Aplikacija za detekciju glazbenih nota

Jelavić, Fran

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 330

APLIKACIJA ZA DETEKCIJU GLAZBENIH NOTA

Fran Jelavić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 330

APLIKACIJA ZA DETEKCIJU GLAZBENIH NOTA

Fran Jelavić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 330

Pristupnik:	**Fran Jelavić (0036523987)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	prof. dr. sc. Ivan Đurek

Zadatak: **Aplikacija za detekciju glazbenih nota**

Opis zadatka:

Objasnite teorijsku osnovu rasporeda glazbenih nota prema oktavama i povežite ih s frekvencijama zvučnog signala. Objasnite princip detekcije glazbenih nota analizom spektralnog sadržaja audio signala. Analizirajte kako se spektralni sadržaj mijenja tijekom vremena i komentirajte kako veličina vremenskog prozora utječe na brzinu analize. Načinite programsku implementaciju koja na osnovi frekvencijske i vremenske analize audio signala detektira kako se najglasnije odsvirane glazbene note mijenjaju kroz vrijeme reprodukcije audio signala

Rok za predaju rada: 28. lipnja 2024.

Zahvala mentoru prof. dr. sc. Ivanu Đureku na prijedlogu teme i mogućnosti rada vlastitim tempom.

SADRŽAJ

1. Uvod

Glazba je složena umjetnost koja koristi zvukove organizirane u tonove i ritmove kako bi stvorila skladne kompozicije. Ključnu ulogu u glazbi igra manipulacija frekvencijama koje omogućuju razlikovanje različitih zvukova i melodija. Svaka glazbena nota odgovara određenoj frekvenciji, a te frekvencije organiziraju se u sustave kao što su ljestvice i harmonije, što omogućava stvaranje melodija i akorda.

Ovaj rad istražuje teorijske osnove glazbenih nota i njihovu povezanost s frekvencijama zvučnih signala. Istražuje principe detekcije glazbenih nota analizom frekvencija te kako veličina vremenskog prozora utječe na brzinu analize i točnost prepoznavanja nota. Poseban dio rada posvećen je karakteristikama gitara koje utječu na njihove raspone nota i frekvencija te analizi spektralne karakteristike zvuka.

Cilj ovog rada je pružiti sveobuhvatno razumijevanje teorijskih i praktičnih aspekata detekcije glazbenih nota, kako bi se čitateljima omogućilo dublje razumijevanje načina na koji tehnologija može pomoći u analizi i interpretaciji glazbenih signala.

2. Frekvencija i visina tona

Zvučni valovi, kao i svi longitudinalni valovi, stvaraju se titranjem objekta unutar medija. Izvor zvučnog vala je objekt koji titra, dok je medij okruženje kroz koje taj val putuje. U kontekstu zvučnih valova, titrajući objekt može biti ljudske glasnice, žica koja titra, zvučnik itd., dok je medij obično zrak ili ljudsko uho. Bez obzira na izvor zvučnog vala, čestice medija kroz koji val prolazi kreću se naprijed-nazad, stvarajući područja sabijanja i razrjeđenja pri određenoj frekvenciji.

Frekvencija vala odnosi se na broj titraja čestica medija u jedinici vremena dok val prolazi kroz medij. Mjeri se kao broj potpunih ciklusa (kompresija i razrjeđenja) koje čestice medija naprave u jednoj sekundi, i izražava se u hercima (Hz). Kada se zvučni val kreće kroz medij, sve čestice titraju istim frekvencijama zbog međusobnog utjecaja najbližih susjeda. Frekvencije titranja čestica u mediju jednake su frekvencijama izvora zvučnog vala. Na primjer, žica gitare koja titra na frekvenciji od 500 Hz uzrokovat će titranje čestica u okolini na istoj frekvenciji od 500 Hz, što rezultira prenošenjem zvučnog vala te frekvencije do uha slušatelja.

Ljudsko uho može detektirati zvukove u frekvencijskom rasponu od otprilike 20 Hz do 20 kHz, ali većina glazbenih instrumenata proizvodi tonove unutar užeg raspona. Na primjer, klavir ima raspon od oko 27,5 Hz (nota A_0) do 4186 Hz (nota C_8).

2.1. Visina tona

Visina tona (eng. *pitch*) je osobina zvuka koja omogućuje razlikovanje viših i nižih tonova. U glazbi, visina tona predstavlja subjektivnu percepciju frekvencije zvuka te se u praksi pojam visine tona i pojam frekvencije često koriste sinonimno (kao što i u ovom radu).[2] Zvukovi s višim frekvencijama percipiraju se kao viši tonovi, dok se zvukovi s nižim frekvencijama percipiraju kao niži tonovi. Na primjer, nota C_4 ima frekvenciju od približno 261,63 Hz i percipira se kao viši ton u odnosu na notu C_3, koja ima frekvenciju od približno 130,81 Hz.

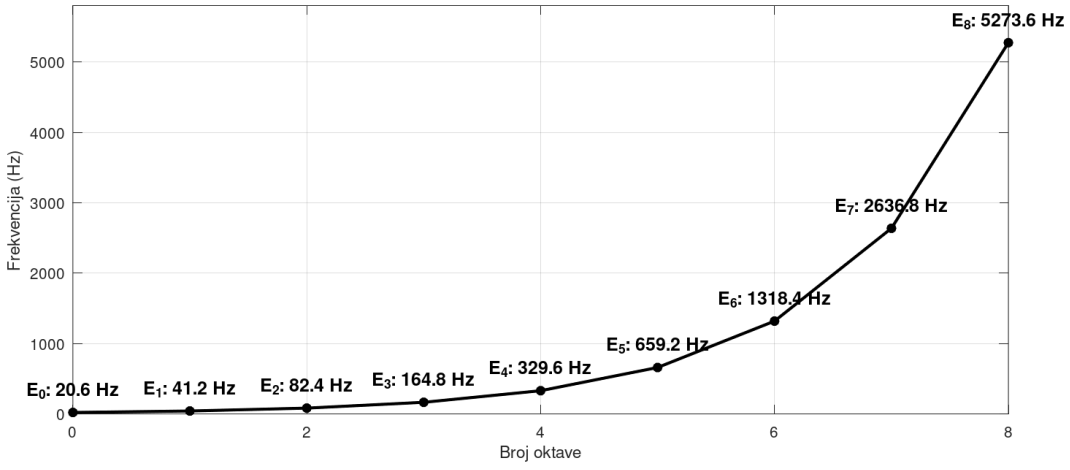
Kako bi uskladili instrumente, glazbenici štimaju instrumente prema referentnoj vi-

sini tona, tzv. koncertnoj visini tona (eng. *concert pitch*). Koncertna visina tona može varirati od ansambla do ansambla, a često se mijenjala kroz povijest. Međunarodna organizacija za standardizaciju (eng. *International Organization for Standardization* - ISO) 1975. godine uvela je standard za štimanje koji koristi 440 Hz za notu A_4, prema kojoj se u odnosu na nju postavljaju ostale note.[3] Iako je 440 Hz najčešće korišten standard, koriste se i drugačije visine tona ovisno o repertoaru.

2.2. Frekvencija u glazbi

U glazbi ljestvica predstavlja bilo koji skup glazbenih nota poredanih prema referentnoj frekvenciji ili visini tona (po standardu nota A_4 s frekvencijom od 440 Hz). Ljestvica u zapadnjačkoj koncertnoj glazbi temelji se na oktavi, što je razlika između dva zvuka čije su frekvencije u omjeru 2:1. Dva tona odsvirana istovremeno s razlikom od jedne oktave zvuče ugodno zajedno, zbog čega oktava ima široku primjenu u glazbi. Većina ljudi može razlikovati dva istovremena zvuka s razlikom u frekvenciji od 7 Hz, dok neki mogu prepoznati razliku čak i do 2 Hz.

Termin intervala koristi se za označavanje razlike u visini tona između dva zvuka.[4] Spomenuto je da su frekvencije dviju nota koje se razlikuju za oktavu u omjeru 2:1. To znači da se uzastopnim povećanjem visine tona za isti interval frekvencija eksponencijalno povećava, iako ljudsko uho to percipira kao linearno povećanje visine tona.



Slika 2.1: Frekvencije note E od E_0 do E_8

Za potrebe preciznog mjerenja razlike u visini tona, intervali se često mjere u centima (eng. *cent*).[5] Cent je jedinica dobivena dijeljenjem polutona na 100 jednakih

dijelova, odnosno dijeljenjem oktave na 1200 jednakih dijelova. Formulu za cent moguće je izraziti logaritmom omjera frekvencija:

$$n = 1200 \cdot \log_2(\frac{f_1}{f_2}) \tag{2.1}$$

pri čemu je n broj izražen u centima (cent), a f_1 i f_2 frekvencije izražene u hercima (Hz).

3. Glazbene note

U glazbi se termin nota koristi za označavanje određene visine tona (i trajanja u notnom zapisu). Nota također predstavlja skup visina tonova koji pripadaju istom razredu (eng. *pitch class*), odnosno visine tonova koji su međusobno odvojeni oktavom. U engleskom govornom području note se obično predstavljaju pomoću prvih sedam slova latinske abecede (A, B, C, D, E, F i G). Osma nota, nazvana oktavom, označava se istim slovom kao i prva, ali ima dvostruko višu frekvenciju.

Kako bi se razlikovale dvije note istog razreda, ali različitih oktava, sustav znanstvenog zapisa visine tona (eng. *scientific pitch notation*) kombinira naziv note s brojem koji označava određenu oktavu. Primjerice, E_2 s frekvencijom od 82,41 Hz je za jednu oktavu viša od E_1 s frekvencijom od 41,20 Hz, a za dvije oktave viša od E_0 s frekvencijom od 20,60 Hz.

Uz broj, nazivu note može se dodati predznak akcidental (eng. *accidental*).[6] Pod akcidentale spadaju povisilica (eng. *sharp* - ♯), koja podiže notu na frekvenciju koja je $\sqrt[12]{2}$ puta veća od trenutne, i snizilica (eng. *flat* - ♭) koja spušta notu na frekvenciju $\sqrt[12]{2}$ puta manju od trenutne. Drugim riječima, povisilica (♯) podiže notu za jedan poluton, dok snizilica (♭) spušta notu za jedan poluton.[7] Na primjer, $F_4^\sharp$ s frekvencijom od 369,99 Hz je za jedan poluton viši od F_4 s frekvencijom od 349,23 Hz, ali također za jedan poluton niži od G_4 s frekvencijom od 392 Hz. Stoga se nota $F_4^\sharp$ može zapisati i kao $G_4^\flat$.

Osim povisilice (♯) i snizilice (♭), pod akcidentale se ubraja i razrešnica (eng. *natural* - ♮), koja se koristi za prirodne note čija visina tona ostaje nepromijenjena, odnosno nije ni povišena ni snižena za poluton.[8] Ovaj akcidental rijetko se koristi jer se bez dodatka podrazumijeva da je nota prirodna.

3.1. Kromatska ljestvica

Kromatska ljestvica (eng. *chromatic scale* ili *twelve-tone scale*) bazira se na podjeli oktave na 12 proporcionalnih intervala, takozvanih polutona (eng. *semitone*), pri čemu

svi intervali imaju jednak omjer frekvencija od $\sqrt[12]{2}$.[9] Konstruirati ljestvicu može se dodavanjem polutona na bilo koju početnu notu. Matematički, frekvencija f_n n-te note u kromatskoj ljestvici može se izračunati kao:

$$f_n = f_0 \cdot (\sqrt[12]{2})^n \tag{3.1}$$

gdje je f_0 frekvencija početne note, a n broj polutona od početne note. Uz navedene informacije može se prikazati kromatska ljestvica konstruirana po noti C_3, zajedno s pripadnim frekvencijama (Hz):

C_3	$C_3^\sharp$	D_3	$D_3^\sharp$	E_3	F_3	$F_3^\sharp$	G_3	$G_3^\sharp$	A_3	$A_3^\sharp$	B_3
130,8	138,6	146,8	155,6	164,8	174,6	185,0	196,0	207,7	220,0	233,1	246,9

Vrijedi napomenuti da note E i B obično slijede note F i C, umjesto njihovih povišenih nota zapisanih kao $E^\sharp$ i $B^\sharp$. Dijatonička ljestvica, bazirana na podjeli oktave na 7 intervala, upotpunjena je kao kromatska ljestvica s 5 novih nota. Kako bi zapisi nota predstavljali jednake visine tona u obje ljestvice, izbjeglo se preimenovanje već utvrđenih nota.[10]

Kromatska ljestvica omogućuje sviranje svih mogućih nota unutar jedne oktave, čineći je osnovom za mnoge melodijske i harmonijske strukture. Zbog svoje intuitivnosti i fleksibilnosti ima značajnu ulogu u glazbi.

4. Frekvencije na primjeru gitare

Postoje raznovrsni tipovi gitara sa šest žica: klasična, akustična, električna i hibridna. Svaka od njih može imati različite karakteristike koje im daju specifičan zvuk. Bitan utjecaj na frekvencijski raspon gitare čine duljina vrata, odnosno broj pragova (eng. *fret*) na vratu, i napetost žica, odnosno štimanje.

4.1. Pragovi

Vrat gitare precizno je podijeljen manjim metalnim trakama na pragove. Pritisak žice na prag ima isti učinak kao skraćivanje žice, a što je žica kraća, to je odsvirani ton viši. Pritiskom pragova niže na vratu (prema tijelu gitare), odsvirane žice titrat će višom frekvencijom, tj. svirat će više note. Time se note učinkovito raspodjeljuju po pragovima na vratu gitare, uključujući one note odsvirane bez pritiska na žice. Klasične i akustične gitare obično imaju vrat podijeljen na 18 do 21 prag, dok električne obično imaju između 21 i 24 praga, a ponekad s iznimkama do 27 pragova.

4.2. Štimanje

Osim broja pragova, na frekvencijski raspon gitare utječe njeno štimanje koje svaka gitara može mijenjati. Štimanje (eng. *tuning*) gitare odnosi se na specifičan raspored nota koje žice sviraju prilikom nepritisnutog titranja, na što utječe napetost žica. Žice se mogu otpustiti ili zategnuti pomoću vijaka na glavi gitare što se, za razliku od pritisaka na pragove, izvodi prije sviranja.

Standardno naštimana (eng. *standard tuning*) gitara podrazumijeva da su žice (od gornje prema donjoj) naštimane prema notama:

$$E_2 \quad A_2 \quad D_3 \quad G_3 \quad B_3 \quad E_4$$

Dodatan primjer štimanja je *Open D*, koje predstavlja raspored nota:

$$D_2 \quad A_2 \quad D_3 \quad F_3^\sharp \quad A_3 \quad D_4$$

4.3. Raspon nota i frekvencija

Uobičajeni frekvencijski raspon gitare može se predstaviti primjerom standardno naštimane gitare s 21 pragom. Frekvencijski raspon gitare s 21 pragom proteže se od note E_2 s frekvencijom od 82,41 Hz do note $C_6^\sharp$ s frekvencijom od 1108,73 Hz. Važno je napomenuti da je najniža frekvencija postignuta sviranjem "otvorene" (nepritisnute) gornje E_2 žice, dok je najviša frekvencija postignuta sviranjem donje E_4 žice, pritisnute na 21. pragu vrata gitare. Sljedeća tablica prikazuje raspon nota takve gitare i njihovih frekvencija:

Tablica 4.1: Raspon nota i frekvencija gitare s 24 praga.

Nota / Frekvencija (Hz)	2	3	4	5	6
C		130,8	261,6	523,3	1047
$C^\sharp$		138,6	277,2	554,4	1109
D		146,8	293,7	587,3	(1175)
$D^\sharp$		155,6	311,1	622,3	(1245)
E	82,41	164,8	329,6	659,3	(1318)
F	87,31	174,6	349,2	698,5	
$F^\sharp$	92,50	185,0	370,0	740,0	
G	98,00	196,0	392,0	784,0	
$G^\sharp$	103,8	207,7	415,3	830,6	
A	110,0	220,0	440,0	880,0	
$A^\sharp$	116,5	233,1	466,2	932,3	
B	123,5	246,9	493,9	987,8	

5. Harmonici

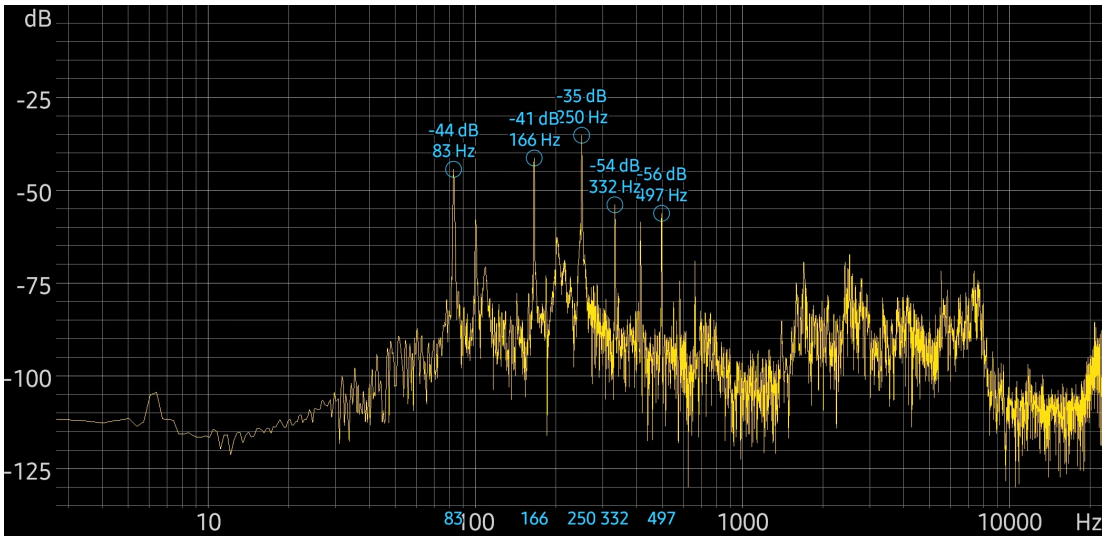
Većina zvukova sastoji se od mješavine različitih frekvencija. Kada žica gitare titra, ona proizvodi temeljnu frekvenciju i njene višekratnike koji se nazivaju harmonici. Pregibi ili harmonici (eng. *harmonics*) nastaju kod zvukova s jednom dominantnom, odnosno temeljnom frekvencijom (eng. *fundamental frequency*), kao višekratne frekvencije te temeljne frekvencije.[11] Temeljna frekvencija najniža je i obično najintenzivnija frekvencija u zvuku.[12] Na primjer, ako je F temeljna frekvencija, harmonici zvuka imat će frekvencije $2F$, $3F$, $4F$, itd. Suma intenziteta frekvencija zvuka sačinjenog od harmonika bi se mogla onda prikazati na sljedeći način:

$$a \cdot F + b \cdot 2F + c \cdot 3F + ... \tag{5.1}$$

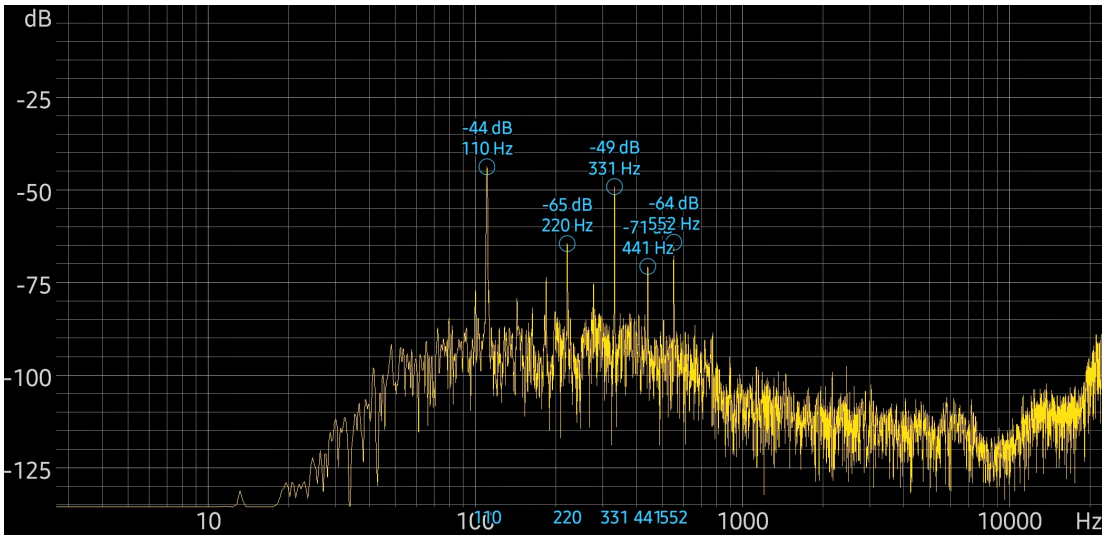
pri čemu su a, b, c koeficijenti intenziteta. Koeficijent a obično je najveći jer je F temeljna frekvencija. Razlike u koeficijentima intenziteta dolaze od različitih izvora zvuka. Različiti instrumenti proizvode različite intenzitete harmonika, što doprinosi njihovoj jedinstvenoj boji zvuka. Osim intenziteta harmonika, razlike u zvuku između dvaju različitih izvora stvaraju i slabije popratne frekvencije, tzv. alikvotni tonovi (eng. *overtones*) koji tonu daju dodatnu boju.[13] Iz ovih razloga, nota odsvirana na dva različita instrumenta zvučat će drugačije, iako ima istu temeljnu frekvenciju, odnosno visinu tona.

5.1. Frekvencijski spektri

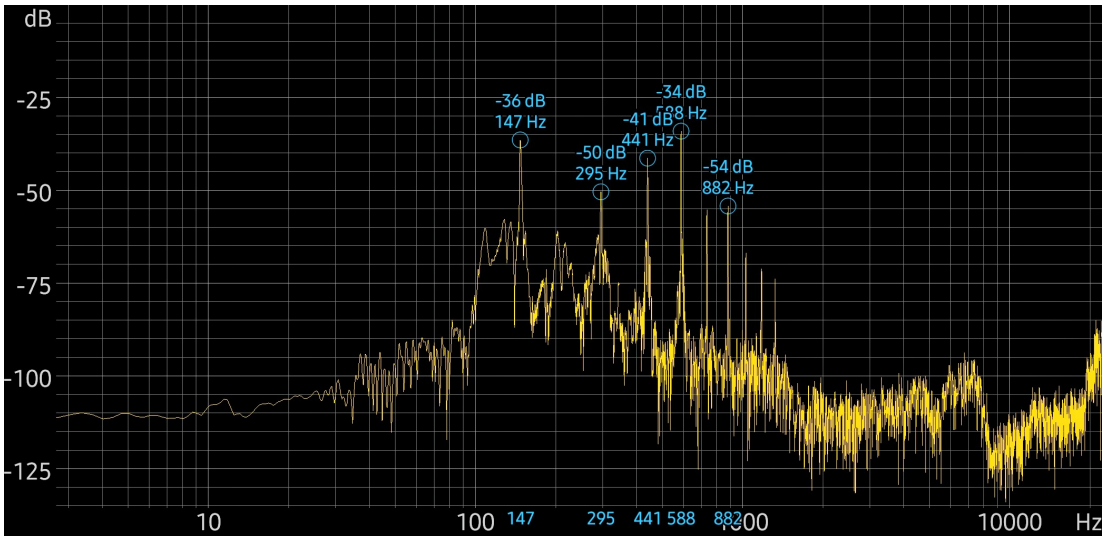
Harmonici se lako mogu istaknuti prikazima frekvencijskih spektara snimljenih zvukova. Slike 5.1 do 5.6 predstavljaju spektrograme frekvencija žica gitare, snimljene pomoću aplikacije *Spectroid*[14] . Lako je uočiti kako se među frekvencijama ističe temeljna frekvencija i njezini harmonici.



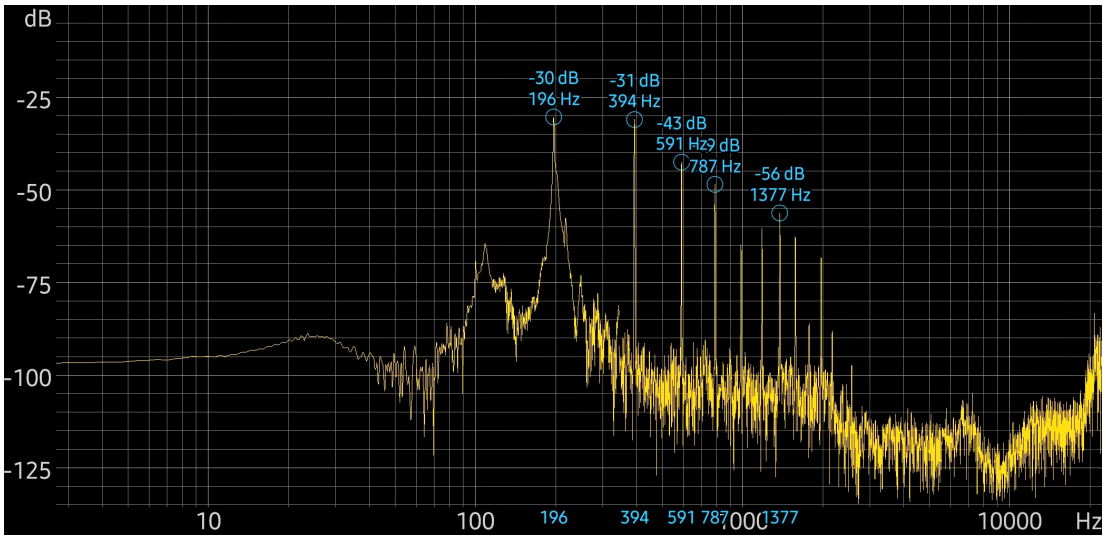
Slika 5.1: Frekvencijski spektar žice E_2



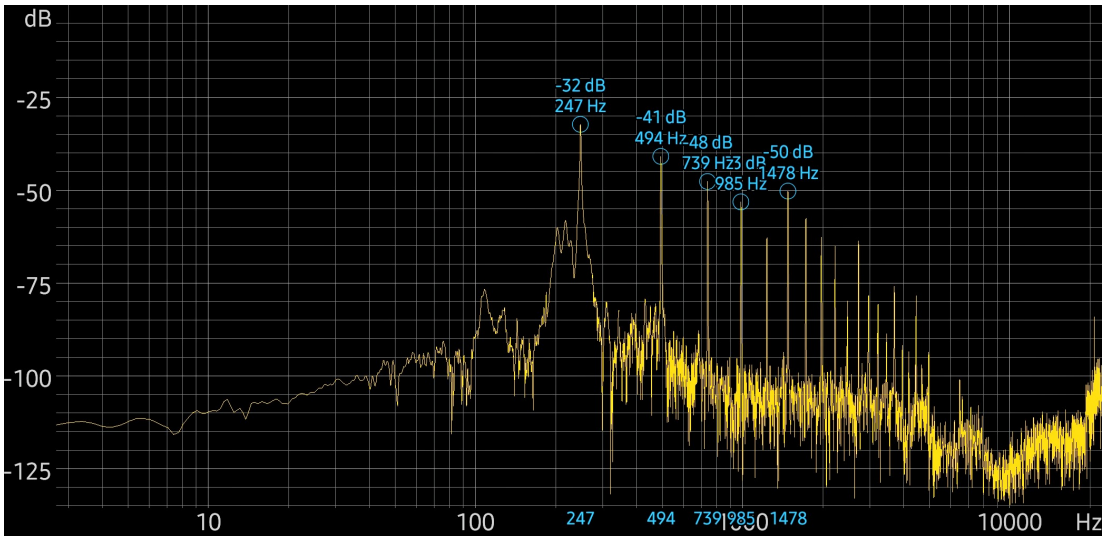
Slika 5.2: Frekvencijski spektar žice A_2



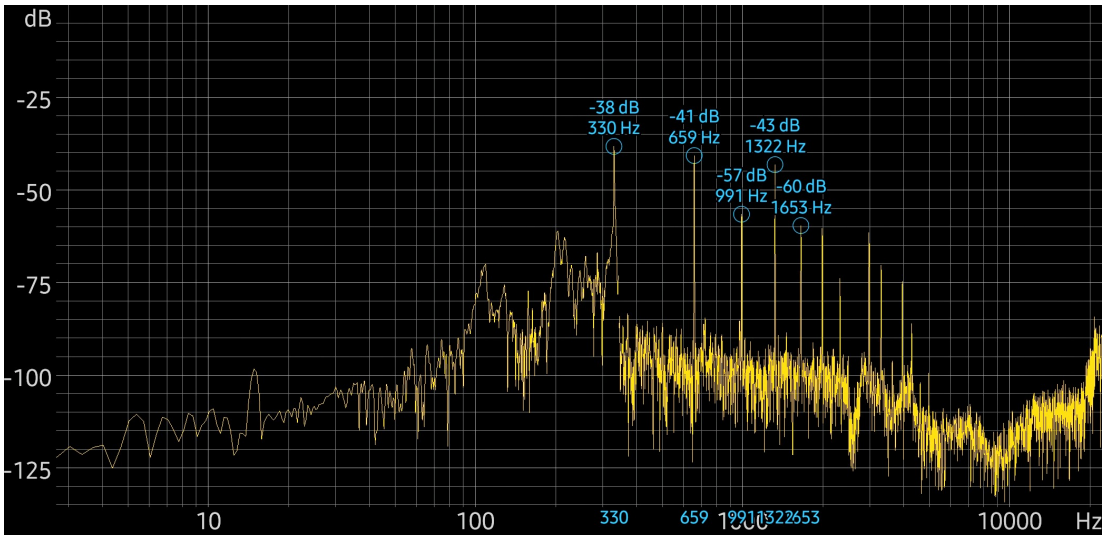
Slika 5.3: Frekvencijski spektar žice D_3



Slika 5.4: Frekvencijski spektar žice G_3



Slika 5.5: Frekvencijski spektar žice B_3



Slika 5.6: Frekvencijski spektar žice E_4

Zbog neidealnih uvjeta pri mjerenju u sklopu ovoga rada, dolazi do manjih odstupanja u vrijednostima višekratnika, ali je pojava harmonika svejedno jasno vidljiva.

Primjerice, slika 5.5 vrlo jasno prikazuje ovaj fenomen, s istaknutom dominantnom frekvencijom od 247 Hz, što je upravo temeljna frekvencija note B_3. Temeljnu frekvenciju redom slijede istaknute frekvencije od 494 Hz, 739 Hz, 985 Hz te 1478 Hz, koji su približni višekratnici temeljne frekvencije. Treba napomenuti da intenzitet harmonika nije uvijek u opadajućem poretku, što se vidi kod harmonika od 1478 Hz koji je jačeg intenziteta od prethodna dva harmonika.

Drugačiji primjer pokazuje slika 5.1. Iako je frekvencija od 83 Hz (što odgovara noti E_2) istaknuta, ona nije najistaknutija frekvencija. Ovo je čest problem pri detekciji frekvencija u zvučnim signalima. Ovisno o snimljenim podacima, temeljna frekvencija može, ali ne mora biti najistaknutija.[15] Više o ovom problemu razmatra se u poglavlju 7.2.

Vrijedi napomenuti da intenzitet temeljne frekvencije i harmonika kroz vrijeme nije konstantan. Česta pojava početkom sviranja note je skok te zatim lagani pad u intenzitetu temeljne frekvencije uz lagani porast intenziteta harmonika. Ovakva pojava dodatno otežava konzistentne rezultate u detekciji temeljne frekvencije.

6. Digitalna obrada signala

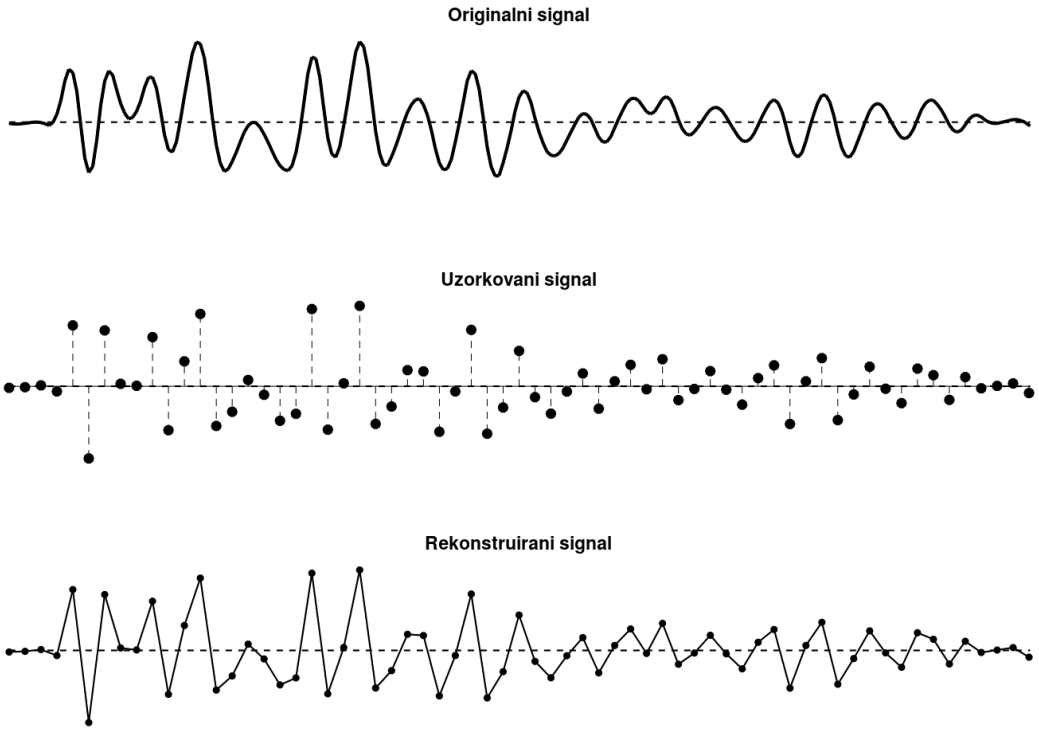
Digitalna obrada signala (eng. *Digital Signal Processing - DSP*) koristi procesore koji preuzimaju snimljene signale iz stvarnog svijeta poput zvuka, govora, video prikaza, temperature, tlaka, radara i drugih signala koji su digitalizirani te nad njima izvode matematičke operacije kako bi ih oblikovali.

Signale je potrebno obraditi kako bi se informacija koju sadrže mogla bolje prikazati, analizirati ili pretvoriti u drugu vrstu signala koja bi mogla biti korisna. Da bi se analogni signal digitalizirao, potreban je analogno-digitalni pretvarač (eng. *Analog-to-digital converter - ADC*) koji će snimljeni analogni signal pretvoriti u digitalni binarni zapis.

Primjena digitalne obrade signala donosi mnoge prednosti u odnosu na analognu obradu, kao što su kompresija podataka, filtriranje i uklanjanje šuma. Pruža veću fleksibilnost za različite primjene te omogućuje otkrivanje i ispravljanje pogrešaka pri prijenosu podataka.[16]

6.1. Uzorkovanje

U kontekstu obrade zvučnog signala, uzorkovanje (eng. *sampling*) je proces pretvorbe zvučnog vala u slijed "uzoraka". Cilj uzorkovanja je kroz vrijeme zabilježiti uzorke ekvivalentne njihovim trenutnim vrijednostima u signalu.



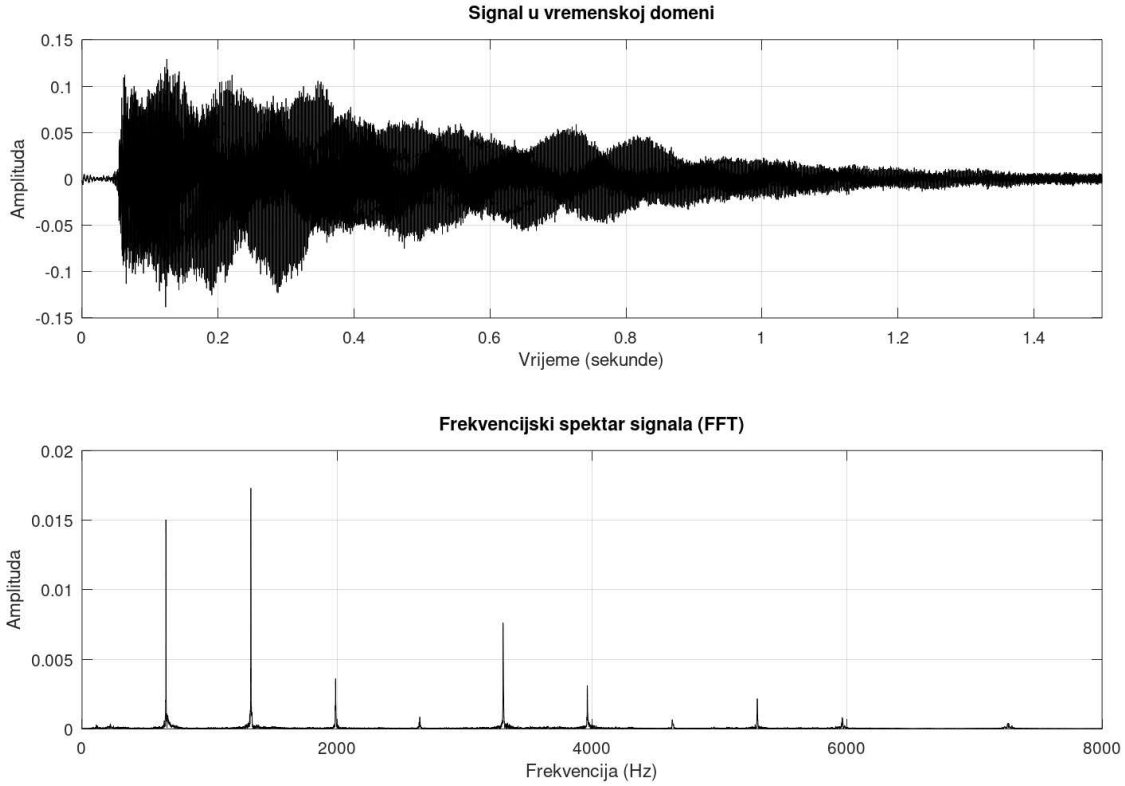
Slika 6.1: Uzorkovanje signala kroz vrijeme

Uzorak (eng. *sample*) uglavnom predstavlja amplitudu signala u određenom trenutku, a frekvencija uzorkovanja (eng. *sample rate* ili *sampling frequency*) predstavlja prosječni broj uzoraka uzetih u jednoj sekundi te se mjeri u hercima (Hz).

Prema Nyquist-Shannonovom teoremu, frekvencija uzorkovanja mora biti najmanje dvostruko veća od najveće frekvencije prisutne u signalu kako bi se signal mogao pravilno rekonstruirati.[17] Budući da se zvučni signali snimaju do gornje granice ljudskog sluha od 20 kHz, obrađuju se frekvencijom uzorkovanja većom od 40 kHz. Najčešće frekvencije uzorkovanja su 44,1 kHz, 48 kHz, 88,2 kHz ili 96 kHz.[18] Frekvencije uzorkovanja iznad 50 kHz rijetko se koriste jer ne doprinose kvaliteti zvuka za ljudsko uho, a zahtijevaju veću širinu pojasa (eng. *bandwidth*) te više resursa za obradu i pohranu podataka.

7. Algoritam brze Fourierove transformacije - FFT

Aplikacija *Spectroid* za detekciju frekvencija koristi algoritam brze Fourierove transformacije (eng. *Fast Fourier Transform - FFT*).[14] Algoritam FFT pretvara signal u skup brojeva, omogućavajući prepoznavanje najdominantnijih frekvencija unutar signala. Njegova sposobnost da razdvoji zvučni signal na njegove sastavne frekvencije čini ga vrlo korisnom metodom u praksi obrade zvučnih signala.[15]



Slika 7.1: Učinak FFT-a na zvučni signal odsvirane žice

7.1. Način rada FFT-a

Kada se signal obrađuje pomoću FFT-a, izlaz je niz (najčešće niskih decimalnih) brojeva koji predstavlja frekvencijsku domenu obrađenog signala, primjerice:

```
fft_output_array = [0.001, 0.003, 0.007, 0.015, 0.005...]
```

Svaki element ovog niza naziva se *bin* i predstavlja određeni raspon frekvencija. Vrijednost svakog elementa označava intenzitet frekvencija unutar tog raspona. Veća vrijednost označava veći intenzitet ili zastupljenost tih frekvencija. Raspon frekvencija za svaki *bin* ovisi o rezoluciji (eng. *resolution*) koja je bitna za sposobnost razlikovanja dviju bliskih frekvencija. Rezolucija se može izračunati na sljedeći način:

```
resolution = sampling_rate / fft_size
```

gdje je `fft_size` veličina spremnika za uzorke (eng. *buffer size*).[19] Veličina *buffer*-a povezana je s vremenom potrebnim za obradu signala: manji *buffer*-i stvaraju manje kašnjenje, ali povećavaju opterećenje procesora i mogućnost grešaka.[20]

Za prepoznavanje frekvencija po *bin*-ovima koristi se sljedeća formula:

```
start_frequency = bin_number * resolution
```

gdje je `start_frequency` početna frekvencija raspona.[15] Raspon frekvencija tako ide od `start_frequency` do `start_frequency + resolution`.

Primjerice, s frekvencijom uzorkovanja od 48 kHz i veličinom *buffer*-a od 1024 uzoraka dobiva se rezolucija od 46,875 Hz po uzorku; što znači da svaki *bin* predstavlja raspon frekvencija dugačak 46,875 Hz. Pomoću prethodne formule možemo izračunati početnu frekvenciju raspona za 12. *bin*, što je 562,5 Hz. Dakle, raspon 12. *bin*-a je od 562,5 Hz do 609,375 Hz.

Rezolucija FFT-a igra ključnu ulogu u preciznosti detekcije frekvencija. Da bi algoritam mogao prepoznati dvije različite note, rezolucija mora biti manja ili jednaka razlici njihovih frekvencija. U navedenom primjeru, rezolucija od 46,875 Hz nije dovoljna za razlikovanje najnižih frekvencija akustične gitare (E_2 na 82,41 Hz i F_2 na 87,31 Hz), gdje je razlika u frekvencijama samo 4,9 Hz. Dakle, potrebno je smanjiti rezoluciju na 4,9 Hz po uzorku ili manje, što se može postići smanjenjem frekvencije uzorkovanja ili povećanjem *buffer*-a, ali uz cijenu dodatnog opterećenja procesora i dužeg vremena za punjenje *buffer*-a, što u krajnju ruku smanjuje koliko su rezultati u stvarnom vremenu.

7.2. Problemi FFT-a

U poglavlju 5 naveden je problem vezan uz detekciju temeljne frekvencije signala. Primjerice, na slici 5.1 temeljna frekvencija note E_2 iznosi 82,41 Hz, ali ta frekvencija nije najsnažnija. Zbog toga se ne može osloniti na prepoznavanje temeljne frekvencije jednostavnim traženjem frekvencije s najvećim intenzitetom. Ova pojava može izazvati probleme u sustavima koji trebaju točno prepoznati visinu tona odsvirane note.

Iako je FFT moćan alat za analizu frekvencija, njegova ograničenja u prepoznavanju temeljne frekvencije zahtijevaju upotrebu dodatnih ili alternativnih algoritama kako bi se postigla točnija detekcija glazbenih nota. U ovom radu, zbog tih ograničenja, koriste se koncepti kompleksnijeg algoritma YIN za prepoznavanje visine tona.

8. Algoritam YIN

YIN algoritam razvili su Alain de Cheveigné i Hideki Kawahara 2002. godine kao unaprjeđenje ranijih metoda za detekciju visine tona. Razvijen je kako bi nadomjestio nedostatke klasičnih metoda poput FFT-a, posebno u kontekstu glazbenih nota, gdje je potrebna visoka točnost i niska stopa pogrešaka. Učestalost grešaka bila je otprilike tri puta manja u usporedbi s najboljim konkurentskim metodama tih vremena. Iako danas postoje precizniji i brži algoritmi, algoritam YIN prikladan je za visoke tonove i jednostavan za implementaciju s niskim kašnjenjem (eng. *latency*).[1]

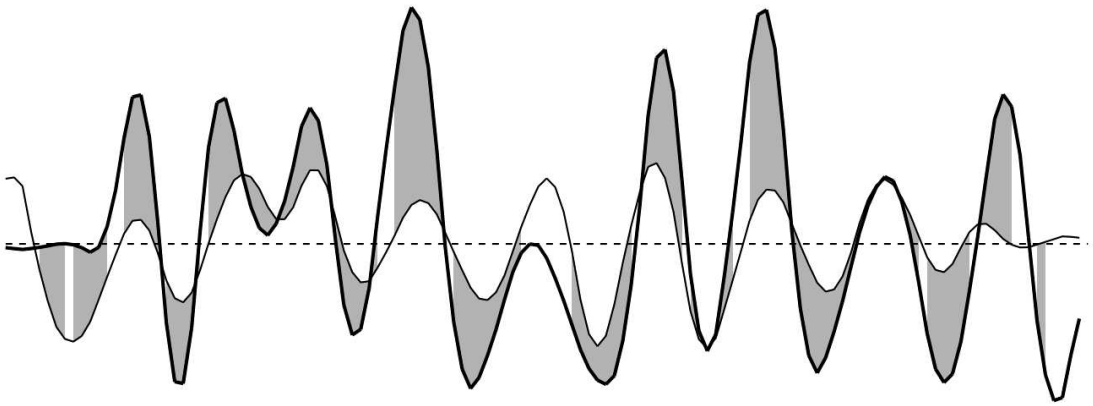
Naziv algoritma inspiriran je kineskom filozofijom *yin* i *yang*, održavajući ravnotežu između točnosti i računalne učinkovitosti.

8.1. Sličnost između dvaju signala

Kako bi se objasnio YIN algoritam, korisno je znati kako proučavanje sličnosti između dvaju signala pridonosi identificiranju temeljne frekvencije i kako se ta sličnost računa. Sličnost dvaju signala može se mjeriti na različite načine, no jedan od najjednostavnijih pristupa je suma proizvoda njihovih vrijednosti u svakoj točki. Sličnost dvaju signala preko cijelog intervala može se izraziti kao:

$$similarity = \sum_{t=0} f(t) \cdot g(t) \tag{8.1}$$

gdje f i g predstavljaju vrijednosti signala u određenom trenutku t. Kada su vrijednosti dvaju signala na istoj strani osi $f = 0$ (oba pozitivna ili oba negativna), proizvod će biti pozitivan, što ukazuje na visoku sličnost. Ako su vrijednosti na suprotnim stranama (jedan pozitivan, drugi negativan), proizvod će biti negativan, što ukazuje na nisku sličnost. Sumiranjem ovih proizvoda možemo dobiti ukupnu mjeru sličnosti koja uzima u obzir sve točke u signalu.



Slika 8.1: Sličnost dvaju signala provjerom zajedničkih polariteta

8.2. Funkcija autokorelacije

Poseban slučaj gdje uspoređujemo signal sa samim sobom, ali s vremenskim pomakom (kašnjenjem), naziva se autokorelacija (eng. *autocorrelation*). Funkcija autokorelacije (eng. *autocorrelation function* - ACF) mjeri koliko je signal sličan samome sebi nakon određenog kašnjenja τ:

$$\mathrm{ACF}(\tau) = \sum_{t=0} f(t) \cdot f(t+\tau) \tag{8.2}$$

Visoke vrijednosti ACF-a na određenom kašnjenju τ ukazuju na to da signal ima značajnu komponentu periodičnog uzorka s periodom τ. Drugim riječima, vrijednost ACF bit će najveća za kašnjenje τ koji pokazuje najveću sličnost između originalnog i zakašnjelog signala. Cilj metode autokorelacije bit će pronaći vrijednost kašnjenja τ, odnosno:

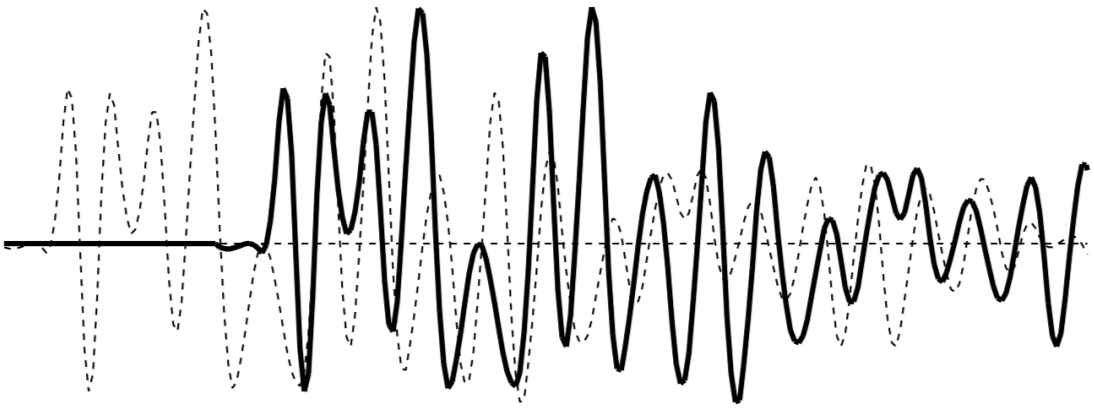
$$\begin{aligned} \hat{\tau} &= \underset{\tau}{\mathrm{argmax}}(\mathrm{ACF}(\tau)) \\ &= \underset{\tau}{\mathrm{argmax}}(\sum_{t=0} f(t) \cdot f(t+\tau)) \end{aligned} \tag{8.3}$$

Pronalazak $\hat{\tau}$ moguće je postići na različite načine. U implementaciji ovog rada koristit će se jednostavna iteracija mogućih vrijednosti τ te usporedba pri svakom koraku.

Pronalaskom $\hat{\tau}$, izračun temeljne frekvencije postaje trivijalan:

$$f_{fundemental} = \frac{f_{sample}}{\hat{\tau}} \tag{8.4}$$

gdje je temeljna frekvencija izražena $f_{fundemental}$, a f_{sample} predstavlja frekvenciju uzorkovanja koja se koristi za snimanje signala.



Slika 8.2: Autokorelacija signala s odmakom kašnjenja

Autokorelacija je korisna za identifikaciju ponavljajućih obrazaca u signalu, što je ključno za detekciju temeljne frekvencije. Za perfektno periodnične signale, ACF će davati točne rezultate u detekciji temeljne frekvencije. Unatoč svojoj korisnosti, funkcija ACF je sama po sebi nedovoljna za preciznu detekciju temeljne frekvencije u složenim zvučnim signalima.

Problem metode autokorelacije je što ne uzima u obzir odstupanja između vrijednosti originalnog i zakašnjelog signala, nego samo predznak tih vrijednosti. Implementacija funkcije ACF kao takve je osjetljiva na promjene u amplitudi signala. Porast amplitude signala kroz vrijeme povećavat će i produkt vrijednosti signala ($f(t) \cdot f(t+\tau)$) te će algoritam naivno birati veća kašnjenja kao najbolje kašnjenje $\hat{\tau}$. Smanjenje amplituda signala imat će suprotan učinak. Ovo je problematično jer vodi do pogreške gdje algoritam bira više, odnosno niže frekvencije umjesto stvarne temeljne frekvencije.

Za potrebe ovoga rada, odnosno detekciju note u zvuku, potrebno je prilagoditi ovu metodu.

8.3. Funkcija razlike

Problem detekcije temeljne frekvencije može se sagledati iz drugačijeg kuta, krenuvši par koraka unazad. Perfektno periodičan signal f s periodom T, u potpunosti će se preklapati sa zakašnjelom kopijom samog sebe ukoliko je to kašnjenje od T. Ovo se može zapisati kao:

$$f(t) - f(t+T) = 0,\ \forall t \tag{8.5}$$

gdje je t vrijeme u određenom trenutku. U kontekstu ovog problema, period T može se sagledati kao prethodno spomenuto kašnjenje τ.

Ovoj jednadžbi mogu se dodati dvije izmjene. Prva izmjena je da se koristi vremenski prozor (eng. *time window*), odnosno raspon od prve do zadnje vrijednosti kašnjenja koja će se analizirati. Umjesto analize preklapanja preko cijelog signala, analizira se manji isječak signala. Uvođenjem prozora postiže se veća brzina i mogućnost analize promjene frekvencije tijekom vremena. Veća veličina prozora zahtjevat će u implementaciji veću računalnu snagu što uzrokuje veće kašnjenje (eng. *latency*) rezultata algoritma. Dodatkom veličine vremenskog prozora W dobivamo jednadžbu:

$$\sum_{t}^{t+W} f(t) - f(t+\tau) = 0 \tag{8.6}$$

Cilj je pronaći vrijednost kašnjenja τ koji zadovoljava jednadžbu. Naravno, ova jednadžba neće nikad biti zadovoljena ako signal nije perfektno periodičan. Budući da će svaka devijacija u vrijednostima signala biti zabilježena u sumi, sličnost će biti veća što je ta suma manja. Stoga je potrebno pronaći τ za koji će ova suma biti najbliža nuli.

Bitno je primjetiti da postoji mogućnost poništavanja pozitivnih i negativnih devijacija. Kako bi se takva pojava izbjegla, kvadrira se razlika u sumi:

$$\sum_{t}^{t+W} (f(t) - f(t+\tau))^2 = 0 \tag{8.7}$$

Ovime se dobiva nova metoda za detekciju temeljne frekvencije, zapisana kao funkcija razlike (eng. *difference function* - DF):

$$\mathrm{DF}(\tau) = \sum_{t}^{t+W} (f(t) - f(t+\tau))^2 \tag{8.8}$$

Dubljim razmatranjem ove funkcije može se izvesti poveznica s funkcijom ACF. Raspisivanjem i substitucijom, funkciju razlike moguće je izraziti pomoću funkcije ACF:

$$\mathrm{ACF}(\tau, t) = \sum_{t=0}^{t+W} f(t) \cdot f(t+\tau) \tag{8.9}$$

$$\begin{aligned} \mathrm{DF}(\tau) &= \sum_{t}^{t+W} (f(t) - f(t+\tau))^2 \\ &= \sum_{t}^{t+W} f(t)^2 + f(t+\tau)^2 - 2 \cdot f(t) \cdot f(t+\tau) \\ &= \mathrm{ACF}(0, t) + \mathrm{ACF}(0, t+\tau) - 2 \cdot \mathrm{ACF}(\tau, t) \end{aligned} \tag{8.10}$$

Ovaj dodatan korak dodatno pojednostavljuje problem te omogućuje implementaciju pomoću ACF. Usporedno s ACF koji za niže vrijednosti pokazuje manju sličnost, DF za niže vrijednosti pokazuje veću sličnost, odnosno bolje preklapanje originalnog i zakašnjelog signala. Drugim riječima, kašnjenje $\hat{\tau}$ tražit će se kao:

$$\begin{aligned} \hat{\tau} &= \underset{\tau}{\operatorname{argmin}}(\mathrm{DF}(\tau)) \\ &= \underset{\tau}{\operatorname{argmin}}(\mathrm{ACF}(0, t) + \mathrm{ACF}(0, t+\tau) - 2 \cdot \mathrm{ACF}(\tau, t)) \end{aligned} \tag{8.11}$$

Funkcija DF manje griješi pri problemu porasta amplitude nego funkcija ACF, budući da će većim odstupanjem vrijednosti signala ukazivati na manju sličnost. Unatoč tome, i ova metoda je osjetljiva na promjene amplituda signala. Ako signal sadrži šum ili lokalne anomalije poput velikih skokova ili padova, ove će promjene značajno povećati vrijednosti DF-a, što može dovesti do netočnih detekcija perioda. Srećom, ovu metodu lako je dodatno prilagoditi.

8.4. Funkcija kumulativne srednje normalizirane razlike

Kao unaprjeđenje na funkciju DF, uvodi se funkcija kumulativne srednje normalizirane razlike (eng. *Cumulative Mean Normalized Difference Function* - CMNDF). Kako bi omogućila stabilniju detekciju temeljne frekvencije u složenim signalima, CMNDF normalizira vrijednosti DF-a:

$$\mathrm{CMNDF}(\tau) = \frac{\mathrm{DF}(\tau)}{\frac{1}{\tau}\sum_{j=1}^{\tau}\mathrm{DF}(j)} = \tau \cdot \frac{\mathrm{DF}(\tau)}{\sum_{j=1}^{\tau}\mathrm{DF}(j)} \tag{8.12}$$

Za svaki korak kašnjenja (od $j = 1$ do τ) računa se kumulativna suma vrijednosti $\mathrm{DF}(j)$ kako bi se mogao izračunati prosjek vrijednosti funkcija DF do trenutnog kašnjenja τ. Funkcija DF zatim se dijeli tim prosjekom kako bi se normalizirala. Ovakav postupak ponavlja se za svaku moguću vrijednost kašnjenja τ te cilj ostaje pronaći τ koji će dati najmanju vrijednost funkcije CMNDF.

Za slučaj bez kašnjenja ($\tau = 0$), CMNDF će primiti vrijednost 1. Budući da ovaj slučaj ne pruža nikakve korisne informacije, njega se ne analizira. CMNDF će pasti ispod vrijednosti 1 kada je DF manja od prosjeka do tada analiziranih kašnjenja.

$$\mathrm{CMNDF}(\tau) = \begin{cases} 1 & \tau = 0 \\ \tau \cdot \frac{\mathrm{DF}(\tau)}{\sum_{j=1}^{\tau}\mathrm{DF}(j)} & \tau > 0 \end{cases} \tag{8.13}$$

Budući da su rezultati funkcije DF normalizirani, velika odstupanja u amplitudama bit će razmotrena s obzirom na ostatak analiziranog prozora. Posljedično, šum i modulacija amplituda imaju manji utjecaj na preciznost algoritma.

8.5. Prag pretrage

Posljednji dodatak CMNDF-u je uvođenje praga (eng. *threshold*). Uvođenje praga tijekom pretrage pomaže u izbjegavanju grešaka uzrokovanih harmonicima, odnosno omogućava razlikovanje temeljne frekvencije od njezinih višekratnika koji također mogu imati niske vrijednosti CMNDF-a. Metoda CMNDF-a s pragom pretrage koristi se na način da se prvo izračunaju sve vrijednosti funkcije CMNDF-a, a zatim da se traži najmanja vrijednost kašnjenja τ za koju je vrijednost funkcije CMNDF manja od postavljenog praga. Ako ni jedno takvo kašnjenje nije pronađeno, uzima se ono koje daje najmanji rezultat CMNDF, tj. globalni minimum.

$$\hat{\tau} = \begin{cases} \min \tau \mid \mathrm{CMNDF}(\tau) < \mathrm{threshold} & \exists\tau\ \mathrm{CMNDF}(\tau) < \mathrm{threshold} \\ \underset{\tau}{\mathrm{argmin}}(\mathrm{CMNDF}(\tau)) & \forall\tau\ \mathrm{CMNDF}(\tau) >= \mathrm{threshold} \end{cases} \tag{8.14}$$

8.6. Dodatni koraci

YIN algoritam uključuje dva dodatna koraka za precizniju detekciju temeljne frekvencije: parabolična interpolacija i najbolja lokalna procjena.

8.6.1. Paraboličnа interpolacija

Paraboličnа interpolacija (eng. *parabolic interpolation*) korisna je u slučajevima kada se zbog nedovoljno visoke frekvencije uzorkovanja ne može precizno izabrati temeljna frekvencija. Ovakva pojava događa se kada se period temeljne frekvencije nalazi između dva uzorka. Interpolacija se zatim koristi jer je računalno jeftinija nego uzorkovanje signala na višoj frekvenciji.

Korak paraboličnе interpolacije nije potreban za svrhe ovog rada, budući da pridodaje računalnoj kompleksnosti algoritma koja pridodaje kašnjenju rezultata u stvarnom vremenu. S druge strane, paraboličnа interpolacija ne pridodaje nužnu količinu preciznosti da bi znatno unaprijedila funkcionalnost prethodnih koraka.

8.6.2. Najbolja lokalna procjena

Metoda najbolje lokalne procjene (eng. *best local estimate*) zadnji je korak u YIN algoritmu koji se koristi kako bi se smanjile pogreške nastale zbog fluktuacija u procjenama. Za svaki trenutak analize t, traži najmanju vrijednost CMNDF u malom okruženju oko tog trenutka, pretraživanjem užeg intervala $[t - \frac{T_{\text{max}}}{2}, t + \frac{T_{\text{max}}}{2}]$, gdje je T_{max} najveći očekivani period. Kao i u prethodnom koraku, u tom intervalu bira se kašnjenje $\hat{\tau}$, za koje je CMNDF najmanji.

Budući da i ovaj korak pridodaje računalnoj složenosti, a pruža zanemarivo veću preciznost, u implementaciji rada ga se izostavlja.

9. Aplikacija

U sklopu ovog rada razvijena je aplikacija za prepoznavanje glazbenih nota, prvenstveno onih u rasponu gitare.

Aplikacija primjenjuje koncepte YIN algoritma te prikazuje visinu tona snimljenog zvuka te najbliže udaljenu notu na kromatskoj ljestvici. Pisana je u integriranom razvojnom okruženju (eng. *Integrated Development Environment* - *IDE*) Visual Studio-u u programskom jeziku C++ te je namijenjena isključivo za Windows operativni sustav.

9.1. Snimanje zvuka

Za snimanje zvuka koristi se pomoć PortAudio[21] biblioteke koja omogućuje pisanje jednostavnih programa za obradu zvuka u C++-u.

9.1.1. Funkcija povratnog poziva

Biblioteka pruža sučelje (eng. *Application Programming Interface* - API) za snimanje i/ili puštanje zvuka, omogućeno preko jednostavne funkcije povratnog poziva (eng. *callback function*):

```
//// FUNKCIJA POVRATNOG POZIVA PORTAUDIO-A:
int paCallback(const void* input_buffer, void* output_buffer,
    unsigned long frames_per_buffer,
    const PaStreamCallbackTimeInfo* time_info,
    PaStreamCallbackFlags status_flags,
    void* user_data) {

    return paContinue;
}
```

Funkcija `paCallback()` poziva se iznova za svaki snimljeni *buffer* audio signala. Učestalost pozivanja ovisit će o njezinim parametrima, specifično veličini *buffer*-a. Vraćanjem vrijednosti 0 (zapisana kao enumeracija `paContinue`), funkcija je

spremna primiti sljedeći *buffer* podataka.

Parametar `input_buffer` predstavlja ulazni *buffer* koji sprema amplitude snimljenog audio signala u decimalnom (*float*) formatu. Slično kao `input_buffer`, u `output_buffer` mogu se spremiti izlazni audio signali. Broj okvira (eng. *frames*), odnosno broj uzoraka po *buffer*-u, predstavljen je parametrom `frames_per_buffer`. Ovaj parametar određuje veličinu parametara `input_buffer` i `output_buffer`.

Dodatni parametri `time_info`, `status_flags` i `user_data` pružaju dodatne informacije i kontekst o strujanju zvuka (eng. *audio stream*) te nisu potrebni za implementaciju.

9.1.2. Uspostava strujanja zvuka

Kako bi se pokrenulo strujanje zvuka, potrebno ga je inicirati preko bibliotekine klase `PaStream` i njenih metoda. Inicijalizacijom PortAudio-a alociraju se resursi koji su biblioteci potrebni za obradu zvuka.

```
PaStream* stream;

// Inicijaliziraj PortAudio
Pa_Initialize();

// Otvori strujanje PortAudio-a
Pa_OpenDefaultStream(&stream,
    num_input_channels, num_output_channels,
    sample_format, sample_rate, frames_per_buffer,
    stream_callback,
    user_data);

// Pokreni strujanje
Pa_StartStream(stream);
```

Kako bi obrada zvuka bila ispravna, pri uspostavi strujanja bitno je pažljivo postaviti parametre funkcije `Pa_OpenDefaultStream()`.

Prvi parametar uzima adresu pokazivača (eng. *pointer*) na `PaStream` objekt.

Parametar `num_input_channels` odnosi se na broj zvučnih kanala (eng. *sound channels*) kojima će se ulazno strujanje služiti. Parametar `num_output_channels` odnosi se na broj izlaznih zvučnih kanala. Primjerice, za obradu ili izvedbu stereo zvuka (lijevi i desni kanali), vrijednost ovih parametara bit će 2.

Četvrti parametar `sample_format` predstavlja format zvučnih uzoraka u strujanju. Najčešći formati su `paFloat32` (32-bitni decimalni broj - *float*), `paInt16`

(16-bitni cjelobrojni broj - *integer*), itd. Izbor formata ovisi o preciznosti kojom se signal nastoji prikazati.

Peti parametar `sample_rate` precizira frekvenciju uzorkovanja kojom će zvuk biti obrađen, dok šesti parametar `frames_per_buffer` određuje veličinu ulaznih i izlaznih *buffer*-a.

Predzadnji parametar `stream_callback` traži pokazivač na funkciju povratnog poziva (`paCallback()`) koju će PortAudio pozivati za obradu zvuka.

Zadnji parametar `user_data` pruža mogućnost prosljeđivanja dodatnih korisničkih podataka funkciji povratnog poziva.

9.1.3. Prekid strujanja zvuka

Jedan od načina za prekid strujanja zvuka je ulazom od korisnika. U implementaciji rada ovo je postignuto pritiskom tipke *Enter*. Program do tog trenutka izvršava obradu zvuka preko funkcije `paCallback()`, ali ne nastavlja s izvršavanjem naredbi iza `std::cin.get();` dok ne primi ulaz od korisnika.

Nastavkom programa se PortAudio-vim metodama zaustavlja i zatvara strujanje te dealociraju resursi koje je biblioteka koristila.

```
// Pricekaj upute korisnika
std::cout << "Press␣Enter␣to␣quit..." << std::endl;
std::cin.get();

// Zaustavi i zatvori strujanje
Pa_StopStream(stream);
Pa_CloseStream(stream);

// Dealociraj PortAudio-ve resurse
Pa_Terminate();
```

9.2. Parametri za detekciju temeljne frekvencije

Parametri vezani za algoritam za detekciju temeljne frekvencije uključuju veličinu *buffer*-a (`buffer_size`), frekvenciju uzorkovanja (`sample_rate`), minimalnu i maksimalnu frekvenciju pretraživanja (`min_freq` i `max_freq`) te prag pretrage (`threshold`).

9.2.1. Veličina *buffer*-a

Veličina *buffer*-a (`buffer_size`) određuje koliko uzoraka zvuka se obrađuje odjednom. Veća veličina *buffer*-a može poboljšati preciznost algoritma, ali također povećava računalnu složenost te time izaziva veće kašnjenje rezultata. Manje veličine smanjuju preciznost algoritma, ali i one mogu povećati kašnjenje rezultata ukoliko su premalene.

```
constexpr size_t BUFFER_SIZE = 4096;
```

9.2.2. Frekvencija uzorkovanja

Frekvencija uzorkovanja određuje broj spremljenih uzoraka zvuka u jednoj sekundi. Viša frekvencija uzorkovanja omogućava precizniju detekciju frekvencija, ali također povećava količinu uzoraka koje treba obraditi. Ovaj parametar treba se prilagoditi frekvenciji uzorkovanja ulaznog uređaja (mikrofona) koji se koristi za najbolju funkcionalnost.

```
constexpr uint16_t SAMPLE_RATE = 48000;
```

9.2.3. Minimalna i maksimalna frekvencija

Minimalna (`min_freq`) i maksimalna (`max_freq`) frekvencija pretraživanja određuju raspon frekvencija koje algoritam pokušava detektirati. Ovi parametri su ključni za sužavanje pretrage na relevantne frekvencije, što poboljšava učinkovitost algoritma. U kodu, ti parametri određuju `tau_min` i `tau_max` vrijednosti koje definiraju granice pretrage najboljeg kašnjenja $\hat{\tau}$.

```
// Najniza frekvencija gitare (E2 - 82.41 Hz)
constexpr float MIN_FREQUENCY = 80.0f;
// Najvisa frekvencija gitare (E6 - 1318.51 Hz)
constexpr float MAX_FREQUENCY = 1320.0f;
```

9.2.4. Prag pretrage

Prag pretrage (`threshold`) omogućuje da se samo frekvencije čije vrijednosti CMNDF-a padnu ispod ovog praga smatraju validnim kandidatima za temeljnu frekvenciju. Ovaj parametar pomaže u eliminaciji šumova i netočnih detekcija.

```
constexpr float THRESHOLD = 0.1f;
```

9.3. Detekcija visine tona

Nakon što se snimi zvuk note koju treba detektirati, vrijeme je za njegovu obradu. Kako bi se detektirala nota, prvo je potrebno pronaći temeljnu frekvenciju (tj. visinu tona), a zatim se pomoću nje može izračunati kojoj noti je ta frekvencija najbliža.

9.3.1. Uspostava algoritma

Detekcija visine tona implementirana je funkcijom `detectPitch()` koja uz navedene parametre iz prošlog odjeljka prima parametar `input_buffer`, tj. *buffer* ulaznih uzoraka snimljenog zvuka.

```
float detectPitch(const float* input_buffer,
    int buffer_size, int sample_rate,
    float min_freq, float max_freq,
    float threshold);
```

Prije same obrade signala, uspostavljaju se varijable `tau_min` i `tau_max`, koje će kasnije poslužiti pretrazi visine tona unutar užeg raspona. Maksimalni vremenski odmak bit će predstavljen s `half_buffer_size`.

Budući da se obrada zvuka ne izvodi nad kontinuiranim signalom, nego nad slijedovima blokova signala, pretraga po odmacima u kašnjenju mora imati ograničenje. Kao česti kompromis između raspona kašnjenja i raspona u uzorcima, uzima se pola *buffer*-a kao navedeno ograničenje. Ovime su raspon kašnjenja i raspon uzoraka jednaki pri svakom koraku obrade. Povezano s teorijom od ranije, `half_buffer_size` predstavlja veličinu vremenskog prozora W. Bitno je napomenuti da se u sklopu ovog rada ova vrijednost uvijek skalira s veličinom `input_buffer`-a te će uvijek biti upola te veličine.

```
int tau_min = sample_rate / max_freq;
int tau_max = sample_rate / min_freq;

int half_buffer_size = buffer_size / 2;
```

9.3.2. Funkcija razlike

Implementacija funkcije razlike izvodi se nalik izrazu (8.8). Za svaki odmak u kašnjenju kvadrira se razlika između svakog koreliranog uzorka te zbraja i dodaje nizu `DF_buffer`. Prolaz kroz sve odmake kašnjenja predstavljen je iteracijom po `tau`, dok je prolaz po uzorcima implementiran kao unutarnja iteracija po `t`.

```
// Izracunaj DF
float* DF_buffer = new float[half_buffer_size];

for (int tau = 0; tau < half_buffer_size; tau++) {
    float difference_sum = 0;
    for (int t = 0; t < half_buffer_size; t++) {
        float difference = input_buffer[t] - input_buffer[t + tau];
        difference_sum += difference * difference;
    }
    DF_buffer[tau] = difference_sum;
}
```

Izvršavanjem ovog dijela koda, `DF_buffer` na svakom indeksu sadrži izračunatu funkciju razlike, ekvivalentno izrazu (8.8).

9.3.3. CMNDF

Nakon funkcije razlike, slijedi slična iteracija po odmacima kašnjenja. Budući da se kumulativnoj sumi pridodaje funkcija razlike pri svakoj iteraciji, moguće je optimizirati izraz (8.13) i izostaviti dodatnu ugniježđenu iteraciju po prošlim odmacima kašnjenja j. Za slučaj $\tau = 0$, $\mathrm{CMNDF}(0)$ postavlja se na 1, dok kumulativna suma kreće od $\mathrm{DF}(0)$.

```
// Izracunaj CMNDF
float* CMNDF_buffer = new float[half_buffer_size];
CMNDF_buffer[0] = 1.0f;

float cumulative_sum = DF_buffer[0];
for (int tau = 1; tau < half_buffer_size; tau++) {
    cumulative_sum += DF_buffer[tau];
    CMNDF_buffer[tau] = DF_buffer[tau] * tau / cumulative_sum;
}
```

Kao rezultat ovog dijela koda, `CMNDF_buffer` sadržavat će vrijednost funkcije CMNDF za svaki odmak u kašnjenju u rasponu od 0 do `half_buffer_size`.

9.3.4. Odabir najboljeg kandidata

Zadnji korak u detekciji visine tona bit će iteracija kroz odmake kašnjenja u rasponu od `tau_min` i `tau_max` koji su ranije definirani. U ovom djelu koda dolazi do male izmjene u metodi što je bila spomenuta u teoriji.

Kako bi se izabrao najbolji kandidat za odmak kašnjenja ($\hat{\tau}$), njegova vrijednost CMNDF mora biti manja od zadanog praga (`threshold`) i manja od svih ostalih vrijednosti CMNDF-a. U slučaju takvog kandidata nema, algoritam neće izabrati nijednog kako algoritam ne bi detektirao krive visine tona. Ovime se nastoji predstaviti detektirane visine tona s većom pouzdanosti te osigurati jasniji i konzistentniji ispis korisniku.

```
// Pronadi najboljeg kandidata za visinu tona
float best_pitch = -1.0f;
float best_correlation = threshold;
for (int tau = tau_min; tau < tau_max; tau++) {
    if (CMNDF_buffer[tau] < best_correlation) {
        best_correlation = CMNDF_buffer[tau];
        best_pitch = static_cast<float>(sample_rate) / tau;
    }
}

delete[] DF_buffer;
delete[] CMNDF_buffer;

return best_pitch;
```

Konačno, funkcija `detectPitch()` vraća visinu tona `best_pitch`. Ako nema kandidata za visinu tona, funkcija vraća vrijednost -1. Sljedeći isječak koda prikazuje funkciju u cijelosti:

```
//// ALGORITAM ZA PREPOZNAVANJE VISINE TONA:
float detectPitch(const float* input_buffer, int buffer_size,
    int sample_rate, float min_freq, float max_freq,
    float threshold) {

    int tau_min = sample_rate / max_freq;
    int tau_max = sample_rate / min_freq;

    int half_buffer_size = buffer_size / 2;

    // Izracunaj DF
    float* DF_buffer = new float[half_buffer_size];

    for (int tau = 0; tau < half_buffer_size; tau++) {
        float difference_sum = 0;
        for (int t = 0; t < half_buffer_size; t++) {
            float difference =
```

```
                input_buffer[t] - input_buffer[t + tau];
            difference_sum += difference * difference;
        }
        DF_buffer[tau] = difference_sum;
    }

    // Izracunaj CMNDF
    float* CMNDF_buffer = new float[half_buffer_size];
    CMNDF_buffer[0] = 1.0f;

    float cumulative_sum = DF_buffer[0];
    for (int tau = 1; tau < half_buffer_size; tau++) {
        cumulative_sum += DF_buffer[tau];
        CMNDF_buffer[tau] = DF_buffer[tau] * tau / cumulative_sum;
    }

    // Pronadi najboljeg kandidata visine tona
    float best_pitch = -1.0f;
    float best_correlation = threshold;
    for (int tau = tau_min; tau < tau_max; tau++) {
        if (CMNDF_buffer[tau] < best_correlation) {
            best_correlation = CMNDF_buffer[tau];
            best_pitch = static_cast<float>(sample_rate) / tau;
        }
    }

    delete[] DF_buffer;
    delete[] CMNDF_buffer;

    return best_pitch;
}
```

9.4. Mapiranje note

Funkcija `getNoteFromPitch()` za jedini parametar prima visinu tona temeljem koje će izračunati najbližu notu do preciznosti polutona.

```
std::string getNoteFromPitch(float pitch);
```

Niz *string*-ova `note_names` sadrži 12 imena za svaki poluton oktave u kromatskoj ljestvici. Kako bi se visina tona pripojila nekom od imena u nizu, prvo je potrebno preformulirati visinu tona u broj indeksa pripadnog imena.

U ranijim poglavljima napomenuto je da se visine tona nota računaju relativno koncertnoj visini tona. Koncertna visina tona A_4 hardkodirana je na 440 Hz, a MIDI (eng. *Musical Instrument Digital Interface*) standard za tu notu predstavljen je brojem 69.[22] Iz izraza (3.1) moguće je izvesti formulu za izračun visine tona note kao:

$$f = 440 \cdot 2^{(n-69)/12} \tag{9.1}$$

MIDI broj note postiže se obrnutom formulom:

$$n = 12 \cdot \log_2(\frac{f}{440}) + 69 \tag{9.2}$$

Kako bi se izračunao broj indeksa, MIDI broj note se dodatno dijeli s 12 (brojem članova niza, tj. polutona u oktavi) i uzima ostatak. Uz poluton note, preostaje izračunati razred, tj. broj oktave. Ovaj izračun postaje trivijalan uz MIDI broj note.

```
std::string getNoteFromPitch(float pitch) {
    static const std::string note_names[] = {
        "C", "C#", "D", "D#", "E", "F",
        "F#", "G", "G#", "A", "A#", "B"
    };
    int note_MIDI = static_cast<int>(
        round(12 * log2(pitch / 440.0) + 69));
    int octave = (note_MIDI / 12) - 1;

    return note_names[note_MIDI % 12] + std::to_string(octave);
}
```

Konačno, aplikacija funkcijom `getNoteFromPitch()` vraća detektiranu notu u obliku *string*-a.

9.5. Funkcija povratnog poziva

Navedena funkcija za detekciju visine tona `getPitch()` i funkcija za mapiranje nota `getNoteFromPitch()` pozivaju se u funkciji povratnog poziva `paCallback()`. Uz to se radi provjera za detekciju te se samo ispisuju note s detektiranom visinom tona.

```
//// FUNKCIJA POVRATNOG POZIVA PORTAUDIO-A:
int paCallback(const void* input_buffer, void* output_buffer,
    unsigned long frames_per_buffer,
    const PaStreamCallbackTimeInfo* time_info,
    PaStreamCallbackFlags status_flags,
```

```
    void* user_data) {

    float detected_pitch = detectPitch((const float*) input_buffer,
        BUFFER_SIZE, SAMPLE_RATE, MIN_FREQUENCY, MAX_FREQUENCY,
        THRESHOLD);
    if (detected_pitch != -1) {
        std::cout << "Note: " << std::setw(5) << std::left <<
            getNoteFromPitch(detected_pitch) <<
            "Pitch: " << std::fixed << std::setprecision(2) <<
            detected_pitch << " Hz" << std::endl;
    }
    return paContinue;
}
```

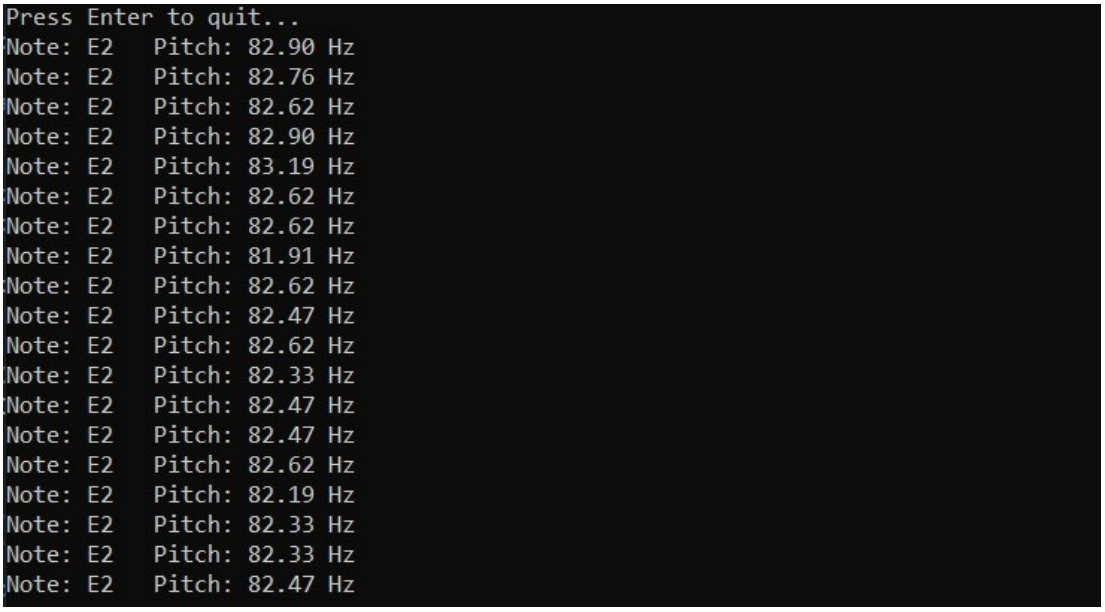
9.6. Primjer rada

Kao primjer za demonstraciju, aplikacija se može testirati sviranjem žica gitare. Sviranjem gornje E_2 žice, aplikacija će snimiti audio signal u slijednim blokovima duljine `BUFFER_SIZE = 4096`, po `SAMPLE_RATE = 48000` uzoraka u sekundi.

Ako se frekvencije ograniče na raspon gitare (od približno 80 Hz do 1320 Hz), raspon odmaka kašnjenja `tau` tijekom pretrage bit će od 36 do 600. Nakon izračuna DF i CMNDF minimuma, jednom od pretraga kandidata izabire se `tau = 582`. Frekvencija zatim se računa prema izrazu (8.4) kao `48000/582 = 82.47`, što je približna visina tona odsvirane žice E_2 koja se traži, preciznije 82.41 Hz.

Nakon detekcije visine tona, nota se mapira izračunom njenog MIDI broja i oktave. Izračunom `note_MIDI` dobiva se broj 40, a ostatak dijeljenja s 12 daje broj indeksa 4. Kako indeksi kreću od 0, traži se peti član u nizu `note_names`, koji je upravo ime "E". Jedino što sad preostaje je izračunati oktavu, koja ispadne broj 2.

Konačno, na zaslon se ispisuje detektirana nota "E2", zajedno s njenom detektiranom visinom tona "82.47 Hz":



Slika 9.1: Primjer izlaza za detektiranu notu E_2

10. Zaključak

Digitalna obrada signala predstavlja širok opseg područja informacijskih tehnologija, od jednostavne obrade zvuka u glazbi do telekomunikacija u satelitskoj industriji. Algoritmi za obradu signala poput FFT razvijaju se i nadograđuju od druge polovice 20. stoljeća te mogu biti korišteni za rješavanje raznovrsnih problema gdje je potrebo provesti analizu frekvencijskog spektra u signalu.

U ovom radu istraženi su teoretski i praktični aspekti detekcije glazbenih nota analizom frekvencija. Prolaskom kroz osnovne pojmove zvuka i glazbe kao što su frekvencije, harmonici i oktave, uz njihovo ponašanje na primjeru gitare, cilj je bio jasno demonstrirati proces detekcije nota.

Kao praktična demonstracija razvijena je aplikacija koja koristi koncepte YIN algoritma kako bi precizno detektirala visine tona u stvarnom vremenu. Razmatranjem ključnih parametara algoritma kao što su frekvencija uzorkovanja i veličina *buffer*-a može se zaključiti da svaki od njih utječe na kompromis između preciznosti i brzine detekcije te njihove vrijednosti ovise o namjeni aplikacije. Parametri poput minimalne i maksimalne frekvencije te praga pretrage omogućavaju da se apriornim znanjem smanji vrijeme pretrage i poveća pouzdanost rezultata.

Implementacija algoritma i primjena informacija iz teorije pokazala je kako digitalna obrada signala može učinkovito pomoći u analizi i interpretaciji glazbenih signala te time glazbenicima pružiti korisne alate. Daljnje istraživanje može se usmjeriti na ubrzanje i preciznost algoritma te proširenje raspona nota koje aplikacija može detektirati.

Danas su algoritmi za detekciju visine tona i nota široko zastupljeni u aplikacijama za štimanje glazbenih instrumenata.

LITERATURA

[1] Alain De Cheveigné and Hideki Kawahara. Yin, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 111(4):1917–1930, 2002.

[2] Stanley S Stevens and John Volkmann. The relation of pitch to frequency: A revised scale. *The American Journal of Psychology*, 1940.

[3] ISO 16:1975 — iso.org. `https://www.iso.org/standard/3601.html`. [Accessed 09-06-2024].

[4] Ebenezer Prout. *Harmony: its theory and practice*. Cambridge University Press, 2011.

[5] Dave Benson. *Music: A mathematical offering*. Cambridge University Press, 2006.

[6] Vlastimir Pericic. Visejezicni recnik muzickih termina. 2008.

[7] Hrvatska enciklopedija (1941.-1945.), Apr 2020.

[8] B Benward and M Saker. Introduction. the materials of music: Sound and time. *Music in theory and practice, pp. xii–25). NY, NY: McGraw-Hill*, 2003.

[9] Johan Sundberg. *In Tune or Not?: a study of fundamental frequency in music practise*. na, 1982.

[10] dijatonika - Hrvatska enciklopedija — enciklopedija.hr. `https://www.enciklopedija.hr/clanak/dijatonika`. [Accessed 05-06-2024].

[11] William A Sethares. *Tuning, timbre, spectrum, scale*. Springer Science & Business Media, 2005.

[12] Fundamental and harmonic frequencies, May 2020.

[13] Thomas Christensen and Jean-Philippe Rameau. Eighteenth-century science and the "corps sonore:" the scientific background to rameau's "principle of harmony". *Journal of Music Theory*, 31(1):23–50, 1987.

[14] Carl Reinke. Spectroid, 2018.

[15] Jeremy Gustine. Guitar tuner - pitch detection for dummies, Jul 2021.

[16] James D Broesch. *Digital signal processing: instant access*. Elsevier, 2008.

[17] Emmanuel J Candès and Michael B Wakin. An introduction to compressive sampling. *IEEE signal processing magazine*, 2008.

[18] Douglas Self. *Audio engineering explained*. Routledge, 2012.

[19] Mary Lourde and Anjali Kuppayil Saji. A digital guitar tuner. *arXiv e-prints*, pages arXiv–0912, 2009.

[20] Sample rate, bit depth and buffer size explained – focusrite audio ..., Jan 2022.

[21] PortAudio - an Open-Source Cross-Platform Audio API — portaudio.com. `https://www.portaudio.com/`. [Accessed 17-06-2024].

[22] MIDI note numbers and center frequencies | Inspired Acoustics — inspiredacoustics.com. `https://inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies`. [Accessed 18-06-2024].

Aplikacija za detekciju glazbenih nota

Sažetak

Ovaj rad objašnjava raspored glazbenih nota po oktavama i njihovu povezanost s frekvencijama zvučnog signala. Pokriva detekciju glazbenih nota analizom frekvencija te implementaciju jednostavnog algoritma za preciznu detekciju nota u stvarnom vremenu.

Ključne riječi: Digitalna obrada signala, FFT, YIN, frekvencijski spektar, harmonici, visina tona, aplikacija za detekciju nota

Application for detection of musical notes

Abstract

This paper explains the arrangement of musical notes by octaves and their relationship to sound signal frequencies. It covers the detection of musical notes through frequency analysis and the implementation of a simple algorithm for accurate note detection in real time.

Keywords: Digital signal processing, FFT, YIN, frequency spectrum, harmonics, pitch, note detection application