri samotnej funkcii stačí zadať vektor hodnôt, ktoré sa majú spriemerovať a dĺžka okna. Ako posledný je typ priemeru (v základe je nastavená na „moving“).

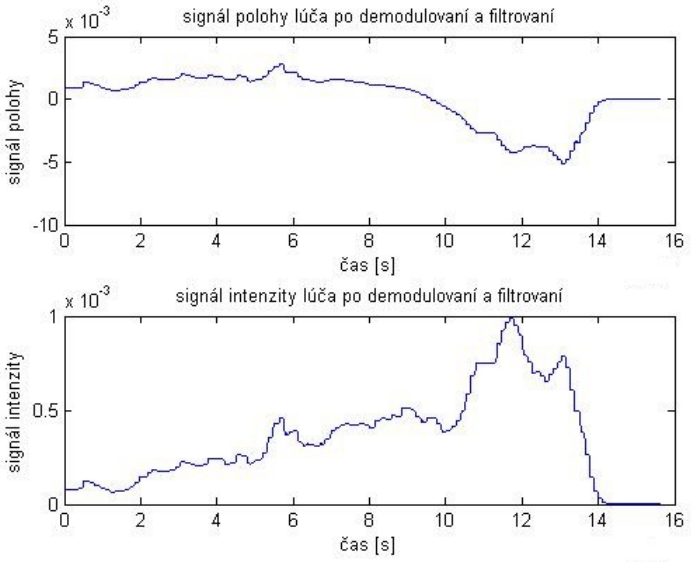
```
signal21=smooth(signal11, fz*2, 'moving');  %priemer súčtovej zložky
signal22=smooth(signal12, fz*2, 'moving');  %priemer rozdielu
```

Filter v úlohe posuvného priemeru sa parametrizuje:

```
filter(ones(1,dlzkaOkna)/dlzkaOkna,1,data);
```

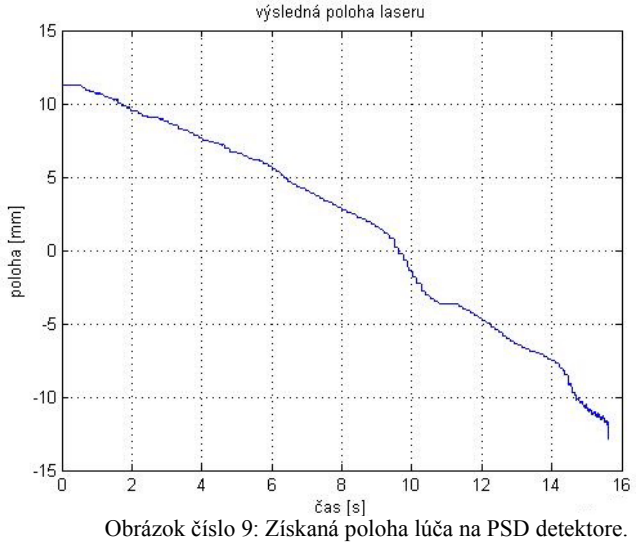
Výsledkom po spriemerovaní signálu by mala byť už mierne zašumená informácia z ľavého rozdielového kanála a pravého súčtového kanála. V princípe po tomto výpočte je už možné rátať výslednú polohu laserového lúča. Ako je vidno na obrázku č.8 , touto úpravou signálu sme sa zbavili aj chýb spôsobených výpadkom lasera. Zároveň je na obrázku č.8 vidno (na grafe signálu polohy) ako sa laser zapol približne pred 1. sekundou, následne od 1. po 14. sekundu dochádza k prechodu lúča cez celú dĺžku PSD detektora, čo je vidieť vďaka postupne klesajúcej rozdielovej zložke a stúpajúcej súčtovej zložke. Medzi 9 a 11 sekundou dochádza k prechodu nulou, teda stredom detektora, s najstrmším priebehom. Ďalej sa poloha ustáli okolo 11 sekundy a potom ďalej klesá. Od 14 sekundy dochádza k okrajovým šumom a následne k vypnutiu lasera. Na koniec ešte prefiltrujeme oba signáli cez dolnopriepustný filter za účelom odstránenia prípadných šumov, ktoré mohli vzniknúť aj počas spracovávania signálu, aj ktoré prešli pásmovou priepusťou. Signál filtrujeme na rovnakú frekvenciu, ktorú sme používali ako pásmo priepusnosti pri pásmovej priepusti. Iná frekvencia nemá veľké opodstatnenie. V Matlabe definujeme dolnopriepustný filter podobne ako pásmovú priepusť.

```
D = fdesign.lowpass('N,Fc',st,fpas,48000);  %parametrizovanie filtra
Hd = design(D);                             %vytvorenie filtra
signal31=filter(Hd,signal21);               %filtrovanie signálu
```



Obrázok číslo 8: Získané demodulované signály z detektora po filtrovaní.

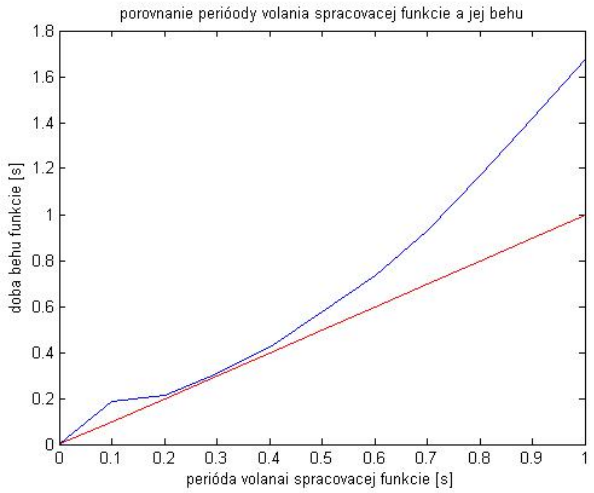
Na záver stačí podeliť rozdielový, polohový, signál súčtovým, čoho výsledkom je aktuálna poloha lasera na PSD detektore. Poloha sa mení schodovito, čo je spôsobené krokom krokového motora, ktorý automaticky prechádzal cez celú dĺžku detektora. Poloha je zosilnená približne dva krát pôsobením spracovacej elektroniky. Pri porovnaní obrázku a predpokladaného vývoja pozície na základe demodulovaných signálov vyššie dochádzame k zhodným výsledkom.



Obrázok číslo 9: Získaná poloha lúča na PSD detektore.

Ako je vidno aj na obrázku č.9 , po spracovaní nahratých signálov vieme relatívne presne získať pozíciu lasera. Problém nastáva v snahe implementovať spracovanie signálu do on-line nahrávania. Samotné spracovanie je výpočtovo, aj napriek snahám zabezpečiť čo najmenšiu náročnosť algoritmu, stále dosť pomalé. Dôsledkom toho vznikla nemožná zhoda medzi periódou volania spracovacej funkcie a dĺžkou samotného behu spracovacej funkcie. Napríklad pri perióde volania spracovacej funkcie 48000 vzoriek, čo je presne jedna sekunda záznamu, trvá samotné spracovanie 1,6771 sekundy. Tento čas bol nameraný na rýchlom 3,6GHz procesore AMD FX. Pri použitý slabšieho 1,66GHz procesora Intel Atom sa čas predĺžil až na 5,4 sekundy. Zároveň je nutné uvedomiť si, že ide čisto o výpočtový čas spracovania dát bez

ďalšieho predlžovania v reálnej aplikácii s nutnosťou volania spracovacích funkcií, prekresľovaním grafov, prípadne ukladaním do súboru. Následné zvyšovanie alebo znižovanie volacej periódy nemá za dôsledok pokles výpočtového času pod samotnú periódu volania. Zároveň na základe realizovaných pokusov je nutné uvedomiť si, že spracovacia funkcia nie je volaná v nezávislom vlákne ako paralelný výpočet. Vďaka tomu, pokiaľ je výpočtový čas menší ako perióda volania, ale výpočtový čas funkcie sa blíži k perióde volania, spomaľuje sa samotný nahrávací algoritmus. Na obrázku je vidno ako so zvyšujúcim sa počtom vzoriek na spracovanie, rastie aj čas potrebný na ich spracovanie. Zároveň je vidno, že neexistuje perióda, pri ktorej by bolo možné on line meranie realizovať.



Obrázok číslo 10: Porovnanie periódy volania a trvania spracovacej funkcie(červená je perióda volania funkcie, modrá sú namerané časy behu spracovacej funkcie s vektormy vzoriek dannej dĺžky).

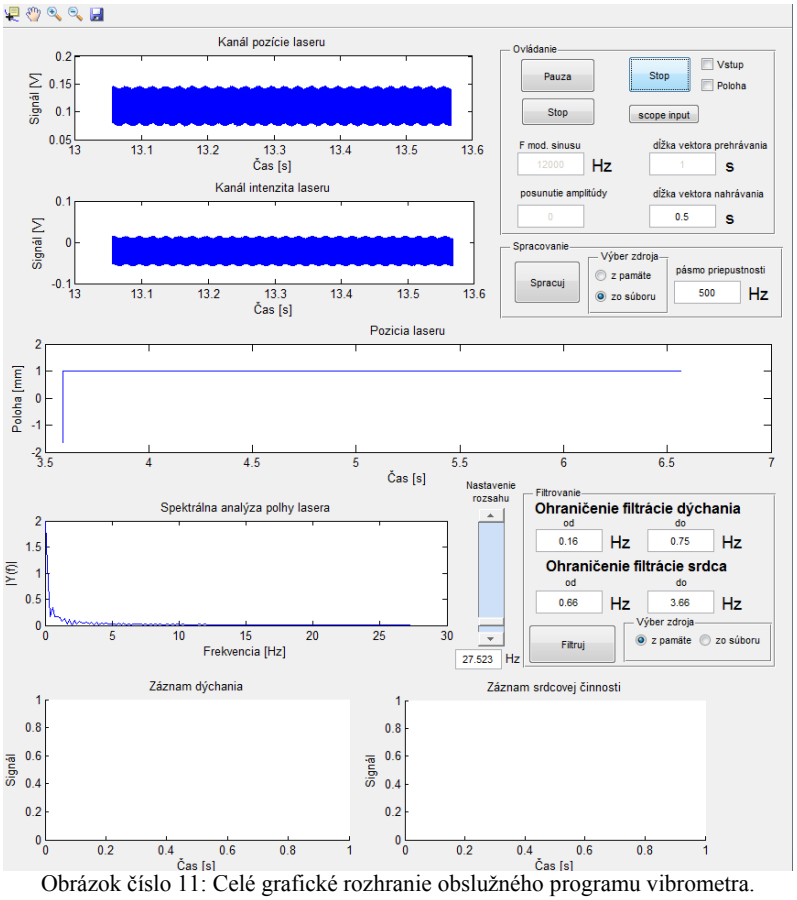
Na základe týchto informácii sme usúdili, že budeme realizovať nezávisle nahrávanie zvuku a následne potom spracovanie samotných dát. S týmto problémom príliš dlhého času spracovania signálu a nemožnosti paralelného behu spracovacej funkcie sme sa obrátili aj na Jaroslava Jirkovského, ktorý je aplikačným inžinierom vo firme HUMUSOFT s.r.o.. Pomocou testov sme si uvedomili, že kameňom úrazu je dvojité volanie funkcie „smooth“, respektíve filtrácia s dlhým „moving average“ filtrom, ktorý je touto funkciou volaný. Z percentuálneho hľadiska zaberá volanie funkcií„smooth“ 98.8 % celého algoritmu. Preto nám poradil, že pokiaľ chceme algoritmus urýchliť, musíme sa sústrediť na túto funkciu. Vzhľadom na to, že funkcia „smooth“ je volaná 2 krát a volania sú nezávislé, môžeme ju volať paralelne s rôznymi dátami pomocou funkcie „spmd“ nachádzajúcej sa v „Parallel Computing Toolboxu“. Algoritmus by potom vyzeral takto:

```
matlabpool open 2     %spustenie dvoch workerov pre paralelný výpočet
signal = [signal11,signal12];         %tvorba dát

spmd
    C=smooth(signal(:,labindex), fz*2,'moving'); %paralelný výpočet
end

signal21=C{1};        %rozdelenie dát
signal22=C{2};
```

Tento krok nám prinesie približne 45% úspory času, čím sa dostávame na hodnotu približne 1,1 sekundy. Čo je v našom prípade stále veľa a pre kratšie vektory dát je úspora minimálna. Druhou možnosťou je urýchlenie samotnej funkcie „smooth“. Funkcia „smooth“ v sebe obsahuje funkciu „filter“ s „moving average“ filtrom (funkcie v MATLABe môže užívateľ zobraziť a editovať príkazom „edit“). Funkciu „smooth“ by bolo možné nahradiť funkciou „filter“ s „moving average“ filtrom a drobnou úpravou signálu by potom bolo možné urýchliť funkciu „filter“. Poslednou možnosťou je urýchliť funkciu „filter“ pomocou funkcií pre GPU výpočty, ktoré sú súčasťou „Paralel Computing Toolboxu“. Tie môžu spracovávať výpočty na grafických kartách nVidia s architektúrou CUDA. Funkcia „filter“ je jednou z funkcií, ktorú pri volaní so vstupmi v dátovom type „gpuArray“, prebiehajúcich na GPU. Týmto by mohlo dôjsť k urýchleniu samotnej filtrácie. Opis oboch týchto navrhovaných postupov je už mimo obsahových možností tohoto článku.



Obrázok číslo 11: Celé grafické rozhranie obslužného programu vibrometra.

Záver

V článku sú stručne opísané jednotlivé najdôležitejšie časti programu pre ovládanie vibrometra a spracovania získaných dát, pričom je pozornosť zameraná na problémy a ich riešenia, ktoré sa vyskytli pri tvorbe programovej časti. Zároveň článok poukazuje na zistené obmedzenia a načrtáva možné riešenia. Pri prehrávaní kontinuálneho sínusu, nahrávaní signálov z vibrometra a samotnom spracovaní dát opisuje dôležité príkazy v Matlabe na ich realizáciu. Najväčšie ťažkosti pri tvorbe programu boli s vytvorením kontinuálneho sínusu na výstupe zvukovej karty. Na vyriešenie tohto problému bol použitý toolbox Data Acquisition. Pri nasledovnom vývoji programu sa vyskytol problém on-line spracovania údajov, v podobe nízkej

výpočtovej rýchlosti nutného spracovacieho algoritmu. V článku sú navrhnuté možné opatrenia pre odstránenie tohto problému, ale na ich aplikáciu je nutný ďalší vývoj. Z tohoto dôvodu bol dokončený spracovací program, aj keď bez možnosti on-line spracovania údajov. Prínos tohto článku je preto hlavne v navrhnutí a odladení programu v Matlabe pre prácu so zvukovou kartou ako AD/DA prevodníka a ozrejmení ako aj práce s frekvenčnými filtrami v Matlabe.

Literárne zdroje

[1] Ing. Víťaz I., CSc: Meranie na filtroch, Meranie v telekomunikáciách 1, Návody na laboratórne cvičenie [online]. Dostupné na internete: < http://www.lm.uniza.sk/dulik/mvt/LC_03_Filtre.doc >

[2] Bc. Ján Sliacky, 2006. Preladiteľný prenosový kanál: Diplomová práca. Žilina: ŽILINSKÁ UNIVERZITA V ŽILINE, Elektrotechnická fakulta, Katedra telekomunikácií, 2006. 70 s.

[3] Band Pass Filter. Dostupné na internete: <http://www.electronics-tutorials.ws/filter/filter_4.html>

[4] Púčik,J.-Cocherová,E.2006. ANALÝZA BIOSIGNÁLOV. Bratislava: Slovenská technická univerzita v Bratislave, Fakulty elektrotechniky a informatiky. 2006. 124 s. ISBN 978-80-227-2833-1

[5] Metódy číslicového spracovania signálov v systémoch zberu dát. Dostupné na internete: <http://www.kemt.fei.tuke.sk/predmety/KEMT111_MPM/_materialy /DAQ_4.pdf>

[6] Zaplatílek,K.-Doňar,B.2006. MATLAB – začínáme se signály. Praha: BEN technická literatura 2006. 272 s. ISBN 80-7300-200-0

[7] Ondráček,O-Púčik,J.-Cocherová,E-Gašparík,I.2006. Číslicové spracovanie signálov BIOSIGNÁLY. Bratislava: Slovenská technická univerzita v Bratislave. 2006. 313 s. ISBN 978-80-227-2695-5

[8] Ondráček,O.2008. Signály a sústavy. Bratislava: Slovenská technická univerzita v Bratislave. 2008. 341 s. ISBN 978-80-227-2956-7

[9] MathWorks, Documentation Center [online]. Dostupné na internete: <http://www.mathworks.com/help/documentation-center.html>

[10] Bc. Marek Vnuk, 2013. Inteligentná budova – ochrana zdravia: diplomová práca. Bratislava: Slovenská technická univerzita v Bratislave Fakulta elektrotechniky a informatiky, 2013. 111 s.