UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN INFORMACIJSKE TEHNOLOGIJE
KOPER

IZVEDBA POSTOPKA ZA SPROTNO PRIKAZOVANJE SPEKTROGRAMA V ODPRTOKODNEM SISTEMU SPHINX-4

(Audio toolkit for displaying spectrogram in real time in the open-source system Sphinx-4)

ZAKLJUČNA NALOGA

Ime in priimek: Irman Abdić
Študijski program: Računalništvo in informatika, 1. stopnja
Mentor: doc. dr. Janez Žibert

Koper, december 2011

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN INFORMACIJSKE TEHNOLOGIJE
KOPER

Famnit
Fakulteta za matematiko, naravoslovje
in informacijske tehnologije

Zahvala

Zahvaljujem se staršem, prijateljem in sodelavcem za podporo. Posebna zahvala je namenjena mentorju dr. Janezu Žibertu in dr. Nickolayu V. Shmyrevu za uspešno in aktivno sodelovanje med pisanjem zaključne naloge.

Povzetek

Za izvedbo postopka sprotnega prikazovanja spektrograma smo uporabili ogrodje odprtokodnega sistema Sphinx-4, ki smo ga raziskali, ter določili pomen spektrograma, ga definirali ter izpostavili omejitve. Opisali smo, kako se izvede Diskretna Fourierjeva transformacija (DFT) ter izvedba Fourierjeve transformacije, v nadaljevanju FT s postopkom hitre FT (FFT). Pri analizi smo izvedli primerjavo našega programa, ki smo ga poimenovali iSound, ter programa AudioTool iz paketa Sphinx-4 in izpostavili razlike in podobnosti. Prav tako smo izvedli primerjavo programa iSound ter nekaterih drugih programov, ki imajo podobne funkcionalnosti. Ugotovili smo, da je iSound popolnoma primerljiv z ostalimi programi.

Ključne besede

obdelava signalov, Fourierjeva transformacija, DFT, FFT, časovno-frekvenčna analiza, spektrogram

Summary

For the real time spectrogram demonstration we used open-source system Sphinx-4, that we researched, determined the meaning and exposed the restrictions. We described Discrete Fourier transformation (DFT) and Fourier transformation with proceeding of Fast Fourier transformation (FFT). During the analysis we compared our program, which we named iSound and program AudioTool from Sphinx-4 package to expose the differences and resemblances. In the similar way, we compared iSound with some other programs that had similar functioning. We realised that the iSound is completely comparable with other programs.

Keywords

signal processing, Fourier transformation, DFT, FFT, time-frequency analysis, spectrogram

Kazalo

Seznam slik

Algoritmi

1 Uvod

Navadno se časovno-frekvenčna analiza zvočnih signalov izvaja naknadno, po zajetem signalu. V naši zaključni nalogi pa smo se osredotočili na sprotno pretvorbo signalov v časovno-frekvenčni prostor, kar nam bi omogočilo sprotno analizo zvočnih signalov.

1.1 Opredelitev področja in opis problema

Področje raziskovalnega dela zaključne naloge zajema računalništvo in programiranje v programskem jeziku *Java*, poznavanje lastnosti analogne in digitalne obdelave zvoka, *Fourierjeve transformacije* in izvedbe hitrih algoritmov za izračun Fourierjeve transformacije, poznavanje odprtokodnega sistema Sphinx-4, izdelave uporabniškega vmesnika in dela z grafiko, poznavanje postopkov za analize, meritve in ocenjevanje delovanja programov.
Običajno se analiza zvočnih signalov dela naknadno, kar pomeni, da zvok najprej zajamemo, šele po končanem zajemu ga analiziramo. Naša naloga je bila izdelati program, ki bi omogočal sprotno opazovanje dinamičnih sprememb signalov v časovno-frekvenčnem prostoru.
Pri tem je bilo potrebno spoznati teoretične osnove časovno-frekvenčne analize, spoznati postopke izvedbe spektrograma in izvesti implementacije postopka za sprotno prikazovanje spektrograma z odprtokodnim orodjem Sphinx-4, ki se uporablja pri gradnji sistemov za razpoznavanje govora.

1.2 Namen, cilji in hipoteza naloge

Namen zaključne naloge je spoznati postopke časovno-frekvenčne analize, ki so zajeti v sistemu Sphinx-4, in implementirati nov postopek časovno-frekvenčne analize, ki bo omogočala sprotno prikazovanje signalov.
V ta namen je potrebno izdelati program, v katerem bi obstoječe postopke iz sistema Sphinx-4 nadgradili za sprotno prikazovanje rezultatov časovno-frekvenčne analize.

Hipoteza zaključne naloge je, da je mogoče v sistemu Sphinx-4 obstoječe postopke za izvedbo analize po zajemu zvoka nadgraditi tako, da izvajamo analizo zvoka sprotno.

1.3 Predvidene metode raziskovanja

Pri raziskovalnem delu zaključne naloge smo se osredotočili na implementacije spektrograma v odprtokodnem sistemu Sphinx-4 s programskim jezikom Java. Pri tem je bilo potrebno najprej izvesti prototipni postopek izvedbe spektrograma v programskem okolju Matlab, kjer smo preučili osnovne gradnike in lastnosti spektrograma.
Zaradi zahteve po sprotni analizi signalov s spektrogramom je bilo potrebno obstoječe postopke prilagoditi za sprotni izračun in izris na zaslon. Pri tem smo preučevali postopke za delo z računalniško grafiko v programskem jeziku Java. Bistven poudarek pa je bil tudi na izvedbi programa, ki bi bil združljiv z ogrodjem sistema Sphinx-4.

1.4 Pregled vsebine zaključne naloge

V zaključni nalogi obravnavamo računalniško izvedbo Fourierjeve transformacije, zato je v drugem poglavju predstavljen najprej kratek uvod v predstavitev digitalnega signala. Opisali smo, kako se izvede Fourierjeva transformacija (FT) ter izvedba FT s postopkom hitre FT (FFT).
Tretje poglavje teoretično opisuje sistem za sprotno časovno-frekvenčno analizo zvočnih signalov. Pri časovno-frekvenčni analizi smo se omejili na predstavitev s spektrogramom. Zato smo v tem poglavju določili pomen spektrograma, ga definirali ter izpostavili omejitve. Poudarili smo pomen oknjenja tako, da smo izrisali tri spektrograme iz istega izvora z različnimi dolžinami oknjenja.
Četrto poglavje je namenjeno predstavitvi odprtokodnega sistema Sphinx-4, ki se uporablja za izgradnjo razpoznavalnikov govora. Predstavili smo delovanje in vlogo ogrodja ter vhodnega vmesnika in predstavili, kako se upravlja s konfiguracijo.
V petem poglavju je opisan postopek za namestitev odprtokodnega sistema Sphinx-4, konkreten primer konfiguracijske datoteke, opisane so knjižnice, ki so bile uporabljene pri izdelavi našega programa, koraki za izris spektrograma ter implementacija iSound barvne sheme.
V šestem poglavju je predstavljen uporabniški vmesnik, njegove prednosti, značilnosti, funkcionalnosti ter analiza delovanja programa. Izvedli smo primerjavo programa iSound s programom AudioTool iz paketa Sphinx-4 ter primerjavo programa iSound z nekaterimi drugimi programi, ki se uporabljajo za sprotni izris spektrograma. Na podlagi meritev in primerjav smo ugotovili primerljivost našega programa z ostalimi in predlagali, kaj je potrebno izboljšati.
Na podlagi dobljenih rezultatov smo v zadnjem poglavju predstavili ugotovitve in preverili njihovo proporcionalnost s postavljeno hipotezo.

2 Računalniška izvedba Fourierjeve transformacije

Zvočna kartica vsakega računalnika pretvarja signal iz analognega v digitalnega in obratno. V našem primeru se bodo vzorci analognega signala z mikrofona pretvorili v digitalne, kot je opisano v podpoglavju 2.1.

2.1 Diskretni signali

Diskretnih signalov ne bomo obravnavali neposredno, temveč jih bomo z vzorčenjem opazovali le ob določenih trenutkih, ki so med seboj razmaknjeni za t_0. Število odčitkov signala na sekundo, imenujemo frekvenca vzorčenja (F) in jo merimo v Hz, ki je $\frac{1}{t_0}$ [14].

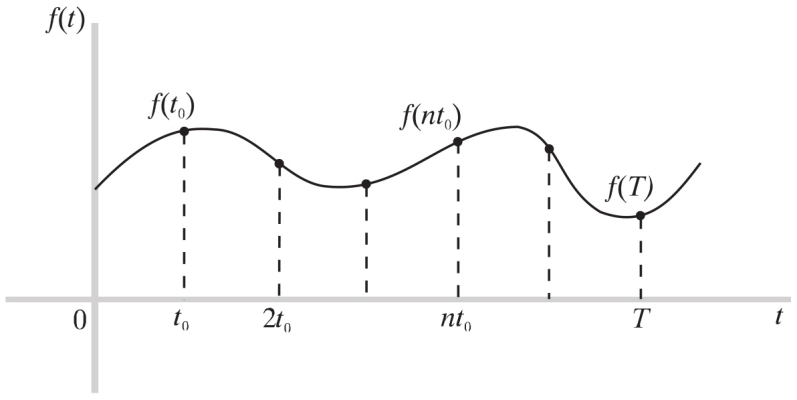
2.1.1 Vzorčenje

Zvezen in frekvenčno omejen signal $f(t)$, opazovan v času med 0 in T, je popolnoma določen, če poznamo njegove vzorce, ki si sledijo v enakomernih časovnih presledkih t_0 (glej sliko 1):

$$t_0 = \frac{1}{2F} \ ,$$

kjer je F najvišja frekvenca, ki jo signal vsebuje.

Vzorčenje s *Shannonovo frekvenco* $\frac{1}{t_0}$ je potreben in zadosten pogoj, da signal $f(t)$ v celoti določimo [14].
Višja kot je frekvenca vzorčenja, bolj natančen je opis signala. Kot je predstavljeno v tabeli *Nastavitve parametrov za AudioTool in iSound* v poglavju 6.2.1, smo pri programu iSound uporabili frekvenco vzorčenja *16 kHz*, ki se običajno uporablja pri izdelavi razpoznavalnikov govora. Frekvenčni spekter signala označimo z Ω, potem je zgornja meja spektra Ω_{max}, ki jo opisuje signal po *Nyquistovem izreku* enaka $\frac{F}{2}$ [29]. Če vzamemo za primer, da je frekvenca vzorčenja *16 kHz*, bo $\Omega_{max} = 8kHz$.
Vsak vzorec je opisan z biti, ki določajo kvaliteto opisa, v našem primeru je *16 bitov*.



Slika 1: Vzorčenje frekvenčno omejenega signala $f(t)$. Slika povzeta po [14].

2.2 Diskretna Fourierjeva transformacija (DFT)

Pri vzorčenju zveznega signala $f(t)$ dobimo zaporedje vzorcev

$$\{f(nT)\} = \{f(0), f(T), f(2T), ..., f([N-1]T)\} \ ,$$

ki so v splošnem lahko kompleksni.
Diskretno Fourierjevo transformacijo $F_D(k\Omega) = \mathcal{F}\{f(nT)\}$ definiramo kot zaporedje vzorcev $\{F_D(k\Omega)\}$ v frekvenčnem prostoru

$$F_D(k\Omega) = \sum_{n=0}^{N-1} f(nT)e^{-jn\Omega Tk}, \tag{1}$$

$$k = 0, 1, 2, ..., N-1 \ ,$$

kjer je N število vzorcev, $\Omega = \frac{2\pi}{NT}$ razmik med vzorci v frekvenčnem prostoru [14].

Diskretna Fourijerjeva transformacija je periodična s periodo $N\Omega$.
Iz osnovne formule (1) lahko z izpeljavami izvedemo zvezo v frekvenčnem prostoru [14], kot opisuje (2)

$$F(\omega)|_{\omega=k\Omega} \doteq T F_D(k\Omega), \tag{2}$$

$$\Omega = \frac{2\pi}{NT}, \tag{3}$$

kjer ω leži v intervalu

$$\tfrac{-N}{2}\Omega < \omega < \tfrac{N}{2}\Omega \ .$$

Izven tega intervala se Fourierjeva transformacija periodično ponavlja. Tako aproksimiramo zvezno Fourierjevo transformacijo z diskretno Fourierjevo transformacijo z določenim pogreškom ϵ, ki je tem manjši, čim manjši je razmik vzorčenja T [14].

2.3 Amplitudni in fazni spekter

Ponazoritev kompleksnega spektra

$F(\omega) = P(\omega) + jQ(\omega)$,
$P(\omega) = ReF(\omega), \quad Q(\omega) = ImF(\omega)$,

$F(\omega) = |F(\omega)|\, e^{j\theta(\omega)}$, kjer je $|F(\omega)|$ amplitudni spekter, $\theta(\omega)$ pa fazni spekter [14].

Pri izrisovanju spektrograma zanemarimo fazni spekter, ker človeško uho razlik v fazi ne zazna. Spremembe, ki jih v zvoku zaznamo so amplitudne, zato upoštevamo pri izrisu spektrograma samo amplitudni spekter $|F(\omega)|$ [9], ki je definiran v enačbi (4).

$$|F(\omega)| = \sqrt{P(\omega)^2 + Q(\omega)^2}. \tag{4}$$

2.4 Izvedba FT s postopkom hitre FT (FFT)

Hitra Fouriereva transformacija (FFT) je učinkovit algoritem za izračun DFT. Obstaja veliko različnih algoritmov FFT, v našem primeru smo uporabili algoritem Cooley-Tukey, ki temelji na metodi "deli in vladaj" [17] in je natančneje opisana v podpoglavju (2.4.1). DFT razgradi zaporedje vrednosti na komponente različnih frekvenc [28]. Ta operacija je uporabna na mnogih področjih, vendar se izkaže, da je njena uporabnost neposredno iz definicije pogosto zelo nizka in se praktično ne uporablja zaradi počasnosti.
FFT je način, s pomočjo katerega do enakega rezultata pridemo hitreje. Za izvedbo DFT algoritma potrebujemo $O(N^2)$ aritmetičnih operacij, medtem ko FFT lahko izračuna enak rezultat v $O(N \log N)$ operacij, kjer N predstavlja število otipkov. FFT algoritmi imajo velik pomen pri digitalnem procesiranju signalov in reševanju parcialnih diferencialnih enačb, pri algoritmih za hitro množenje velikih celih števil. Najbolj znani FFT algoritmi so odvisni od razcepa števila N, vendar obstajajo tudi FFT z $O(N \log N)$ kompleksnostjo za vse N.

2.4.1 Metoda "deli in vladaj"

V tem poglavju smo obdelali metodo, ki je osnova za vse hitre algoritme DFT, ki uporabljajo pristop "deli in vladaj" [17]. DFT je v bistvu matrično-vektorski produkt.
Naj bo $(x_0, x_1, ..., x_{N-1})^T$ vektor vhodnih vzorcev

$$(X_0, X_1, ..., X_{N-1})^T$$

vektor pretvorjenih vrednosti po FT in W_N, N-ti koren enote $W_N = e^{-j2\pi/N}$.
DFT lahko tako zapišemo kot:

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & W_N^3 & \cdots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & W_N^6 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix}. \quad (5)$$

Neposredna ocena matrično-vektorskega produkta potrebuje N^2 kompleksnih množenj in primerjav (zaradi enostavnejše razlage si predstavljajmo, da so vsi signali kompleksni). Osnovna ideja metode "deli in vladaj" je delitev osnovnega problema (O) na več manjših (o), tako, da velja (6) [17]:

$$\sum cena(o) + cena(delitev) < cena(O). \quad (6)$$

Prava prednost metode "deli in vladaj" je v tem, da lahko delitev na manjše probleme pogosto izvajamo rekurzivno, s čimer zmanjšamo kompleksnost, zato na njej temeljijo vsi hitri algoritmi [17]. Problem razdelimo tako, da preučimo dele prvotnega zaporedja, nad njimi izvedemo DFT in nato iz posameznih delov rekonstruiramo DFT prvotnega zaporedja. V podpoglavju 2.4.2 smo opisali primer izračuna FT s postopkom FFT, ki uporablja omenjeno metodo.

2.4.2 Primer izračuna FT s postopkom hitre FT (FFT)

Poskušali smo na najbolj enostaven način opisati izračun FT s postopkom hitre FT, zato smo si izbrali primer, v katerem smo računali FFT za štiri otipke ($N = 4$). Izhodiščni vzorec signala smo označili z x_0, zaradi krajšega pisanja smo pri zapisu jedra izpuščali indeks N. Tako smo dobili [33]:

$$X(n) = \sum_{k=0}^{N-1} x_0(k) W^{nk}, n \in (0, N-1). \tag{7}$$

Enačba (7) je v našem primeru pravzaprav sestavljena iz štirih enačb ($N = 4$), za vsak otipek spektra ena [33]:

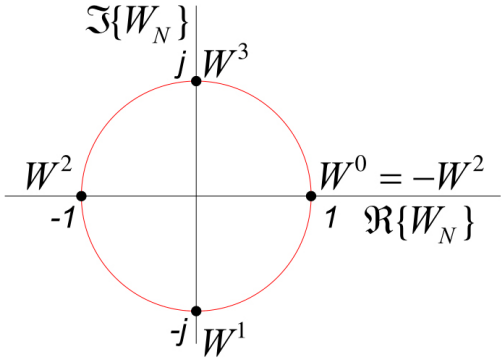
$$\begin{aligned}
X(0) &= x_0(0)W^0 + x_0(1)W^0 + x_0(2)W^0 + x_0(3)W^0, \\
X(1) &= x_0(0)W^0 + x_0(1)W^1 + x_0(2)W^2 + x_0(3)W^3, \\
X(2) &= x_0(0)W^0 + x_0(1)W^2 + x_0(2)W^4 + x_0(3)W^6, \\
X(3) &= x_0(0)W^0 + x_0(1)W^3 + x_0(2)W^6 + x_0(3)W^9,
\end{aligned}$$

oziroma v matrični obliki

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \cdot \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix}. \tag{8}$$

Prvi korak pri razvoju algoritma FFT za (8) je, da delno izračunamo vrednosti koeficientov W^{nk} [33]:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \cdot \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix}. \tag{9}$$



Slika 2: *Cikličnost.* Lega W^0 v kompleksni ravnini. Slika povzeta po [33].

Vrednosti koeficientov smo dobili z upoštevanjem cikličnosti W^{nk}, ki je predstavljena na sliki 2, v analitični obliki jo zapišemo z [33]:

$$W^{nk} = W^{n\,k\,mod\,N},$$

kjer je $n\ k\,mod\,N$ = ostanek deljenja $\frac{nk}{N}$, torej pri $N = 4$, $n = 2$ in $k = 3$ dobimo $W^{2\cdot 3} = W^6 = W^2$.

Drugi korak je bil iskanje faktorjev v matriki (9) tako, da lahko matriko zapišemo v obliki

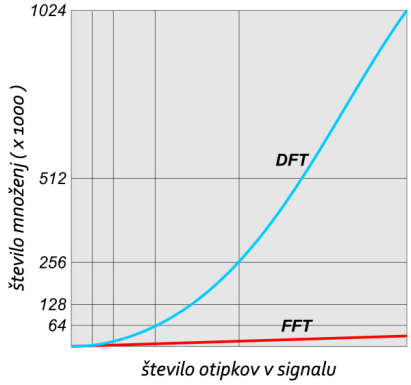
$$\bar{X}_n = \begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \cdot \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix}. \tag{10}$$

Pri enačbi (10) smo opazili, da sta druga in tretja vrstica zamenjani [33].

Naslednji korak je izračun produkta desne matrike in vektorja $x_0(n)$ [33]:

$$\bar{X}_n = \begin{bmatrix} x_1(0) \\ x_1(1) \\ x_1(2) \\ x_1(3) \end{bmatrix} = \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \cdot \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix}. \tag{11}$$

Opazili smo, da za izračun $x_1(0)$ potrebujemo eno kompleksno množenje (z W^0) in eno kompleksno seštevanje: $x_1(0) = x_0(0) + W^0 x_0(2)$. Zaradi splošnosti postopka, W^0 nismo zamenjali z 1 [33].
Element $x_1(1)$ smo prav tako določili z enim kompleksnim množenjem in seštevanjem. Pri računanju $x_1(2)$ pa smo potrebovali le eno kompleksno seštevanje, saj velja $W^0 = -W^2$: $x_1(2) = x_0(0) + W^2 x_0(2)$ in $x_1(2) = x_0(0) + W^0 x_0(2)$, kjer smo množenje $W^0 x_0(2)$ že izvedli pri računanju $x_1(0)$. Iz istega razloga smo imeli pri računanju $x_1(3)$ eno kompleksno seštevanje. Vmesni vektor $x_1(k)$ smo tako izračunali s štirimi kompleksnimi seštevanji in dvema kompleksnima množenjima [33].
Postopek smo nadaljevali, dokler nismo dobili vseh vmesnih vektorjev. Množenja, ki smo jih pri prejšnjih korakih že izvedli smo upoštevali v nadaljevanju in tako zmanjšali število množenj [33].
Računanje $\bar{X}_n$ po enačbi (10) je zahtevalo 4 kompleksna množenja in 8 kompleksnih seštevanj. Če bi računali X_n s pomočjo običajne DFT, bi potrebovali 16 kompleksnih množenj in 12 kompleksnih seštevanj. Sliki 3a in 3b prikazujeta izračune kompleksnega množenja in seštevanja za DFT in FFT za vrednosti $N = 4$ ($\gamma = 2$) [33].



(a) Graf za primerjavo števila množenj pri *DFT* in *FFT* v odvisnosti od N.

FFT	$\frac{N\gamma}{2} \quad \left(= \frac{4 \cdot 2}{2} = 4\right)$	kompleksnih množenj
	$N\gamma \,(= 4 \cdot 2 = 8)$	kompleksnih seštevanj
DFT	$N^2 \quad (4^2 = 16)$	kompleksnih množenj
	$N(N-1) \,(= 4 \cdot 3 = 12)$	kompleksnih seštevanj

(b) Tabela za primerjavo števila kompleksnih množenj in seštevanj pri DFT in FFT za konkretne vrednosti N in γ.

Slika 3: Primerjava števila množenj pri *DFT* in *FFT*. Sliki povzeti po [33].

Slabost procesa faktorizacije je, da rezultat ni bil več urejen po istem vrstnem redu, kot vhodni podatki, kar je postranski problem FFT. Spomnimo se, da je rezultat FFT $\bar{X}_n$ in ne X_n [33], torej:

$$\bar{X}_n = \begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} \neq \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = X_n$$

Težavo smo rešili tako, da smo številke otipkov zapisali kot binarne vrednosti v nasprotni smeri ter jih s to preprosto metodo uredili, kot prikazuje enačba (12) [33]. Če zapišemo binarno vrednost tako, da obrnemo vrstni red se 10_B pretvori v 01_B, 01_B se pretvori v 10_B, 00_B in 11_B se ne spremenita. Tak način ureditve podatkov se v računalništvu uporablja pogosto, zaradi preproste realizacije. V pomnilniškem registru v računalniku se podatki že hranijo v binarnem zapisu, torej, potrebno je le v desno zamakniti vrednosti.

$$\bar{X}_n = \begin{bmatrix} X(00_B) \\ X(10_B) \\ X(01_B) \\ X(11_B) \end{bmatrix} \Rightarrow \begin{bmatrix} X(00_B) \\ X(01_B) \\ X(10_B) \\ X(11_B) \end{bmatrix} = X_n \tag{12}$$

3 Sistem za sprotno frekvenčno analizo zvočnih signalov

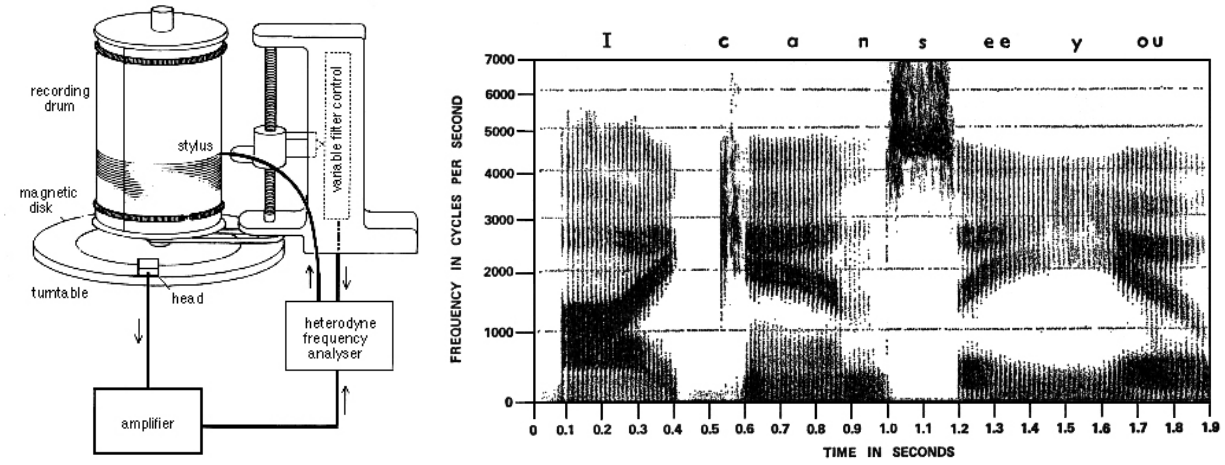
3.1 Definicija spektrograma

Spektrogram nam prikazuje, kako se kratkočasovni amplitudni spekter signala spreminja v odvisnosti od časa. Na področju časovno-frekvenčne obdelave signala je ena izmed najbolj priljubljenih metod za predstavitev in vizualizacijo signala v skupnem časovno-frekvenčnem prostoru. Sopomenki za izraz spektrogram sta tudi spektralni slap in sonogram. Spektrogram se uporablja na različnih raziskovalnih področjih: prepoznavanje fonetičnih zvokov pri analizi živalske zvočne komunikacije [1], instrumentalna in vokalna glasbena produkcija [24], sonar/radar [16], razpoznavanje govora [8], zdravstvo [21] [11], genetika [10], seizmologija [31] itd.

3.2 Spektralna analiza

Spektrograme lahko običajno ustvarimo na dva načina: usklajeno, s pomočjo serije frekvenčnih pasovnih filtrov (to je bil edini način pred prihodom moderne digitalne obdelave signala), ali pa s pomočjo obdelave zvočnega signala v od časa z uporabo FFT [32].

Analogna metoda s pasovnimi filtri uporablja analogne metode obdelave pri razdelitvi signala v frekvenčne pasove; velikosti na izhodih filtrih nadzorujejo posamezni senzorji, ki beležijo spektrogram kot sliko na papirju, kar prikazujeta sliki 4a in *4b* [20].



(a) Naprava za izris spektrograma na papir. Slika povzeta po [20].

(b) Spektrogram na papirju. Slika povzeta po [20].

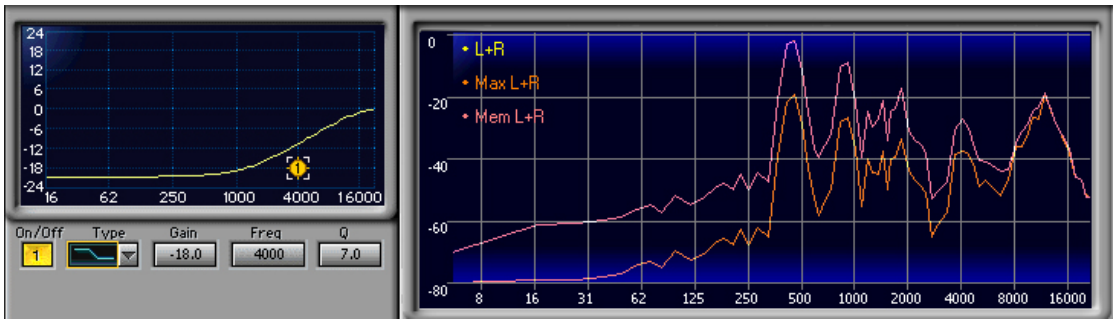
Slika 4: Izris spektrograma s pomočjo analognih metod.

3.2.1 Digitalna metoda za spektralno analizo

Digitalna metoda za ustvarjanje spektrograma z uporabo FFT se običajno izvaja nad posameznimi časovnimi kosi vhodnega signala, ki se jih sproti ali pa kasneje smiselno združuje in predstavi kot celoto. Proces je sledeč:

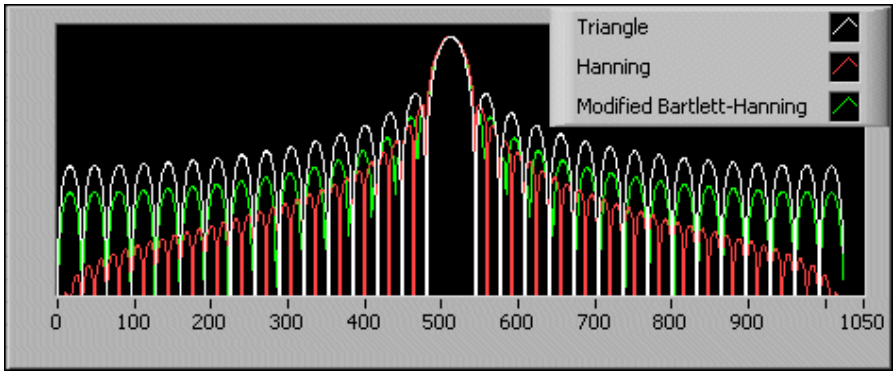
1. Nad vhodnim signalom izvajamo digitalno vzorčenje podatkov v odvisnosti od časa.

2. Pri posameznemu časovnemu kosu je potrebno izpostaviti frekvence, ki so za nas najbolj pomembne. Slika 5 prikazuje poslabitev frekvenc pod 4kHz za -18dB, s tem izpostavimo tiste, ki so višje od 4kHz. Podrobnejši opis, kako to naredimo, je v poglavju 5.1.3. Ta korak je opcijski, kar pomeni, da ga lahko tudi izpustimo.



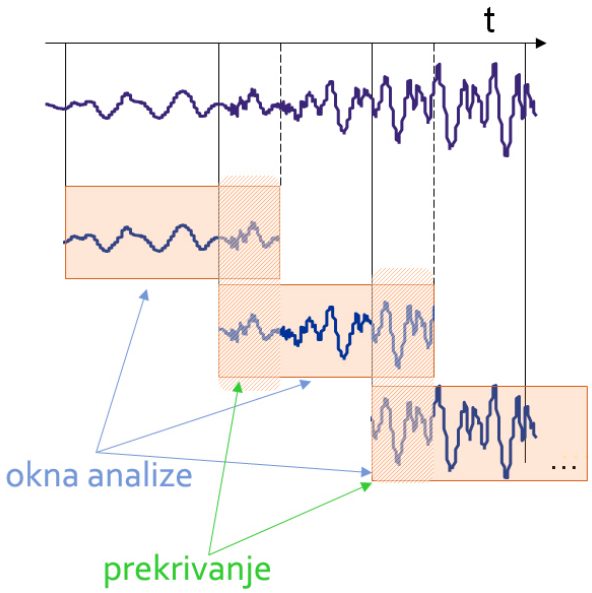
Slika 5: Poslabitev nizkih frekvenc v programu WavLab 5 s pomočjo vtičnika Waves Q1 (levo) in prikaz frekvenčnega spektra pred obdelavo (viola črta) in po obdelavi (oranžna črta), s pomočjo vtičnika Waves PAZ Frequency (desno).

3. S pomočjo oknjenja časovni kos razrežemo na okna, ki se delno prekrivajo, kot je prikazano na sliki 7. Uporabimo lahko različna okna, glede na tipe signalov, ki jih obdelujemo. Slika 6 prikazuje primerjavo različnih vrst oknjenj, ki se pogosto uporabljajo, iz katere so razvidne razlike in njihov vpliv na signal. Pri analizi govornih signalov se najpogosteje uporablja Hammingovo okno, ki je prikazano na sliki 18. Postopek Hammingovega oknjenja je natančneje opisan v poglavju 5.1.4.

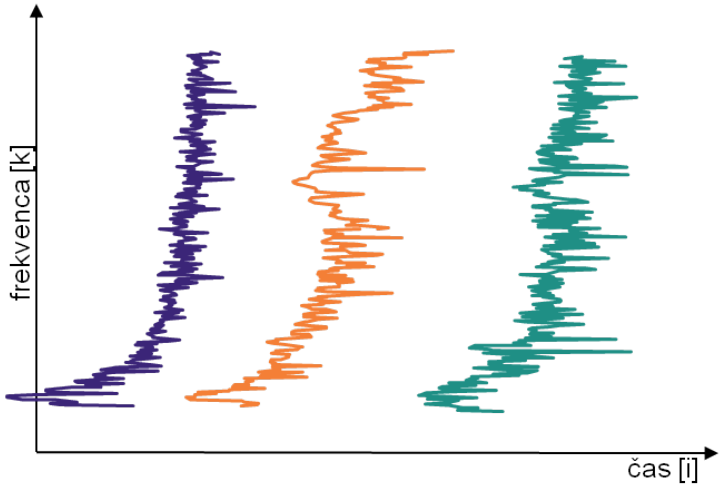


Slika 6: Razlika med trikotnim, Hanningovim in Bartlett-Hanningovim oknjenjem. Slika povzeta po [6].

4. Po oknjenju izvedemo nad časovnim kosom DFT s FFT. Tako dobimo FT oknjenega signala za vsak kos. Če izračun ponovimo za vsak del signala, kot je to prikazano na sliki 7, dobimo porazdelitev energije signala po frekvenci (amplitudni spekter) za vsak časovni odsek signala, kot je prikazano na sliki 8. Skupno predstavitev časovnih odsekov signala imenujemo spektrogram. Pri tem je potrebno določiti širino okna in definirati preskoke v času med sosednjima oknoma. Postopka DFT in FFT sta natančneje opisana v poglavju 2.



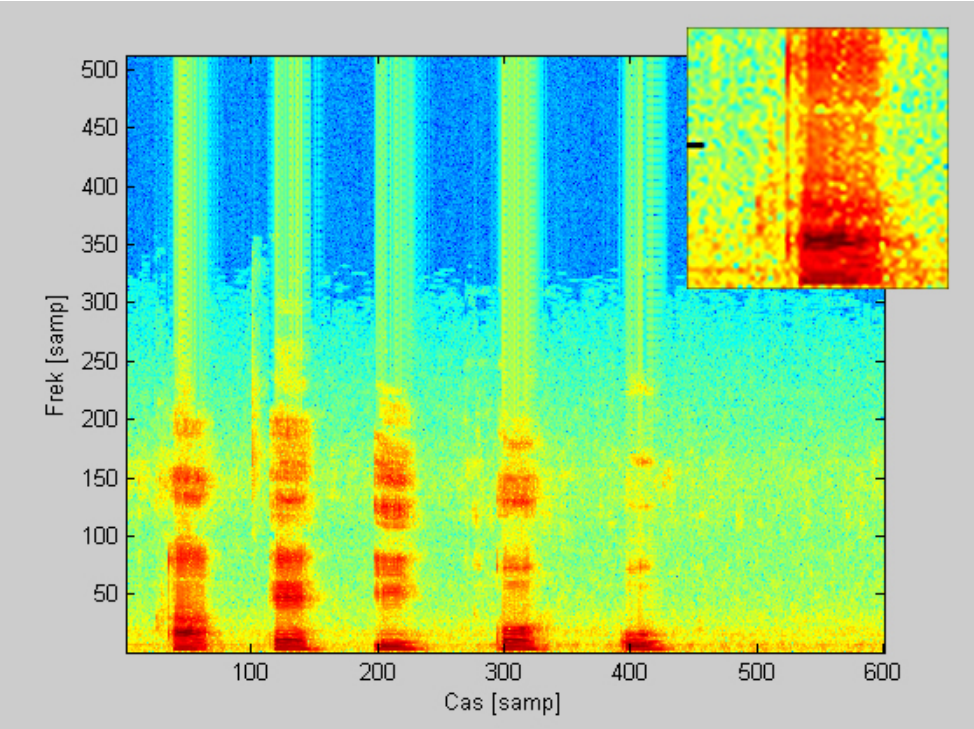
Slika 7: Vzorčenje zvočnega signala in prekrivanje posameznih oken. Slika povzeta po [9].



Slika 8: Amplitudni spekter za vsak časovni kos posebej – spektrogram. Slika povzeta po [9].

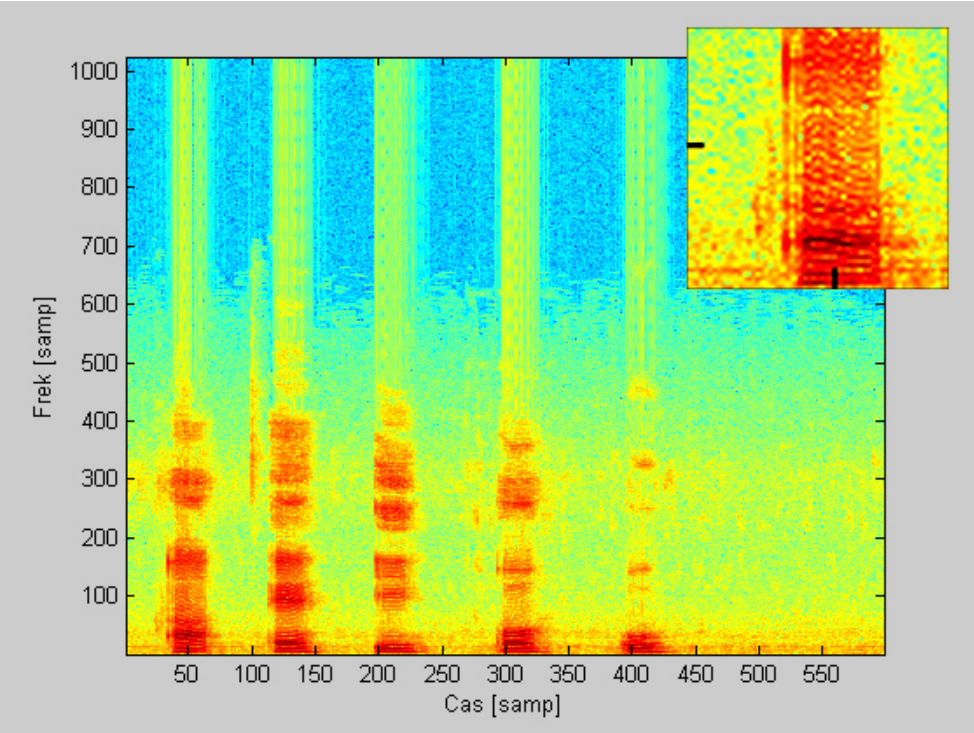
3.2.2 Izvedba spektrograma v Matlabu

Za lažje razumevanje pomembnosti oknjenja pri spektralni analizi smo v programu *Matlab*[1]*, ver. R2011b* izdelali in testirali funkcijo, ki s pomočjo vhodnih parametrov izriše spektrogram. Vhodni parametri so zvočni posnetek (*.wav datoteka*), dolžina okna (ms) in čas pri preskoku okvirja (ms). Za testiranje smo si izbrali tri primere, v katerih sta zvočni posnetek in čas pri preskoku okvirja enaka, spreminjali pa smo dolžino okna z namenom ugotavljanja sprememb pri izrisu spektrograma. Za zvočni posnetek smo si izbrali mono posnetek (v standardni CD kvaliteti) moške izgovorjave samoglasnikov (a, e, i, o, u), čas pri preskoku okvirja smo določili $10ms$, dolžino okna pa smo spreminjali na vrednosti $20ms$, $32ms$ ter $50ms$.
Če si ogledamo slike od 9 do 11, ugotovimo, da se povečuje frekvenčni razpon (os y) in izris spektrograma se spreminja. Opazimo lahko, da z večanjem dolžine okna pridobimo večjo frekvenčno ločljivost ter manjšo časovno ločljivost, kar rezultira v izris spektrograma z razmazano osjo x. Z manjšanjem dolžine okna se veča časovna ločljivost ter manjša frekvenčna ločljivost, kar rezultira v izris spektrograma z razmazano osjo y. Na vsaki sliki posebej je v desnem kotu zgoraj povečan in izostren enak del slike, iz katerega je jasno razvidno, kako se s spreminjanjem dolžine okna spreminja izris spektrograma.

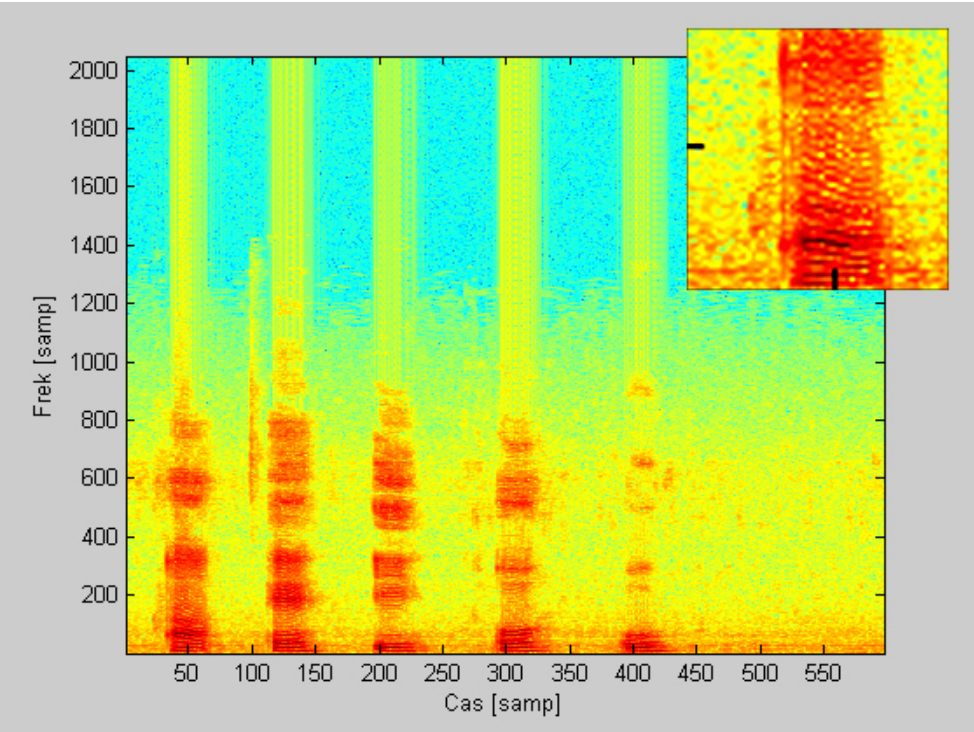


Slika 9: Izris spektrograma pri dolžini okna $20ms$.

[1]Matlab je programsko orodje za numerično računanje, ki ga je razvilo podjetje *MathWorks*.



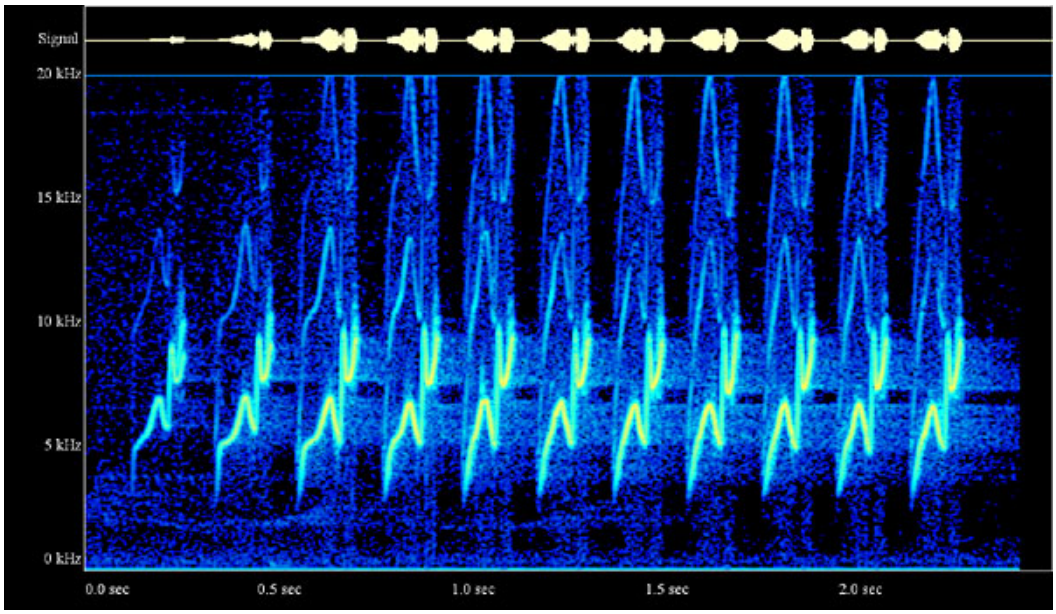
Slika 10: Izris spektrograma pri dolžini okna $32ms$.



Slika 11: Izris spektrograma pri dolžini okna $50ms$.

3.3 Prikazovanje spektrograma

Najpogostejši format za prikaz spektrograma je graf z dvema geometrijskima dimenzijama: horizontalna os predstavlja čas, vertikalna os frekvenco, tretji podatek, ki ga lahko razberemo s spektrograma je intenzivnost ali moč signala za določeno frekvenco, ki je predstavljena z barvo (temnejša/svetlejša kot je, močnejši zvok se nahaja na določeni frekvenci ob določenem času).
Obstaja veliko različic prikaza: včasih sta vertikalni in horizontalni osi zamenjani, tako da čas teče vertikalno, včasih je amplituda namesto z barvo predstavljena kot tretja geometrijska dimenzija (z), tako dobimo izris 3D slike signala. Frekvenco in amplitudo lahko predstavimo linearno ali logaritmično, odvisno, čemu je graf namenjen. Običajno je zvok predstavljen z logaritmično amplitudno osjo (dB), frekvenca pa linearno, da poudari odnose harmonij ali logaritmično, da poudari razmerje med posameznimi toni.



Slika 12: 2D spektrogram. Slika povzeta po [27].

3.4 Omejitve

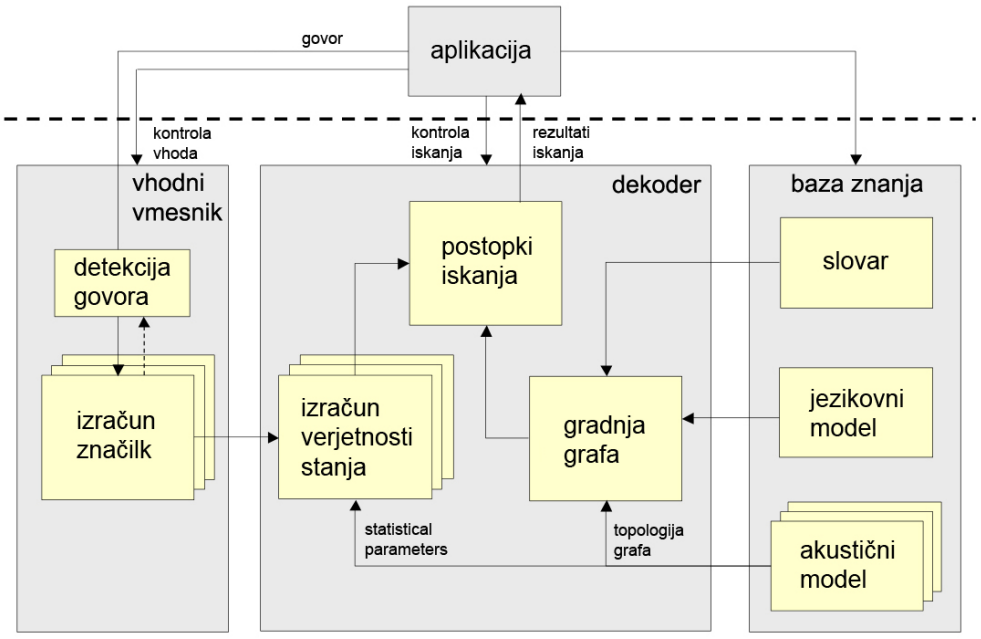
Spektrogram ne vsebuje informacije o fazi signala, vsebuje le informacije o amplitudi signala, kar smo poudarili in pokazali že v poglavju 2.3. To je razlog, zaradi katerega ni mogoče ustvariti izvorni signal iz spektrograma, čeprav je v primerih, ko začetna faza ni pomembna, mogoče generirati uporaben približek izvornega signala [2].

4 Predstavitev programskega paketa Sphinx-4

Sphinx-4 je fleksibilen, modularen in razširljiv odprtokodni sistem za razvoj sistemov za razpoznavanje govora.
Podobna orodja za razvoj razpoznavalnikov govora so še: HTK [12], ISIP [13], AVCSR [22] in ostale starejše različice Sphinx sistema. Večinoma so ti sistemi strogo usmerjeni in optimizirani za delo na specifičnih področjih. Rezultat tega so omejitve in ovire pri razvoju sistema na sorodna področja, katerih uporabnost se razlikuje od prvotnega namena sistema [19].
Iz tega razloga in mnogih drugih so oblikovali sposobno razvojno ekipo, da ustvari Sphinx-4, nadgradljiv, vsem dostopen in brezplačen sistem, napisan v programskem jeziku Java.

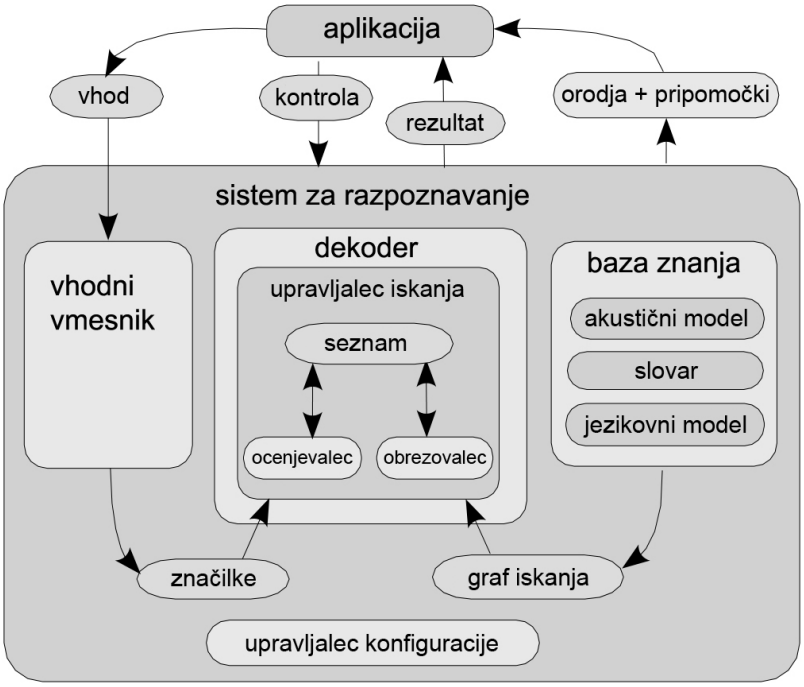
4.1 Opis sistema Sphinx-4

Ogrodje (*Framework*) Sphinx-4 je bilo zasnovano za visoko stopnjo prilagodljivosti in modularnosti. Splošno arhitekturo sistema prikazuje slika 13 – za lažje razumevanje je na sliki 14 Sphinx-4 ogrodje ponovno predstavljeno, vendar nekoliko drugače. Vsak označeni element na sliki prikazuje modul, ki ga je mogoče enostavno nadomestiti in eksperimentirati z različnimi izvedbami, ne da bi morali spremeniti druge dele sistema. Obstajajo trije osnovni moduli v sklopu Sphinx-4 ogrodja: vhodni vmesnik (*Frontend*), dekoder (*Decoder*) in baza znanja (*Knowledge base*) [18] [19].



Slika 13: *Sphinx-4 ogrodje*. Ogrodje je sestavljeno iz treh glavnih sklopov: vhodnega vmesnika, dekoderja in baze znanja. Pomembnejši del Sphinx-4 ogrodja je tudi upravljalec konfiguracije (*Configuration manager*). Slika povzeta po [18].

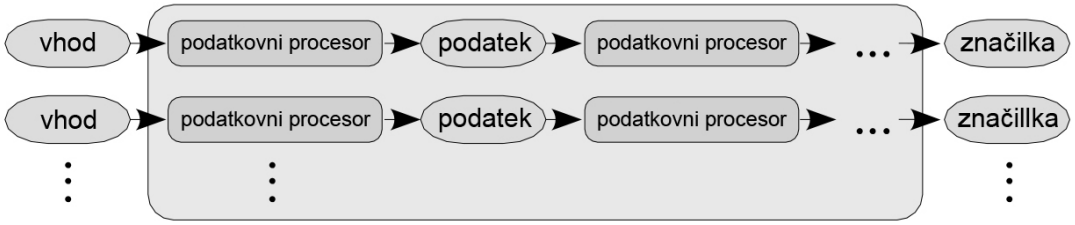
Vhodni vmesnik poskrbi za to, da prevzame enega ali več vhodnih signalov ter jih parametrizira v zaporedje značilk (*Features*), ki so potrebne za razpoznavanje govora. Baza znanja ima nalogo, da poveže akustične modele (*AcousticModel*) z osnovnimi enotami govora in zgradi graf za razpoznavanje govora. Graf se zgradi na podlagi besed, ki so v slovarju (*Dictionary*) prevedene v zaporedje govornih enot. Povezave med besedami na grafu pa so še dodatno utežene z verjetnostmi zaporedja posameznih besednih zvez, ki jih ocenimo z jezikovnimi modeli (*LanguageModel*) [19].
Upravljalec iskanja (*SearchManager*) uporablja lastnosti iz vhodnega vmesnika in iskalni graf za izvajanje dekodiranja in pridobitev rezultatov. Med izvajanjem procesov razpoznavanja lahko kadarkoli sprožimo zahtevek za vključevanje ali izključevanje modulov iz tega procesa [19].
Slika 14 prikazuje komunikacijo med posameznimi bloki in aplikacijo. Aplikacija poda vhodnemu vmesniku željeni vhod, tako dobimo na izhodu vmesnika značilke, ki jih lahko s pomočjo dekoderja in baze znanja uporabimo za razpoznavanje govora. Za izgradnjo sistema za razpoznavanje je potrebno pravilno nastaviti konfiguracijsko datoteko, s katero manipulira upravljalec konfiguracije (*ConfigurationManager*) [19].



Slika 14: *Sphinx-4 ogrodje dekoderja.* Vhodni vmesnik, dekoder in baza znanja so glavni moduli. Povezava med bloki in aplikacijo je prikazana z usmerjenimi črtami. Slika povzeta po [19].

4.2 Vhodni vmesnik

Namen vhodnega vmesnika je parametrizacija vhodnega signala (zvoka) v zaporedje izhodnih značilk. Kot je prikazano na sliki 15, lahko vhodni vmesnik združuje več vzporednih verig, ki jih sestavljajo zamenljivi moduli, imenovani podatkovni procesorji (*DataProcessors*). Podpora več verigam omogoča sočasno procesiranje različnih tipov parametrov, ki imajo lahko isti ali različen vir vhodnega signala. To omogoča ustvarjanje sistemov, ki lahko sočasno dekodirajo s pomočjo različnih parametrov, kot sta MFCC in PLP, celo parametre, katerih izvor ni zvok, ampak video [19].



Slika 15: *Sphinx-4 vhodni vmesnik.* Slika povzeta po [19].

Kot pri ISIP [35] sistemu, vsak podatkovni procesor omogoča iz vhodnega vmesnika povezavo vhodnega in izhodnega signala na druge podatkovne procesorje, kar pomeni, da imamo lahko poljubno dolge verige le-teh. Vhod in izhod vsakega podatkovnega procesorja so generični podatkovni objekti, ki vsebujejo obdelane vhodne podatke kot tudi oznake, ki nam povejo, kako se dogodki med podatki razvrščajo (npr. kje se nahaja končna točka). Zadnji podatkovni procesor v vsaki verigi je odgovoren za izdelavo podatkovnega objekta, sestavljenega iz parametriziranih signalov, ki jih bo uporabljal dekoder [19].
Tako kot AVCSR [22] sistem tudi Sphinx-4 omogoča uporabo vzporednih zaporednih funkcij, le da to lahko izvajamo nad poljubnim številom vzporednih podatkovnih tokov. Komunikacija med posameznimi bloki poteka po principu zasnove "pull". Zasnova "pull" omogoča, da podatkovni procesor zahteva podatke od prejšnjega modula takrat, ko jih potrebuje, kar je nasprotje sistemov z bolj konvencionalno zasnovo "push", kjer modul pošilja izhodne podatke v trenutku, ko so generirani. Zasnova "pull" omogoča podatkovnim procesorjem, da s pomočjo pomnenja uporabnikom zagotovijo možnost vpogleda v podatke v odvisnosti od časa [19].
Sphinx-4 nam s pomočjo generičnega vhodnega vmesnika omogoča uporabo različnih podatkovnih procesorjev, ki nad signalom izvajajo različne tehnike procesiranja. Te implementacije vključujejo podporo za naslednje: branje številnih formatov z različnih virov, branje vhodnega signala s sistemske zvočne naprave za sprotno delo s podatki, filtriranje signalov, oknjenje signalov, Fourierjevo transformacijo (s pomočjo FFT), filtriranje, diskretno kosinusno transformacijo (DCT), "end pointing", izračun značilk govornih signalov, koeficientov govornega kepstra (MFCC), koeficientov linearne predikcije (LPCC) ali koeficientov PLP (RASTA PLP) [19].

4.3 Upravljanje konfiguracije

Arhitektura vhodnega vmesnika pri sistemu Sphinx-4 omogoča fleksibilnost. Vhodni vmesnik mora biti oblikovan s pomočjo cevovodov (pipeline) podatkovnih procesorjev, kjer mora biti celotna sestava cevovodov nastavljiva s konfiguracijsko datoteko. Možno je tudi dodati več instanc primerov enakih podatkovnih procesorjev istemu vhodnem vmesniku, kjer lahko vsakemu izmed njih nastavimo različne parametre. Prav tako je mogoče dodati več različnih vhodnih vmesnikov v isto konfiguracijsko datoteko.

Algoritem 1 Primer konfiguracijske datoteke

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
<component name="ime.FrontEnd" type="Java.razred.FrontEnd">
<propertylist name="pipeline">
<item>ime.1.komponente.v.cevovodu</item>
<item>ime.2.komponente.v.cevovodu</item>
</propertylist>
<component name="ime.1.komponente.iz.cevovoda" type="Java.razred.1.komponente"/>
<property name="nastavljiva.spremenljivka" value="vrednost"/>
</component>
<component name="ime.2.komponente.iz.cevovoda" type="Java.razred.2.komponente"/>
</component>
</config>
```

Konfiguracijska datoteka je v bistvu XML tip datoteke, zato je potrebno najprej na vrhu vključiti vrstico, ki nam pove različico XML-a in privzeto kodiranje (v našem primeru UTF-8). Potrebno je dodati še *<config></config>* področje, znotraj katerega bo sistem Sphinx-4 bral nastavitve. Prva komponenta določa vse podatkovne procesorje, ki so uporabljeni v vhodnem vmesniku. Ime našega vhodnega vmesnika, ki definira cevovod, je *ime.FrontEnd*; spremenljivki type določimo ime njenega Java razreda. Sledi ji niz naštetih podatkovnih procesorjev v seznamu *propertylist*, poimenovane *pipeline* in določajo lastnosti razreda vhodnega vmesnika. Ostale komponente, ki so naštete pod *ime.FrontEnd* komponento določajo lastnosti, Java razred in nastavljive spremenljivke vsakega podatkovnega procesorja posebej. Če si npr. ogledamo algoritem 1 "Primer konfiguracijske datoteke" in poiščemo podatkovni procesor *ime.1.komponente.iz.cevovoda*, lahko razberemo, da ima definirano spremenljivko z imenom *nastavljiva.spremenljivka* na neko začetno vrednost. Če bi si ogledali *Java.razred.1.komponente*, bi ugotovili, da je *nastavljiva.spremenljivka* ena izmed nastavljivih lastnosti tega razreda. Tako kot smo nastavili spremenljivko na neko začetno vrednosti, ki se razlikuje od privzete vrednosti, lahko nastavimo katerokoli izmed ostalih vrednosti. Konkreten primer lastnosti razreda je predstavljen v poglavju 5.1.1, kjer opisujemo lastnosti razreda mikrofona.

5 Izvedba postopka za sprotno prikazovanje spektrograma

5.1 Konfiguracijska datoteka in knjižnice uporabljene pri našem projektu

Algoritem 2 Konfiguracijska datoteka za naš program

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
<component name="microphone" type="edu.cmu.sphinx.frontend.util.Microphone">
<property name="closeBetweenUtterances" value="true"/>
</component>
<component name="frontEnd" type="edu.cmu.sphinx.frontend.FrontEnd">
<propertylist name="pipeline">
<item>streamDataSource</item>
<item>preemphasizer</item>
<item>windower</item>
<item>fft</item>
</propertylist>
</component>
<component name="preemphasizer"type="edu.cmu.sphinx.frontend.filter.Preemphasizer"/>
<component name="windower"type="edu.cmu.sphinx.frontend.window.RaisedCosineWindower"/>
<component name="fft" type="edu.cmu.sphinx.frontend.transform.DiscreteFourierTransform"/>
<component name="streamDataSource" type="edu.cmu.sphinx.frontend.util.StreamDataSource">
<property name="sampleRate" value="16000"/>
</component>
</config>
```

5.1.1 Komponenta edu.cmu.sphinx.frontend.util.Microphone

Mikrofon zajema zvok iz osnovnih sistemskih vhodov ter ga pretvarja v objekte. Ko se zgodi klic metode *startRecording()*, se ustvari nova nit, ki zajema podatke iz vhoda in se ustavi s klicem metode *stopRecording()*. Metoda *getData()* nam vrne zajet zvok kot podatkovne objekte.
Mikrofon bo poskušal pridobiti zvočno napravo s formatom, določenim v konfiguracijski datoteki. Če te naprave z določenim formatom ni mogoče pridobiti, bo poskušal pridobiti napravo, ki ima višjo hitrost vzorčenja, kot je zapisano v konfiguracijski datoteki. Ostali parametri formata, kot so velikost vzorca, vrstni red zlogov, znak in kanal, ostanejo enaki. Če kljub temu še vedno ne najde naprave, ustvari oznako, da je prišlo do napake in metoda *startRecording()* nam vrne vrednost $FALSE$.

Opis parametrov konstruktorja Microphone

parametri konstruktorja	**tip**	**razlaga**
sampleRate	int	vzorčenje podatkov
bitsPerSample	int	število bitov na vzorec
channels	int	število kanalov
bigEndian	boolean	vrstni red zlogov podatkov
signed	boolean	ali so podatki predznačeni
closeBetweenUtterances	boolean	sproščanje avdio naprave, ko ne zaznamo govora v signalu Na nakaterih operacijskih sistemih (Linux) sproščanje in ponovno odpiranje avdio naprave ne deluje najbolje. Privzeta frednost je $FALSE$ za Linux in $TRUE$ za ostale
msecPerRead	int	število v milisekundah, ki določa, koliko naj bo časovno dolga posamezna časovni interval pri zajemanju zvoka
keepLastAudio	boolean	ali želimo obdržati v spominu zadnji del obdelanega signala, dokler se naslednji ne posname
stereoToMono	java.lang.String	pretvorba stereo zvoka v mono Trenutno lahko nastavimo vrednosti na "average," ki združi kanala in naredi povprečje in "selectChannel," ki omogoča izbiro kanala
selectedChannel	int	kanal, ki se bo uporabljal, če je spremenljivka *stereoToMono* nastavljena na "selectChannel"
selectedMixerIndex	java.lang.String	katero enoto mešalke naj se uporabi Nastavitev na vrednost "default" pomeni, da prepustimo sistemu, da se odloči, kateri mikser bo uporabljal. "Last" pomeni, da sistem izbere zadnji podprt kanal ali pa celoštevilska vrednost, ki predstavlja indeks od *Mixer.Info*, ki ga dobimo tako, da kličemo *AudioSystem.getMixerInfo()*.
audioBufferSize	int	velikost predpomnilnika pri branju zvoka

Tabela *Opis parametrov konstruktorja Microphone* prikazuje seznam parametrov konstruktorja *Microphone* ter tip in namen posameznega parametra. Pri stolpcu *razlaga* smo opisali najprej namen konstruktorja, če je bilo potrebno smo v nadaljevanju podrobneje razložili delovanje in možnosti pri izbiri vrednosti.

5.1.2 Komponenta edu.cmu.sphinx.frontend.FrontEnd

Vhodni vmesnik (FrontEnd) je ovojni razred za verigo podatkovnih procesorjev. Vsebuje metode za manipulacijo in navigacijo skozi posamezne podatkovne procese, kar je tudi uprizorjeno na sliki 16.

Slika 16: Inicializacija podatkovnih procesorjev vhodnega vmesnika. Slika povzeta po [4].

Podatki vstopijo in izstopijo iz vhodnega vmesnika ter potujejo med posameznimi podatkovnimi procesorji. Navadno je vhodni podatkovni tip nek zvočni signal (mikrofon, datoteka), vendar mu lahko ponudimo katerikoli veljaven podatkovni tip (npr. slika). Za sprotno izrisovanje spektrograma bo tako v našem primeru vhodni podatkovni tip signal z mikrofona ter izhodni podatkovni tip *Data*, ki vrača podatkovne kose obdelanega zvočnega signala.
FrontEnd uporablja "pull" model za pridobivanje podatkov. Če želimo ustvariti izhod iz vhodnega vmesnika, moramo to storiti z naslednjo kodo:

Algoritem 3 Pridobivanje podatkov iz vhodnega vmesnika

```
FrontEnd frontend = ...;
Data output = frontend.getData();
```

5.1.3 Komponenta edu.cmu.sphinx.frontend.filter.Preemphasizer

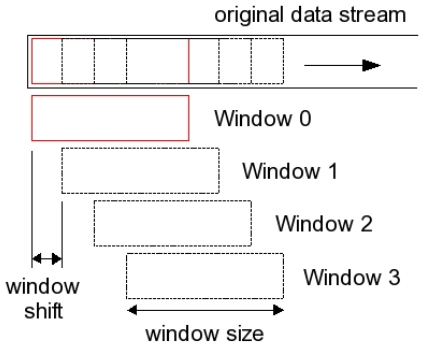
Preemphasizer implementira filter, ki izpostavi visoke frekvence, po navadi tako, da zniža nizke frekvence, kot je prikazano na sliki 5. Govorni signali po navadi oslabijo pri višjih frekvencah za približno 20dB, za kar sta glavna vzroka akustika prostora in karakteristike mikrofona. Komponenta *Preeamphasizer* po navadi sprejme podatkovni objekt, ki predstavlja zvočne podatke, in vrne isti predelan podatkovni objekt. Za vsako vrednost $X[i]$ (kjer i predstavlja čas) iz vhodnega podatkovnega objekta X, uporabimo naslednjo formulo, s pomočjo katere generiramo izhodni podatkovni objekt Y:

$$Y[i] = X[i] - (X[i-1] \cdot FaktorPoudarka). \tag{13}$$

Pri enačbi (13) ima FaktorPoudarka vrednost, definirano z lastnostjo
PROP_PREEMPHASIS_FACTOR, ki je po privzeti nastavitvi nastavljena na 0,97.

5.1.4 Komponenta edu.cmu.sphinx.frontend.window.RaisedCosineWindower

Komponenta *RaisedCosineWindower* izvaja oknjenje signala. Oknjenje izvaja tako, da signal razreže na enake časovne kose, ki se med seboj prekrivajo, kot je to prikazano na sliki 17. Dolžino okna in stopnjo prekrivanja je potrebno določiti, kot smo to storili v poglavju 3.2.2, pri izvedbi spektrograma v Matlabu.

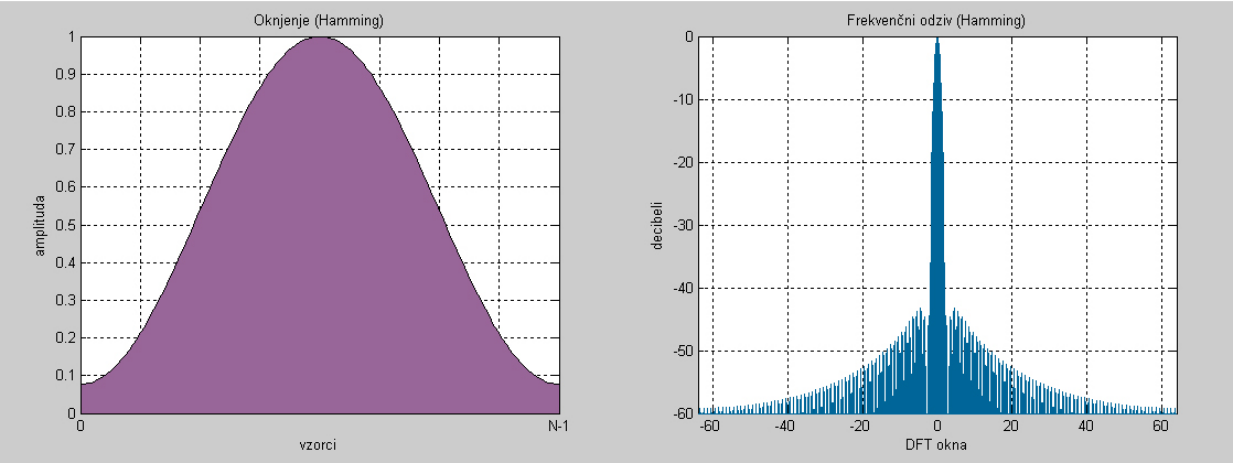


Slika 17: Odnos med izvornim podatkovnim tokom, velikostjo okvirja in zamikom. Slika povzeta po [5].

Dodatno se signal v vsakem oknu množi z okensko funkcijo. Okenskih funkcij je več, pri razpoznavanju govora se običajno uporablja okenska funkcija $W(n)$, dolžine N (*velikost okna*), ki ima naslednjo formulo:

$$W(n) = (1 - a) - (a \cdot cos(\frac{2\pi n}{N-1})). \tag{14}$$

Spremenljivko a nastavimo z lastnostjo PROP_ALPHA. Pri vrednosti $a = 0,46$ dobimo Hammingovo okno (*Hamming window*), ki je prikazano na sliki 18, pri vrednosti $a = 0,5$ Hanningovo okno (*Hanning window*), ki je na sliki 6 narisano z rdečo barvo, pri vrednosti $a = 0$ pa pravokotno okno.



Slika 18: Hammingovo oknjenje in frekvenčni odziv. Slika je ustvarjena s pomočjo programa Matlab.

5.1.5 Komponenta edu.cmu.sphinx.frontend.transform.DiscreteFourierTransform

Komponenta *DiscreteFourierTransform* računa diskretno Fourierjevo transformacijo (DFT) nad vhodnimi podatki z uporabo hitre Fourierjeve transformacije (FFT). Izračun DFT je v tem primeru samo amplitudni spekter, ki ga uporabljamo pri razpoznavanju govora, fazni spekter pa zavržemo. Vsaka vrednost vrnjenega spektra predstavlja moč frekvenc za trenutno okno. Če imamo npr. frekvenco vzorčenja $44100Hz$, je frekvenčni rang enak $22050Hz$, posamezen stolpec v spektrogramu je razdeljen na 512 frekvenčnih območij in posamezno območje bo široko $22050/512 = 43.066Hz$ [29].
Število FFT točk izberemo tako, da je potenca števila 2, ki je enaka ali večja od števila otipkov v oknu. Število FFT točk je možno nastaviti z lastnostjo PROP_NUMBER_FFT_POINTS. Dolžina vrnjenega spektra je število FFT točk, deljeno z 2, plus 1, to pa zato, ker je vhodni signal realen, je FT simetrična in zato lahko vzamemo eno polovico FFT vrednosti.
Vsak klic *getData()* nam vrne amplitudni spekter od enega okna. Za prikaz spektrograma celotnega opazovanega zvočnega signala, moramo zbirati spektre vseh posameznih oken signala (DFT je natančneje razložena v poglavju 2.2, FFT pa v poglavju 2.4).

5.2 Sprotno razporejanje podatkov

Tako pri primerih programov iz osnovnega paketa Sphinx-4 kot tudi ostalih primerih, dostopnih na svetovnem spletu, nismo zasledili implementacije sprotnega prikazovanja spektrograma s pomočjo sistema Sphinx-4. V našem primeru je pridobivanje in razporejanje podatkov z mikrofona rešeno tako, kot določa naslednja psevdo koda:

Algoritem 4 Pridobivanje in razporejanje podatkov z vira

```
vir← inicializacija
Data izhod← frontend.getData();
while izhod≠ DataEndSignal do
  if izhod= DoubleData then
    izrisovanje
  end if
  izhod← frontend.getData();
end while
```

Kot lahko razberemo, algoritem 4 najprej vzpostavi komunikacijo z vhodnim virom (npr. mikrofon, zvočna datoteka, slika). Ustvari se objekt tipa *Data*, v našem primeru ima ime *izhod*. V zanko vstopamo s pogojem, da se ponavlja, dokler *izhod* ni enak DataEndSignal, ki mu sporoča, da zaključi z branjem. Znotraj zanke je še "varovalka" (pogojni stavek), ki preprečuje, da bi prišlo do nepričakovane prekinitve zaradi morebitne izjeme (spremembe tipa spremenljivke *izhod*).

5.3 Izrisovanje spektrograma

Izris spektrograma poteka v treh korakih, ki so prikazani na sliki 19: pred-procesiranje podatkov, razporejanje in priprava podatkov za izris, izbira barvne sheme in izris spektrograma. Predprocesiranje se izvede s pomočjo vhodnega vmesnika (5.1.2), v katerem signal s podatkovnega vira obdelamo s pomočjo podatkovnih procesorjev in ga pripravimo za nadaljnjo uporabo glede na potrebe. Razporejanje in priprava podatkov za izris ter izbira barvne sheme in izris spektrograma sta natančneje predstavljena v nadaljevanju. Predobdelava signalov je že bila predstavljena v prejšnjih poglavjih (poglavji 4 in 5).

Slika 19: Slika poenostavljeno prikazuje osnovne tri korake za izris spektrograma; pred-procesiranje podatkov, razporejanje in priprava podatkov za izris, izbira barvne sheme in izris spektrograma.

5.3.1 Razporejanje in priprava podatkov za izris

Kot je opisano v poglavju 5.2, sprejemamo podatke od vhodnega vmesnika, dokler nam sistem Sphinx-4 ne signalizira konec z *DataEndSignal*. Signal je razdeljen na podatkovne kose. Vsak podatkovni kos je predstavljen s številom vrednosti, ki je enako polovici števila FFT točk. Pri našem programu smo npr. določili število FFT točk 512, torej imamo 256 vrednosti, s katerimi je določena magnituda frekvenčnega spektra za podatkovni kos. Izračunamo logaritem vsake vrednosti in vse dobljene vrednosti, ki so manjše od 0, zanemarimo ter jih statično nastavimo na 0. Intenziteta vsake vrednosti se določa dinamično, glede na najvišjo magnitudo, ki jo zaznamo med obdelavo podatkov. To nam omogoča, da intenzivnost barv spektrograma prilagajamo energiji signala (v tihih prostorih lahko izrišemo tihe zvoke).
V trenutku, ko bomo preglasili trenutno najvišjo intenziteto, bo ta zamenjana z novo in celoten izris se bo prilagodil novi najvišji vrednosti. Dobljene vrednosti shranjujemo v dvodimenzionalno polje (matriko), iz katerega s pomočjo barvnih shem izrisujemo spektrogram.

5.3.2 Izrisovanje

Po vsakem razporejanju in pripravi podatkov dobimo na vhodu dvodimenzionalno polje z imenom $intensitiesList\,[x]\,[y]$, ki hrani obdelane vrednosti, pripravljene za izris. Vrednosti x predstavljajo števila oken za izris, y pa število otipkov posameznega okna. Pred vsakim izrisom je potrebno iz polja odstraniti najstarejšo vrednost x s pripadajočimi vrednostmi y in dodati najnovejšo – tako dobimo polje s statično določenim številom zadnjih oken za izris.
Pred vstopom v zanko, ki bere vrednosti iz polja je potrebno ustvariti objekt tipa *BufferedImage*, v katerega sprotno shranjujemo sliko in ga poimenujemo *spectrogram*. V prvo zanko vstopimo s pogojem, da se branje vrednosti x izvaja dokler ne pridemo do zadnje. Za vsako prebrano x vrednost vstopimo v novo zanko s pogojem, da se branje vrednosti y izvaja dokler ne pridemo do zadnje. Po vsakem branju vrednosti y, s pomočjo barvne sheme izračunamo RGB vrednosti (0-255) za otipek ter izračunano barvo shranimo v objekt *spectrogram*.
Kot lahko razberemo iz algoritma 5, po prebranih vrednostih y za vsak x, kličemo funkcijo *repaint();*, ki poskrbi za to, da se slika iz *spectrogram* sproti izrisuje. Za delovanje funkcije *repaint();*, je bilo potrebno osnovno proceduro *paintComponent(Graphics g)* prepisati tako, da izrisuje trenutno sliko, ki je shranjena v *spectrogram*.
Izris poteka ločeno, v lastni niti, ki mu omogoča, da izrisuje trenutni spektrogram, med računanjem vrednosti za naslednji izris. To pomeni, da morajo izračun, razporejanje podatkov in izbira barvne sheme trajati manj kot izris, v nasprotnem primeru se začnejo podatki kopičiti in izris zaostaja.

Algoritem 5 Izrisovanje

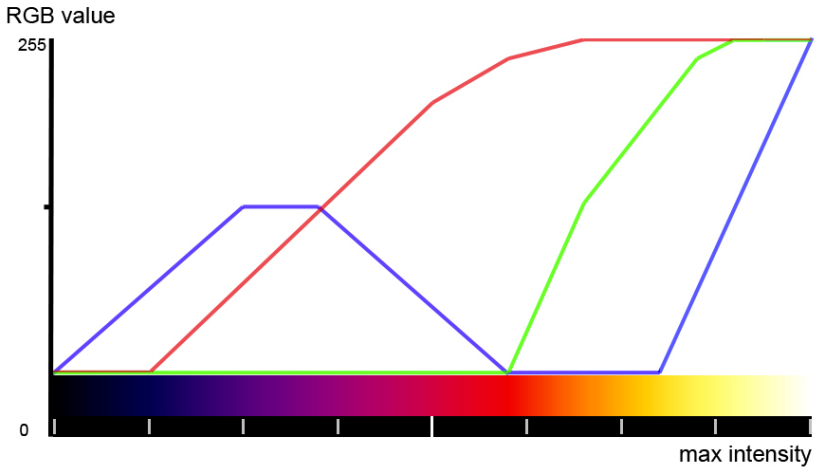
```
if data= DoubleData then
  i← 0;
  intensitiesList[x] [y] ← AddNewIntensities(data);
  intensitiesList[x] [y]← RemoveLastX;
  BufferedImage spectrogram← newBufferedImage(width, height);
  while i< x do
    i← i + 1;
    j← 0;
    while j< y do
      j← j + 1;
      pixel← getRGB(intensitiesList[i] [j]);
      spectrogram.setRGB(pixel);
    end while
    repaint();
  end while
  data← frontEnd.getData();
end if
```

5.3.3 Barvne sheme

Barvne sheme uporabimo za prikaz magnitude spektrograma [15]. Za izdelavo barvnih shem je potrebno določiti *RGB model* (Red, Green, Blue), po katerem se barve spreminjajo v odvisnosti od intenzitete. RGB model predstavlja kombinacijo rdeče, zelene in modre barve, s pomočjo katerih lahko dobimo več kot 16.77 milijonov barv [3] [7]. Vsaka barva je opisana z 256 vrednostmi, od 0 do 255.
Definirali smo tri različne barvne sheme: shemo s sivinami, črno-zeleno shemo ter shemo po meri, ki smo jo poimenovali *iSound shema* in je prikazana na sliki 20. S pravilno izbiro barvne sheme lahko značilnosti signala bolje izpostavimo in prilagodimo potrebam. Obstajata dva načina določanja RGB vrednosti: *linearni*, s pomočjo katerega RGB vrednosti naraščajo/padajo linearno z naraščanjem/padanjem intenzitete signala, in *logaritemski*, ki računa logaritemske vrednosti vhodnega signala, po katerih naraščajo/padajo RGB vrednosti.

iSound shema

Za izdelavo iSound sheme smo v programu Photoshop CS2 izdelali barvni spekter, ki je prikazan na sliki 20 pod x osjo, in določa RGB vrednosti v odvisnosti od intenzitete.



Slika 20: Slika prikazuje RGB vrednosti v odvisnosti od intenzitete posameznega okna za iSound. Os y prikazuje RGB vrednosti, ki jih dobimo ob določeni intenziteti.

6 Delovanje postopka za sprotno prikazovanje spektrograma

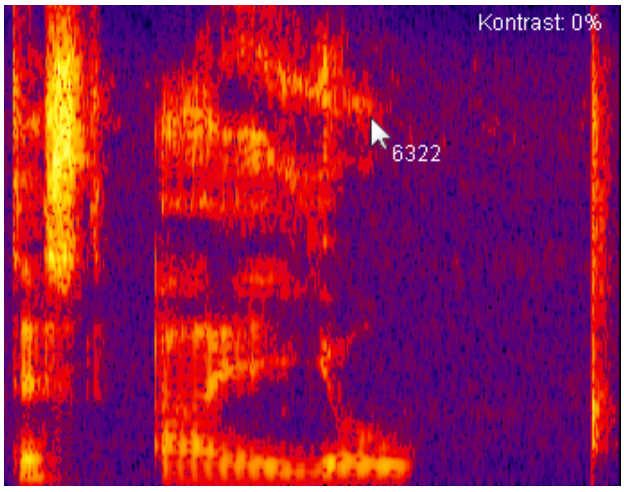
6.1 Predstavitev uporabniškega vmesnika

V naslednji poglavjih je opisan uporabniški vmesnik našega programa in z njim povezane funkcije. Vodilo pri izdelavi uporabniškega vmesnika je bilo preprosto upravljanje programa in funkcionalen izgled, ki bo uspešno sprotno izrisoval spektrogram.

6.1.1 Nastavitev parametrov pred zagonom

Pred zagonom je mogoče za vsak modul nastaviti lastnosti, ki so določene kot parametri pri ustvarjanju konstruktorja modula. Nekatere osnovne lastnosti za posamezne module so opisane v prejšnjih poglavjih, v našem programu so dodane še štiri nove konstantne spremenljivke, ki so poimenovane *WIDTH*, *FREEZE_PICTURE*, *COLOR_OPTION*, *SHOW_INFO*. Funkcije spremenljivk in dogodki, ki so vezani na njih so natančneje opisana v poglavju 6.1.3.
Spremenljivka *WIDTH* je tipa *integer* in določa širino področja (v slikovnih točkah) znotraj okna, v katerem se bosta izrisovala običajni zvočni signal in spektrogram. Nastavitev širine ne predstavlja ovire pri povečanju okna med samim izvajanjem izrisa spektrograma in običajnega zvočnega signala, kajti s povečanjem se nastavljena širina in višina linearno raztegneta.
Spremenljivka *FREEZE_PICTURE* je tipa *boolean* in določa stanje dogodka ob levem kliku na področje izrisa spektrograma.
Spremenljivka *COLOR_OPTION* je tipa *int* in omogoča izbiro privzete barvne sheme pred zagonom.
Spremenljivka *SHOW_INFO* je tipa *boolean*, omogoča vklop/izklop prikaza označene y osi in stopnje kontrasta.

6.1.2 Postavitev elementov



Slika 21: Izgled uporabniškega vmesnika in postavitev elementov.

Slika 21 prikazuje izgled našega programa med izrisovanjem spektrograma v iSound barvni shemi, kjer temno vijolična barva označuje najnižjo magnitudo signala, rumena najvišjo. V tem primeru lahko vidimo, da je funkcija za prikaz podrobnosti med izrisovanjem spektrograma vklopljena, ker je v desnem kotu zgoraj prikazana stopnja kontrasta in ob miškinem kazalcu desno spodaj frekvenca za določeno področje izrisa. Namenoma smo se izogibali gumbom, menijem in ostalim elementom iz *Java.awt.** in *Java.swing.** knjižnic zaradi preglednosti in enostavnosti.

6.1.3 Osnovne funkcije in dogodki

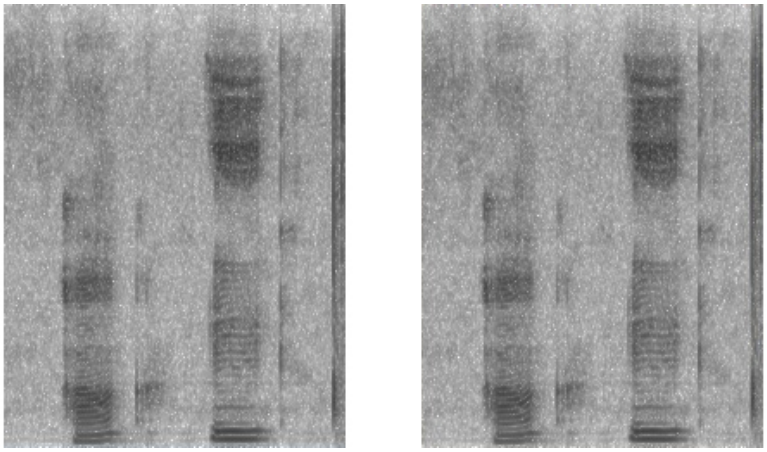
Funkcije aplikacije so razdeljene smiselno na dve podskupini. V prvo skupino spadajo funkcionalnosti, ki so že opisane v prejšnjih poglavjih (npr. pridobivanje signala iz mikrofona ali izris spektrograma) in jih lahko poimenujemo *osnovne funkcije*.
V drugo skupino spadajo funkcionalnosti, ki se sprožijo ob dogodkih (*ang.* events). V našem programu so implementirani štiti dogodki: zamrznitev in odmrznitev izrisovanja spektrograma, preklapljanje med barvnimi shemami med izrisovanjem spektrograma, spreminjanje kontrasta med izrisovanjem spektrograma, prikaz podrobnosti med izrisovanjem spektrograma in v zamrznjenem stanju.
Zamrznitev in odmrznitev izrisovanja spektrograma lahko omogočimo tako, da pred zagonom prevajanja programa nastavimo spremenljivko *FREEZE_PICTURE*. Če je postavljena na vrednost *true*, je zamrznitev slike omogočena in ob levem kliku na področje, kjer se izrisuje spektrogram, se izrisovanje novih vrednosti začasno ustavi in nadaljuje ob ponovnem desnem kliku. Postavitev na vrednost *false* pomeni, da je dogodek na levi klik izklopljen.
Preklapljanje med barvnimi shemami med izrisovanjem spektrograma lahko omogočimo tako, da pred zagonom prevajanja programa ali v konfiguracijski datoteki nastavimo spremenljivko *SELECT_COLOR*. Če je postavljena na vrednost *true*, je spreminjanje barvnih shem med izrisovanjem spektrograma omogočeno in ob desnem kliku na področje, kjer se izrisuje spektrogram, se barvna shema spremeni. V našem programu smo implementirali tri barvne sheme, vendar je omogočena enostavna nadgradnja, s katero lahko dodajamo poljubno mnogo shem. Postavitev na vrednost *false* pomeni, da je dogodek na desni klik izklopljen.
Spreminjanje kontrasta med izrisovanjem spektrograma je v osnovi omogočena funkcionalnost, ki jo upravljamo z vnosom znakov "+" in "-" (s "+" povečujemo stopnjo kontrasta, z "-" jo zmanjšujemo). Osnovna ideja je, da z večanjem in manjšanjem kontrasta izboljšamo kakovost sprotnega prikazovanja (med samim izrisom) in tako izpostavimo karakteristike obdelovanega signala.
Prikaz podrobnosti med izrisovanjem spektrograma je omogočeno ob pritisku znaka "i". Prikaže nam označeno os y (kHz) ter stopnjo kontrasta v %.

6.2 Analiza delovanja programa

Analizo delovanja našega programa smo izvedli v treh delih. V *prvem delu* je bila izvedena primerjava programa iSound s programom AudioTool iz Sphinx-4 paketa. Program AudioTool najprej zajame zvok ter ga šele naknadno analizira in izriše spektrogram ter osnovni signal. Namen tega je izpostaviti razlike ter uporabnost, kar bo pripomoglo pri dokazovanju zastavljene hipoteze. V *drugem delu* smo izvedli primerjavo našega programa z nekaterimi drugimi programi za sprotno analizo signala. Preverjali smo štiri sklope lastnosti. Namen je bil prikazati primerljivost našega programa z nekaterimi drugimi programi za sprotno analizo signala in si natančneje ogledati razlike med njimi. Primerjali smo način konfiguracije, preglednost uporabniškega vmesnika, podprte funkcije in obremenitev računalnika.
V *tretjem delu* pa smo izvedli meritve ter izpostavili sistemske zahteve programa iSound, kar je osnova za opredelitev področij izboljšave.

6.2.1 Primerjava programa iSound s programom AudioTool

Pri primerjavi programa iSound s programom AudioTool je bilo potrebno upoštevati, da AudioTool ni namenjen za sprotno analizo in obdelavo signala. Potrebno je bilo pokazati, da s pomočjo programa iSound na račun sprotnega izrisa spektrograma ne izgubimo na kakovosti pri izrisu. Zato smo primerjali izris, ki je nastal pri obeh programih pod enakimi pogoji – analizirali smo enak podatkovni vir in imeli enake nastavitve. S programom iSound smo povečali uporabnost analize z izrisom spektrograma tako, da smo dodali možnost izbire barvnih shem, poenostavili navigacijo med izrisovanjem in omogočili prikaz podatkov.



Slika 22: Leva slika prikazuje izris spektrograma v AudioTool, desna slika prikazuje izris spektrograma v našem programu.

Če bi želeli dokazati, da sta spektrograma s slike 22 enaka bi bilo potrebno amplitudne vrednosti obeh programov med izrisovanjem izpisovati v datoteko ter ju s pomočjo matrike odšteti.
Spektrograma s slike 22 sta nastala tako, da smo z istega mikrofona (podatkovnega vira) s pomočjo programov AudioTool in iSound zajemali zvočni signal pod enakimi pogoji, ki so predstavljeni v tabeli *Nastavitve parametrov za AudioTool in iSound*. Pogoji, ki smo jih določili za posamezne podatkovne procesorje, so predstavljeni kot parametri razredov: Microphone, Preemphasizer, Windower, FFT. Frekvenca vzorčenja je $16000Hz$, kar ustreza frekvenčnemu razponu od 0 do 8000Hz, vsak podatkovni kos iz mikrofona je velik 10ms in vsak vzorec je opisan s 16 biti. Za oknjenje smo določili velikost okna na 25.625ms, zamik okna pa 10ms, število odtipkov je 512.

Nastavitve parametrov za AudioTool in iSound

Microphone	**Preemphasizer**	**Windower**	**FFT**
frekvenca vzorčenja *16000Hz* velikost pomnilnika *10ms* bitov na vzorec *16bit*	preemphasis faktor *0.97*	velikost okna *25.625ms* zamik okna *10ms*	število odtipkov *512*

Primerjava AudioTool in iSound

lastnosti	**AudioTool**	**iSound**
Barva spektrograma	sivine	sivine in barvni izris
Izboljšava izrisa	-	povečanje kontrasta med izrisovanjem spektrograma
Konfiguracija	pred prevajanjem	pred zagonom in prevajanjem
Navigacija	meni, gumbi	vnos znakov, levi in desni klik
Podatki o izrisu	-	podatki za y os

Pri primerjavi programov AudioTool in iSound smo analizirali pet področij, ki so predstavljena v tabeli *Primerjava AudioTool in iSound*, in sicer: barvo spektrograma, možnost izboljšave izrisa, način konfiguracije in navigacije ter podatke o izrisu.

Program AudioTool:

- Izrisuje spektrogram v sivinah, kar zadovolji osnovne kriterije. S to metodo izrisa ni mogoče s prostim očesom določiti, v katero območje intenzitete spada posamezen odtipek.
- Ne podpira izboljšave izrisa, kar je slabo, če npr. snemamo posnetek, ki traja eno uro, in šele po tem ugotovimo, da ga je potrebno izboljšati in da ta funkcionalnost ni podprta, imamo opcijo, da analiziramo shranjen posnetek v drugem programu ali posnamemo drugega pod boljšimi pogoji.
- Podpira način konfiguracije pred prevajanjem, kar ni dobro v primeru, ko želimo spreminjati konfiguracijo po tem, ko je program že preveden.
- Za navigacijo uporablja gumbe in meni, ki sta zelo razširjena elementa, njihova uporaba je intuitivna, tako se povprečni uporabnik hitro nauči uporabljat program.
- Ne podpira prikaza kakršnihkoli oznak ali podatkov o izrisu, osi niso označene.

Program iSound:

- Podpira izris spektrograma v sivinah in barvah ter logaritemski in linearni skali. Izris v barvah omogoča določitev približne intenzitete otipkov.
- Omogoča izboljšavo izrisa z določanjem kontrasta, ki ga lahko spreminjamo med izrisovanjem.
- Podpira način konfiguracije pred prevajanjem in zagonom.
- Uporablja za navigacijo vnos znakov ter levi in desni klik, kar je lahko za povprečnega uporabnika nejasno.
- Omogoča prikaz stopnje kontrasta in frekvenco v Hz.

6.2.2 Primerjava našega programa z nekaterimi drugimi programi za sprotno analizo signala

Pri primerjavi našega programa z nekaterimi drugimi programi za sprotno analizo zvočnih signalov smo si izbrali tri programe [23], ki imajo podobne funkcionalnosti kot naš program. To so: SFS/RTSPECT Version 2.4 [25], SFS/RTGRAM Version 1.3 [26], Wave surfer 1.8.8 [30].

Konfiguracija

	način konfiguracije	
program	**pred zagonom**	**med delovanjem**
SFS/RTSPECT Version 2.4	ni podprto	mono/stereo, preemphasis, najvišja frekvenca, frame rate, razpon magnitude
SFS/RTGRAM Version 1.3	ni podprto	najvišja frekvenca, ozkopasovni/širokopasovni, čas na piksel, dinamični rang, barvno/sivine
Wave surfer 1.8.8	ni podprto	nastavitve za določanje komand z znaki, nastavitve za ime in tip datoteke, nastavitve za prilagoditev uporabniškega vmesnika, zvočne nastavitve
iSound	omogočena zamrznitev slike, širina, omogočena izbira barve, nastavitve mikrofona, nastavitve za hitro FT, nastavitve za oknjenje in še veliko drugih	ni podprto

Način konfiguracije določa, kako lahko nastavimo parametre, ki določajo način izrisovanja spektrograma. Nastavimo jih lahko pred zagonom in med delovanjem programa. Pri programih SFS/RTSPECT Version 2.4, SFS/RTGRAM Version 1.3 in Wave surfer 1.8.8 konfiguracija pred zagonom ni podprta. Med delovanjem programa imajo omenjeni programi veliko nastavitvenih možnosti, ki so izpostavljene v tabeli *Konfiguracija*. Pri programu iSound je omogočeno nastavljanje parametrov pred zagonom, speminjanje nastavitev med delovanjem pa zaenkrat še ni podprto.

Vizualno

program	**spektrogram**
SFS/RTSPECT Version 2.4	ne izrisuje spektrograma – magnituda v odvisnosti od frekvence za dani trenutek, označene osi
SFS/RTGRAM Version 1.3	frekvenca v odvisnosti od časa z magnitudo označeno s sivinami, označene osi
Wave surfer 1.8.8	frekvenca v odvisnosti od časa z magnitudo označeno s sivinami, označene osi
iSound	frekvenca v odvisnosti od časa z magnitudo označeno s sivinami/barvno, označene osi

Kot je razvidno iz tabele *Vizualno*, program *SFS/RTSPECT Version 2.4* ne izrisuje spektrograma, zato bomo pri podrobnejši primerjavi izrisa spektrograma upoštevali ostale tri programe. Program Wave surfer 1.8.8 se je med testiranjem najbolje odrezal, saj je med programi, ki smo jih primerjali najbolj dodelan.

SFS/RTGRAM Version 1.3:

- Izrisuje spektrogram v sivinah.
- Ne izrisuje spektrogram tekoče, temveč po posameznih kosih (10px).

Wave surfer 1.8.8:

- Izrisuje spektrogram v sivinah.
- Izrisuje spektrogram tekoče, vendar posnetek, ki ga sprotno prikazuje snema le 1min. Po tem se ustavi.

iSound:

- Izrisuje spektrogram v sivinah in barvno.
- Izrisuje spektrogram tekoče in neomejeno dolgo. Ne omogoča vpogleda v zgodovino, starejšo od tiste, ki je prikazana v oknu.

Podprte funkcije

	funkcije	
program	**med izrisovanjem**	**v mirovanju**
SFS/RTSPECT Version 2.4	pavza	izrisovanje, tiskanje, nastavitev parametrov
SFS/RTGRAM Version 1.3	pavza	izrisovanje, nastavitev parametrov, predvajanje posnetka, izris mreže
Wave surfer 1.8.8	pavza	izrisovanje, nastavitev parametrov, predvajanje posnetka, snemanje, urejanje posnetka, povečava in še veliko drugih
iSound	pavza, prikaz podatkov za y os in stopnje kontrasta, preklapljanje med barvnimi shemami, spreminjanje kontrasta	izrisovanje, prikaz podatkov za y os in stopnje kontrasta

Kot je razvidno iz tabele *Podprte funkcije*, so funkcije za upravljanje z izrisom spektrograma med izrisovanjem slabo podprte – poleg pavze pri ostalih programih ni druge funkcije, ki bi omogočala uporabniku, da vpliva na sprotni izris spektrograma ali kakorkoli drugače upravlja s programom med izrisovanjem. Program iSound temelji na ideji, da je potrebno pri dodajanju novih funkcij, čim več funkcij prilagoditi za upravljanje sprotnega izrisa spektrograma. Omogočeno je prikazovanje podatkov za y os ter stopnje kontrasta, preklapljanje med barvnimi shemami in spreminjanje kontrasta.

6.3 Sistemske zahteve programa

V tem poglavju smo merili sistemske zahteve programa in izvajali primerjavo z ostalimi programi. Cilj meritev je bil, da s pomočjo primerjave programov ugotovimo, ali je naš program primerljiv z ostalimi naštetimi programi, v čem je boljši/slabši in kakšna je obremenitev procesorja, če onemogočimo izrisovanje spektrograma ter izvajamo le računske operacije.
Na podlagi meritev iz poglavja 6.3.1 in analize iz poglavja 6.2 bomo v poglavju 6.3.2 ugotovili, kaj je treba izboljšati za hitrejše procesiranje podatkov, manjšo obremenjenost grafičnega procesorja in katere funkcije bi lahko še dodali, da bi povečali uporabnost programa.
Programi so testirani na delovni postaji z operacijskim sistemom Windows 7 Home Premium, service pack 1, s procesorjem Intel Core i3 2.13GHz, spominom 4GB in grafično kartico NVIDIA GeForce GT 320M s posodobitvjo 1.5.20, različico PsihX 9.11.0621 in gonilniki 285.62, ki se uvršča med slabše grafične kartice srednjega razreda [34].

6.3.1 Meritve obremenitev računalnika

Pri meritvah obremenjenosti računalnika smo izvajali meritve za vsak program posebej. Preverili smo, v katerem programskem jeziku je program napisan, koliko obremeni procesor in spomin ter koliko niti se izvaja med delovanjem. Programe smo pognali ter jih eno uro različno obremenjevali in opazovali, kako se spreminjajo vrednosti.
Kot je razvidno iz tabele *Obremenitev računalnika*, sta SFS/RTSPECT Version 2.4 in SFS/RTGRAM Version 1.3 napisana v programskem jeziku Microsoft Visual C++, Wave surfer 1.8.8 v TCLTK. Program SFS/RTSPECT Version 2.4 izstopa po nizki obremenjenosti procesorja in spomina. S tem namenom je bil tudi uvrščen med programe za testiranje, da bi pokazali, koliko bolj so obremenjeni procesorji pri izrisu spektrograma.
iSound izstopa po visokem številu niti, razlog za to je programski jezik Java.

Obremenitev računalnika

		obremenitev		
program	**programski jezik**	**procesor**	**spomin**	**št. niti**
SFS/RTSPECT Version 2.4	Microsoft Visual C++	0.11% - 1%	9.248MB	5
SFS/RTGRAM Version 1.3	Microsoft Visual C++	24% - 26%	70.33MB	6 - 8
Wave surfer 1.8.8	TCLTK	7% - 11%	30.12MB	6-9
iSound	Java	12% - 17%	62.452MB	27 - 31

Dodatno smo preverili tudi, koliko se spremeni obremenjenost procesorja pri delovanju programa iSound, če onemogočimo izris spektrograma ter izvajamo le računske operacije. Računanje brez izrisa porabi približno stokrat manj procesorskega časa in se giblje od 0.12% do 0.3% obremenjenosti procesorja.

6.3.2 Izboljšave

Na podlagi analize iz poglavja 6.2 in meritev iz poglavja 6.3 lahko ugotovimo, katere segmente za sprotni izris spektrograma je mogoče izboljšati in s tem povečati uporabnost ali razbremeniti procesor. Večina težav je povezanih z vizualizacijo, razporejanjem velike količine podatkov in programskim jezikom Java.
Izboljšave lahko smiselno razdelimo v dve skupini: dodajanje funkcij in pohitritev izrisa.
Izboljšave z dodajanjem funkcij lahko razdelimo na dve podskupini: funkcije za uporavljanje s programom pred izrisom in funkcije za uporavljanje s programom po izrisu.
Izboljšave za pohitritev izrisa lahko razdelimo na tri podskupine: pohitritev računskih procesov, pohitritev razporejanja podatkov in pohitritev vizualizacije.

Dodajanje funkcij

Izboljšavi z dodajanjem funkcij za upravljanje s programom pred izrisom in po izrisu sta zelo pomembni. Funkcija za upravljanje s programom med izrisovanjem pa omogoča boljšo vizualizacijo, prikaz podatkov o signalu in opozoril ter omogoča boljšo navigacijo. Implementacija dodatnih funkcij zahteva več procesorskega časa, pri čemur nas najbolj omejujeta hitrost procesorja grafične in zvočne enote ter s tem povezano zaostajanje sprotnega prikazovanja signala.
Funkcij, s katerimi lahko opremimo iSound, je veliko: filtriranje signala v času in frekvenci, določitev frekvenčnega razpona, ki ga želimo opazovati, odstranjevanje šuma ipd.

Pohitritev izrisa

Pohitritev izrisa je mogoče izvesti na več načinov, in sicer kot: pohitritev računskih procesov (FFT, oknjenje), pohitritev razporejanja podatkov, pohitritev vizualizacije.
Na področju *pohitritve računskih procesov* smo omejeni z dejstvom, da je Sphinx-4 zgrajen iz manjših modulov, ki so med seboj odvisni. Če izhajamo iz tega, bi spremembe razredov iz paketa Sphinx-4 lahko pomenile prilagajanje celotnega okolja novemu konceptu.
Pohitritev razporejanja podatkov lahko dosežemo z optimizacijo, ki bi bila osredotočena predvsem na zmanjšanje števila nepotrebnih zank in pogojnih stavkov med izrisovanjem (npr. preverjanje najvišje intenzitete pred izrisovanjem vsakega okna). Nad prihajajočimi podatki je potrebno izvesti računske operacije samo enkrat in jih dodati v polje z obdelanimi podatki. Pri vsakem izrisu ne smemo obdelovati vseh podatkov.
Za *pohitritev vizualizacije* je potrebno predhodno prilagoditi način razporejanja podatkov. Če vzamemo za primer program SFS/RTGRAM Version 1.3 [26], lahko opazimo pri izrisu spektrograma, da se ne izrisuje tekoče, temveč po posameznih kosih (tako izris zamuja do nekaj $10ms$). Za tako vrsto izrisa je potrebno tudi razporejanje podatkov prilagoditi, in sicer tako, da zbira podatke v podatkovne kose s fiksno dolžino ter jih izrisuje, ko se posamezni podatkovni kosi napolnijo s podatki. Postopek, s katerim želimo zmanjšati število podatkov za izris tako, da lahko s pridobljenimi podatki še vedno izrišemo uporaben spektrogram imenujemo *interpolacija vizualizacije*.

7 Sklepi

V zaključni nalogi je predstavljena izvedba postopka za sprotno prikazovanje spektrograma, v kateri so predstavljene knjižnice, ki smo jih uporabili pri našem projektu, in primer konfiguracijske datoteke, v kateri lahko vse parametre vključenih knjižnic poljubno nastavljamo pred zagonom.
Izris spektrograma smo razdelili na tri korake ter podrobneje obrazložili vsakega posebej: predprocesiranje podatkov, razporejanje in priprava podatkov za izris, izbira barvne sheme in izris spektrograma. Opisali smo lastno barvno shemo iSound tako, da smo določili spreminjanje RGB vrednosti v odvisnosti od intenzitete. S pomočjo iSound barvne sheme je mogoče lažje razbrati intenziteto za posamezen otipek.
Pri opisu programa iSound smo prikazali in opisali izgled, funkcionalnosti ter izvedli primerjavo programa iSound s programom AudioTool iz Sphinx-4 paketa z namenom, da ugotovimo ali se izris spektrograma našega programa razlikuje od izrisa spektrograma, ki ga ustvari program AudioTool. Primerjava izrisov kaže popolnoma enakovredne informacije o zvočnem signalu. S tem smo tudi potrdili zastavljeno hipotezo.
Pri raziskovanju odprtokodnega sistema Sphinx-4 lahko povzamemo, da so pri njegovem razvoju sodelovali izkušeni programerji ter raziskovalne in izobraževalne ustanove, saj ima sistem dobro zastavljen koncept, ki omogoča enostavno razširljivost. Kljub temu je pri podrobnejšem pregledu izvorne kode opaziti površnost in nedodelanost nekaterih funkcij, ob katerih so zakomentirana obvestila, da jih je treba dodelati ali zamenjati z bolj učinkovitimi funkcijami. Če se bo sistem razvijal, lahko sklepamo, da bodo obstoječe pomankljivosti odpravljene in bo zagotovo eden izmed najbolj uspešnih tovrstnih odprtokodnih sistemov.
Izvedli smo tudi primerjave programa iSound z nekaterimi drugimi sorodnimi programi. Ugotovili smo, da čeprav je iSound napisan v programskem jeziku Java in je v fazi razvoja, da je primerljiv in konkurenčen, kar je razvidno iz tabele *Obremenitev računalnika* iz poglavja 6.3.1. Pri analizi smo ugotovili, da pri ostalih programih med samim izrisom ni mogoče vplivati na izris ali kakorkoli drugače spreminjati nastavitve. Pri vseh izvedenih meritvah je bil program iSound uspešen in stabilen. Testiran je bil na različnih operacijskih sistemih; primerjava je pokazala, da program iSound ne glede na operacijski sistem deluje enako, tudi pod pogojem, da ga prevedemo in poženemo na različnih operacijskih sistemih. Med iskanjem podobnih programov nismo zasledili na internetu nobenega, ki bi bil napisan v programskem jeziku Java in bi sprotno izrisoval spektrogram.
Produkt zaključne naloge je merilno orodje, ki ima širok nabor nastavljivih parametrov pred prevajanjem, kot tudi pred zagonom. iSound je zanimiv za mnoga raziskovalna področja in uporaben v indrustriji, kjer so potrebe po uporabi časovno-frekvenčne analize signalov, kot npr. pri akustični analizi zvočnih signalov, sistemih za razpoznavanje in tvorjenje govora, analizi govora, vibroakustiki itd. Prednost programa iSound je fleksibilnost, ki je privzeta iz Sphinx-4 sistema in omogoča nadgradnjo, izboljšavo osnovnih funkcij in dodajanje novih, kot je npr. sprotno prepoznavanje govora.

Literatura

[1] Birdsongs.it. `http://www.birdsongs.it/index.asp`, 2011.

[2] B. Boashash. Estimating and interpreting the instantaneous frequency of a signal. i. fundamentals. *Proceedings of the IEEE*, 80(4):520–538, 1992.

[3] N. Boughen. *LightWave 3D 7.5 lighting*. Wordware Publishing, Inc., 2003.

[4] Carnegie Mellon University cmusphinx.sourceforge.net. `http://cmusphinx.sourceforge.net/sphinx4/javadoc/edu/cmu/sphinx/frontend/package-summary.html`, 2011.

[5] Carnegie Mellon University cmusphinx.sourceforge.net. `http://cmusphinx.sourceforge.net/sphinx4/javadoc/edu/cmu/sphinx/frontend/window/RaisedCosineWindower.html`, 2011.

[6] National Instruments Corporation. `http://zone.ni.com/reference/en-XX/help/371361B-01/lvanlsconcepts/char_smoothing_windows/`, 2006.

[7] Handprint.com. `http://handprint.com/HP/WCL/color18a.html`, 2009.

[8] J. Žibert. Delovanje sistemov za razpoznavanje govora, 2008. Seminar: Razpoznavanje govora.

[9] J. Žibert. Izpeljava koeficientov melodičnega kepstra za razpoznavanje govora, 2011.

[10] D. Sussillo in A. Kundaje in D. Anastassiou. Spectrogram analysis of genomes. *EURASIP Journal on Applied Signal Processing*, 2004:29–42, 2004.

[11] P. Boersma in D. Weenink. Praat, a system for doing phonetics by computer. *Glot international*, 5(9/10):341–345, 2002.

[12] S. Young in G. Evermann in D. Kershaw in G. Moore in J. Odell in D. Ollason in V. Valtchev in P. Woodland. *The HTK book*, volume 2. Citeseer, 1997.

[13] R. Sundaram in J. Hamaker in J. Picone. Twister: The isip 2001 conversational speech evaluation system. In *Proceedings of the Speech Transcription Workshop*.

[14] F. Mihelič in L. Gyergyek in T. Ebenšpanger. *SIGNALI – priročnik z zbirko rešenih nalog, 3. dopolnjena izdaja*. Založba FE in FRI, 2005.

[15] A. Hard in L. Sivik. A theory of colors in combination–a descriptive model related to the ncs color-order system. *Color: Research and applications*, 26(1):4–28, 2001.

[16] J.A. Simmons in M. Ferragamo in C.F. Moss in S.B. Stevenson in R.A. Altes. Discrimination of jittered sonar echoes by the echolocating bat, eptesicus fuscus: the shape of target images in echolocation. *Journal of Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology*, 167(5):589–616, 1990.

[17] P. Duhamel in M. Vetterli. Fast fourier transforms: a tutorial review and a state of the art. *Signal Processing*, pages 259–299, 1990.

[18] P. Lamere in P. Kwok in W. Walker in E. Gouvea in R. Singh in B. Raj in P. Wolf. Design of the cmu sphinx-4 decoder. In *Proceedings of the 8th European conference on speech communication and technology*, pages 1181–1184. Citeseer.

[19] W. Walker in P. Lamere in P. Kwok in B. Raj in R. Singh in E. Gouvea in P. Wolf in J. Woelfel. Sphinx-4: A flexible open source framework for speech recognition. *Sun Microsystems, Report Number: TR-2004-139*, 2004.

[20] F. Winckel in T. Binkley. *Music, sound and sensation: A modern exposition.* Dover Publications, 1967.

[21] F.A. Saunders in W.A. Hill in B. Franklin. A wearable tactile sensory aid for profoundly deaf children. *Journal of Medical Systems*, 5(4):265–270, 1981.

[22] X. Liu in Y. Zhao in X. Pi in L. Liang in A.V. Nefian. Audio-visual continuous speech recognition using a coupled hidden markov model. In *Seventh International Conference on Spoken Language Processing*, 2002.

[23] M. Huckvale. Internet institute for speech and hearing. `http://www.speechandhearing.net/laboratory/tools.php`, 2010.

[24] Infinite Wave Mastering. `http://src.infinitewave.ca`, 2011.

[25] UCL Division of Psychology and Language Sciences. `http://www.phon.ucl.ac.uk/resource/sfs/rtspect`, 2006.

[26] UCL Division of Psychology and Language Sciences. `http://www.phon.ucl.ac.uk/resource/sfs/rtgram`, 2010.

[27] University of Virginia. `http://faculty.virginia.edu/linganth/Program/LingAnthHomepage.htm`, 2011.

[28] K. Prahallad. Topic: Spectrogram, cepstrum and mel-frequency analysis. `http://www.speech.cs.cmu.edu/15-492/slides/03_mfcc.pdf`, 2011. Speech Technology: A Practical Introduction, 2008.

[29] R. Romero. `http://www.vlf.it/fft_beginners/fft_beginners.html`, 2011.

[30] TMH. Speech, Music and Hearing. `http://www.speech.kth.se/wavesurfer`, 2010.

[31] U.S. Geological Survey. `http://earthquake.usgs.gov/monitoring/spectrograms/24hr/`, 2011.

[32] B. Truax. *The world soundscape project's handbook for acoustic ecology.* ARC Publications, Vancouver, BC, 1978.

[33] Ž. Čučej. *Teorija signalov: digitalni signali in sistemi.* Wordware Publishing, Inc., 2002.

[34] videocardbenchmark.net. `http://www.videocardbenchmark.net/high_end_gpus.html`, 2011.

[35] S.J. Young. The htk hidden markov model toolkit: Design and philosophy. *Department of Engineering, Cambridge University, UK, Tech. Rep. TR*, 153, 1993.

A Priloge

A.1 Program iSound

Zadnjo različico programa iSound si lahko poberete s predstavitvene spletne strani.

- `http://isound.irmanabdic.com`

A.2 Izjava o avtorstvu

UNIVERZA NA PRIMORSKEM
UNIVERSITÀ DEL LITORALE / UNIVERSITY OF PRIMORSKA

FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN INFORMACIJSKE TEHNOLOGIJE
FACOLTÀ DI SCIENZE MATEMATICHE NATURALI E TECNOLOGIE INFORMATICHE
FACULTY OF MATHEMATICS, NATURAL SCIENCES AND INFORMATION TECHNOLOGIES

Glagoljaška 8, SI – 6000 Koper
Tel.: (+386 5) 611 75 70
Fax: (+386 5) 611 75 71
www.famnit.upr.si
info@famnit.upr.si

UNIVERZA NA PRIMORSKEM
UNIVERSITÀ DEL LITORALE
UNIVERSITY OF PRIMORSKA

Titov trg 4, SI – 6000 Koper
Tel.: + 386 5 611 75 00
Fax.: + 386 5 611 75 30
E-mail: info@upr.si
http://www.upr.si

IZJAVA O AVTORSTVU ZAKLJUČNE NALOGE

Spodaj podpisani/a IRMAN ABDIĆ, z vpisno številko [redacted],

vpisan/-a v študijski program RAČUNALNIŠTVO IN INFORMATIKA,

sem avtor/-ica zaključne naloge z naslovom:

IZVEDBA POSTOPKA ZA SPROTNO PRIKAZOVANJE SPEKTROGRAMA V ODPRTOKODNEM SISTEMU SPHINX-4

S svojim podpisom zagotavljam, da je predložena zaključna naloga izključno rezultat mojega lastnega dela. Prav tako se zavedam, da je predstavljanje tujih del kot mojih lastnih kaznivo po zakonu.

V KOPRU, dne 15. 12. 2011

Podpis avtorja/-ice: _____