Implementacija faznog vokodera

Lelas, Zvonko

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1254

IMPLEMENTACIJA FAZNOG VOKODERA

Zvonko Lelas

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1254

IMPLEMENTACIJA FAZNOG VOKODERA

Zvonko Lelas

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1254

Pristupnik:	**Zvonko Lelas (0036540175)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Antonio Petošić

Zadatak: **Implementacija faznog vokodera**

Opis zadatka:

U završnom radu će se teorijski analizirati efekt faznog vokodera s promjenom parametara signala u frekvencijskoj domeni. Analizirati će se brzina izvođenja efekta s obzirom na podjelu ulaznog signala na blokove, njegovu frekvencijsku analizu i promjenu njegovih parametara u frekvencijskoj domeni i vraćanje u vremensku domenu. Efekt će se implementirati na sustavu za obradu zvučnog signala u stvarnom vremenu BELA te će se analizirati njegovi parametri i njihov utjecaj na brzinu izvođenja algoritma.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

Uvod

U ovom završnom radu detaljno se razmatra teorijski koncept faznog vokodera te se upoznaje s razvojnim okruženjem Bela koje će poslužiti kao platforma za implementaciju faznog vokodera i tri efekta: šaptanje, robotizacija i promjena tonaliteta. Kroz sustav Bela bit će implementiran i jednostavno grafičko korisničko sučelje (GUI) i osciloskop te spektralni analizator kako bi se olakšalo korištenje i praćenje rada faznog vokodera u stvarnom vremenu.

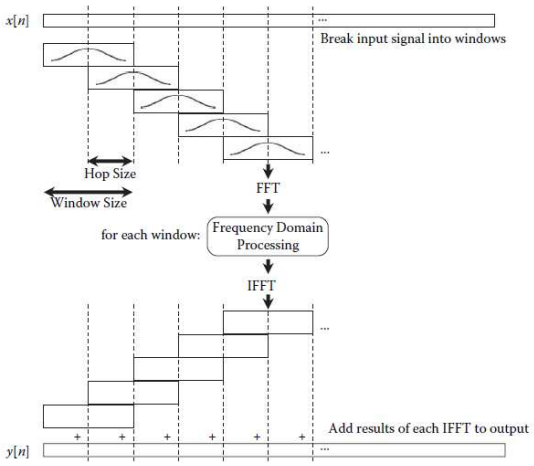
Fazni vokoder je tehnika je tehnika obrade zvuka koja manipulira frekvencijom i fazom signala kako bi se postigli razni efekti. Ovaj rad će istražiti zašto se fazni vokoder koristi za ove efekte, prikazujući kako je lakše postići željene rezultate u frekvencijskoj domeni nego u vremenskoj domeni.

Kroz praktičnu implementaciju faznog vokodera na Bela platformi, cilj je demonstrirati prednosti ove tehnike obrade zvuka i njezinu primjenu u stvaranju različitih audio efekata visokih performansi i malog kašnjenja.

Ovaj rad će pružiti uvid u teorijske i praktične aspekte faznog vokodera te pokazati kako se ova tehnika može koristiti za stvaranje raznovrsnih zvučnih efekata s visokom kvalitetom i fleksibilnošću.

1. Fazni vokoder

Termin fazni vokoder koristi se za opisivanje skupine tehnika analize i sinteze zvuka koje obavljaju obradu signala u frekvencijskoj domeni. Dok se većina audio efekata implementira izravno na nadolazećem signalu u vremenskoj domeni, fazni vokoder manipulira frekvencijom i fazom kako bi postigao željeni efekt. Ove tehnike koriste se za različite audio efekte, uključujući istezanje vremena, promjenu tonaliteta, robotizaciju i šaptanje [1]. Osnovne operacije faznog vokodera koje su opisane u nastavku, prikazane su na *Slika 1.1*.



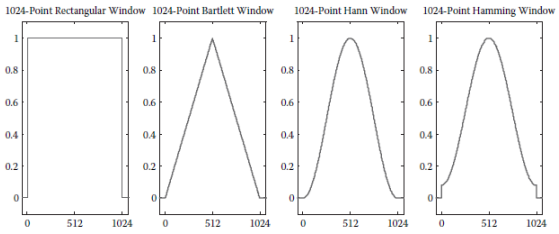
Slika 1.1 Osnovne operacije faznog vokodera [2]

1.1. Razdvajanje ulaznog signala

Iz ulaznog kontinuiranog signala uzimaju se uzastopni uzorci veličine *N*, tj. veličine okvira (*window size*). Okvire obrađuje prozorska funkcija koja prima N pozitivnih vrijednosti, dok su sve ostale vrijednosti 0. Ovisno o implementaciji, postoje različite prozorske funkcije, od kojih su najznačajnije pravokutna, Hann, Bartlett i Hamming funkcija koje su prikazane na *Slika 1.2*

Ovisno o odabranoj funkciji, preklapanje okvira različito utječe na izlazni signal. Pravokutni prozor može uzrokovati bočne režnjeve u frekvencijskoj domeni, dok Hann i Hamming prozori pružaju bolji kompromis između vremenske i frekvencijske rezolucije [1]. Česta implementacija faznog vokodera koristi Hann [1] prozorske funkcije (1) jer omogućuju manji prijenos energije jedne frekvencije u drugu, što rezultira čišćim zvučnim signalom.

$$w_{hann}[n] = \begin{cases} 0.54 - 0.56\ \cos\frac{2\pi n}{N-1}, 0 \le n\ \le N \\ 0, |n|\ \ge N \end{cases} \quad (1)$$



Slika 1.2 Osnovne prozorske funkcije [2]

Prozoriranje se inače koristi kod konstrukcije digitalnih filtara modificiranjem idealnog odziva funkcije sinc(x).

1.2. Transformacija u frekvencijsku domenu

Nakon što je okvir proveden kroz prozorsku funkciju, primjenjuje se brza Fourierova transformacija (FFT). S obzirom na to da je uzorak podijeljen na N točaka, FFT mora biti veličine $M \geq N$. Iako se u praksi često koristi $M = N$, ponekad veći FFT-ovi mogu dati optimiziranija rješenja. Na primjer, korištenje potencija broja 2 pridonosi računskoj učinkovitosti.

Kombinacija prozorske funkcije i brze Fourierove transformacije naziva se kratkotrajna Fourierova transformacija (STFT), koja je dana izrazom (2)[1]. STFT je algoritam za izračun diskretne Fourierove transformacije signala te kao izlaz proizvodi M ravnomjerno razmaknutih binova u frekvencijskoj domeni.

Frekvencijski signali su uvijek realni brojevi, ali vrijednost frekvencijskih binova je kompleksna vrijednost. Zbog jednostavnosti, ona se često prikazuje polarnom reprezentacijom, kombinirajući amplitudu i fazu.

$$X[m,k] = \sum_{n=-\infty}^{\infty} x[n]h[m-n]e^{-\frac{j2\pi nk}{M}} \qquad (2)$$

Smisao FFT-a je određivanja stupnja sličnosti ulaznog niza uzoraka x[n] sa nizom analizirajućih funkcija.

1.3. Modifikacija amplitude i faze

Modificiranjem amplitude i/ili faze postižu se željeni efekti faznog vokodera. Neki od tih efekata su navedeni u sljedećim potpoglavljima.

1.3.1. Robotizacija

Robotizacija (*robotisation*) se najčešće primjenjuje na glasovima. Ovaj efekt postavlja fazu svakog frekvencijskog bina na 0, zadržavajući istu amplitudu, što rezultira robotskim glasom. Postavljanje faze na nula ekvivalentno je transformiranju svakog broja u realan, tj. gubi se imaginarni dio kao što je prikazano formulom (3)[2]. Ovaj proces rezultira signalom koji čuva vokalne formante, ali svaka frekvencijska komponenta počinje od nule, stoga nema glatkog povezivanja s prethodnim skokom. Ton robotiziranog glasa određuje se formulom (4), gdje je f_s stopa uzorkovanja, a H veličina skoka [2]. Zvuk efekta ovisi o veličini prozora; mali prozori smanjuju razumljivost robotiziranog glasa, dok veliki prozori umanjuju robotičnost.

$$MAG = \sqrt{{a_k}^2 + {b_k}^2} \tag{3}$$

$$f = \frac{f_s}{H} \tag{4}$$

1.3.2. Šaptanje

Efekt šaptanja (whisperisation)eliminira osjećaj visine tona postavljanjem faze na slučajne vrijednosti u rasponu od 0 do 2π. Kao rezultat toga, gubi se bilo kakav osjećaj periodičnosti od jednog okvira do drugog. Formula za dobivanje efekta šaptanja dana je formulom (5)[2]. Najvažniji parametar efekta šaptanje je veličina prozora N. Kraći prozori su učinkovitiji u eliminiranju bilo kakve visine tona (*pitch*).

$$X_k = \sqrt{{a_k}^2 + {b_k}^2}(\cos\theta_k + j \sin\theta_k) \tag{5}$$

1.3.3. Proširenje vremena

Vremensko istezanje (*time shift*) mijenja trajanje signala bez promjene visine tona. Za implementaciju ovog efekta koristi se osnovni odnos vremena, frekvencije i faze ($\phi = \omega \cdot t$). Dakle, frekvencija se može očuvati, a vrijeme mijenjati pod cijenu faza signala. Iako se gubi početna faza, rezultati ovog efekta su vrlo dobri. S obzirom da duljina trajanja ulaznog i izlaznog signala nije ista, ovaj efekt se ne može primjenjivati na signale u stvarnom vremenu, ali je podloga za efekt tonskog skaliranja.

1.3.4. Skaliranje frekvencije

Tonsko skaliranje (*pitch shift*) mijenja visinu tona bez promjene brzine izvođenja. Jednostavnom manipulacijom algoritma za istezanje vremena lako se može postići željeni efekt. Promjenom brzine izvođenja za faktor *R* dobije se dulji izlazni uzorak, ali ako se istom uzorku ponovno promijeni brzina izvođenja za *1/R*, kao rezultat na zvučniku se osjeti promjena u visini tona, ali ne i u brzini reprodukcije signala [1].

1.3.5. Disperzija u frekvencijskoj domeni

Efekt disperzije (*dispersion*) postiže se odgodom u različitim vremenskim intervalima određenih frekvencijskih komponenti. Rezultira širenjem zvuka u vremenu, uzrokujući gubitak koherencije, osobito kod prolaznih dijelova signala. U telekomunikacijama, ovo se obično smatra nepoželjnim efektom, ali u glazbi može biti korisno za stvaranje specifičnih zvučnih efekata [2]. U faznom vokoderu, disperzija se može dogoditi tijekom vremenskog skaliranja ako svaki frekvencijski bin ima različitu fazu, što rezultira različitim frekvencijama i fazama sintetiziranih parcijala.

1.4. Povratak u vremensku domenu

Modificirani blokovi se vraćaju u vremensku domenu pomoću inverzne brze Fourierove transformacije (IFFT) kako bi se dobio konačni izlazni signal. Ovaj proces je poznat kao sinteza ili rekonstrukcija, koja proizvodi *M* točaka u vremenskoj domeni. Ponekad je potrebno primijeniti dodatni prozor kako bi se smanjili čujni artefakti prije dodavanja rezultata u izlazni međuspremnik. Nakon toga slijedi preklapanje-dodavanje proces, gdje se obrađeni okviri dodaju kako bi se formirao konačni izlazni signal [1].

2. Bela.io

Bela je moćna platforma za interaktivnu obradu audiosignala koja omogućava stvaranje sofisticiranih audio sustava s visokom brzinom obrade podataka u stvarnom vremenu i malim kašnjenjem (10ak ms) Kombinacija prilagođene sklopovske i programske potpore čini Belu idealnim rješenjem za projekte koji zahtijevaju brzu i pouzdanu obradu zvuka i vanjskih senzora.

2.1. Bela sklopovski sustav

Implementacija faznog vokodera radit će se unutar Bela sklopovskog sustava koji služi za stvaranje interakcije senzora i zvuka. Bela je pogodna za izradu faznog vokodera zbog svoje visoke brzine obrade podataka u stvarnom vremenu, niske latencije i pružanja obrade zvuka i senzora na jednom uređaju [3].

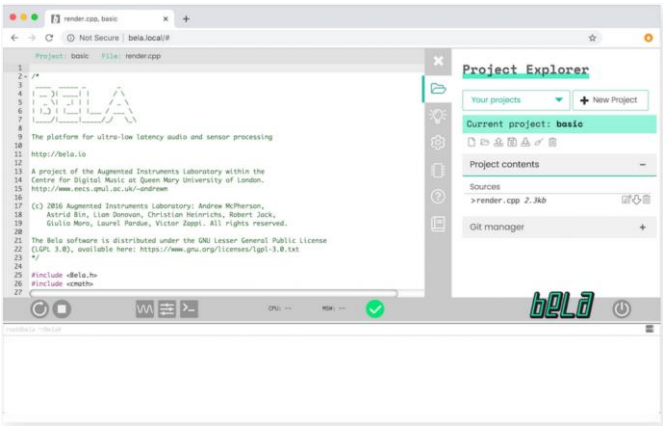
Bela sklopovski sustav prikazan je na Slika *2.3*.

2.2. Bela sustav programske potpore

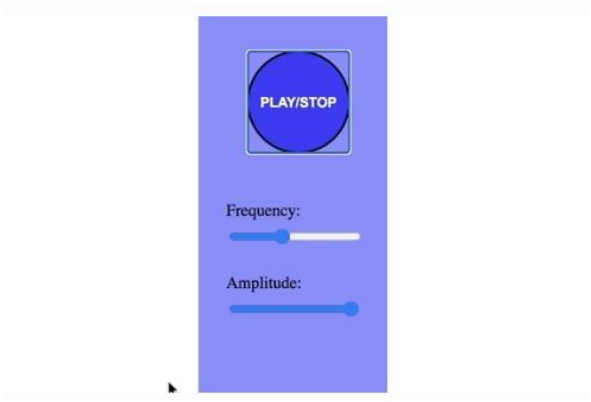
Bela programska potpora je prilagođeni operacijski sustav koji radi isključivo na Bela uređajima. Uključuje Bela IDE, integrirano razvojno okruženje koje olakšava pisanje i testiranje koda. Kao sustav koji kombinira snažnu računalnu obradu s fleksibilnim mogućnostima povezivanja i niske latencije, Bela predstavlja idealno rješenje za istraživanje granica interaktivne audio obrade [3].

Bela IDE prikazan je na *Slika 2.1*.

Osim performansi, Bela pruža širok niz edukacijskih materijala za upoznavanje s radom na Bela uređajima te mogućnost izrade grafičkog korisničkog sučelja (GUI), što je u svrhu ovog rada i više nego dovoljno. Primjer izrađenog sučelja prikazan je na *Slika 2.2*



Slika 2.1 Bela IDE [3]



Slika 2.2 Primjer korisničkog sučelja izrađen unutar Bela IDE [3]

Slika 2.3 Bela sklopovski sustav [3]

3. Implementacija faznog vokodera

Implementacija počinje izgradnjom osnovne strukture faznog vokodera, u koju se naknadno ubacuju zasebni algoritmi efekata. Efekti koji su implementirani u svrhu ovog završnog rada su robotizacija, šaptanje i promjena tonaliteta. Nadalje, implementiran je i jednostavno grafičko korisničko sučelje koje služi za odabir efekta te veličine prozora skoka kako bi usporedile latencije u ovisnosti o veličini prozora.

Uz GUI, Bela nudi i prikaz ulaznog i izlaznog signala u vremenskoj i frekvencijskoj domeni za pregled primjene efekata. Dani rezultati koriste se kao dokaz učinkovitosti faznog vokodera za odabrane efekte, upravo iz razloga što su ti efekti teški za implementirati u vremenskoj domeni.

U funkciji `bool setup(BelaContext *context, void *userData)` koja je prikazana na *Slika 3.1* postavljeni su parametri za inicijalizaciju. Inicijalizirani su FFT i veličine prozora za izračun SFFT-a, `gScope` koji služi za prikaz grafova, `gGui` i `gGuiController` za prikaz klizača (*sliders*) te dretva za FFT.

```
bool setup(BelaContext *context, void *userData)
{
    // Set up the FFT and its buffers
    gFft.setup(gFftSize);
    gInputBuffer.resize(gBufferSize);
    gOutputBuffer.resize(gBufferSize);

    // Calculate the window
    gAnalysisWindowBuffer.resize(gFftSize);
    gSynthesisWindowBuffer.resize(gFftSize);
    recalculate_window(gFftSize);

    // Initialise the number of samples between successive pulses
    // and the length of the pulse
    // gTestInterval = context->audioSampleRate / kTestsPerSecond;
    // gPulseLength = context->audioSampleRate * kPulseLength;

    // Initialise the oscilloscope
    gScope.setup(2, context->audioSampleRate);

    // Set up the GUI
    gGui.setup(context->projectName);
    gGuiController.setup(&gGui, "0 = Pitch Shift, 1 = robotisation, 2 = whisperisation");

    // Setup GUI sliders
    gPitchShiftSliderIdx = gGuiController.addSlider("Shift (semitones)", 0, -24, 24, 0.1);
    gHopSizeSliderIdx = gGuiController.addSlider("Hop Size (256)", 256, 64, 1024, 1);
    gEffectSliderIdx = gGuiController.addSlider("Effect", 0, 0, 2, 1);

    // Set up the thread for the FFT
    gFftTask = Bela_createAuxiliaryTask(process_fft_background, 50, "bela-process-fft");

    return true;
}
```

Slika 3.1 Funkcija *setup*

Nakon toga, u funkciji void render(BelaContext *context, void *userData), koja se koristi kao glavna funkcija, uzima se vrijednosti od GUI-a u slučaju da su se promijenile, što je prikazano na *Slika 3.2*.

```
// Get the pitch shift in semitones from the GUI slider and convert to ratio
float pitchShiftSemitones = gGuiController.getSliderValue(0);
gPitchShift = powf(2.0, pitchShiftSemitones / 12.0);

// Read GUI controls
int hopSize = (int)gGuiController.getSliderValue(1);

//if hop size changes, recalculate the window based on an overlap factor of 4
if(hopSize != gHopSize) {
    int newLength = hopSize * 4;
    if(newLength > gFftSize)
        newLength = gFftSize;
    recalculate_window(newLength);

    gHopSize = hopSize;
}
```

Slika 3.2 Promjene vrijednosti u ovisnosti o GUI vrijednostima

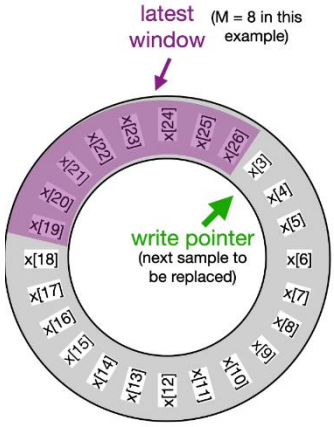
Sljedeći zadatak je čitati ulazne podatke koji na Belu dolaze preko mikrofona, što je već implementirano unutar Bela IDE pomoću funkcije audioRead(). Prikaz unosa ulaznih podataka prikazuje *Slika 3.3*. Ulazni podatci se šalju na gInputBuffer koji je implementiran kao kružni međuspremnik s pokazivačem gInputBufferPointer kao što je prikazano na *Slika 3.5*. Dretva FFT obrađuje te podatke i ne dozvoljava čitanje neobrađenih podataka, stoga sljedeći korak u *render* funkciji jest uzimanje podataka koji su obrađeni pomoću gOutputBufferReadPointer pokazivača te slanje istih na izlaz. Prikaz ispisa izlaznih podataka nalazi se na *Slika 3.4*. Za slanje podataka na zvučnik Bela također ima implementiranu funkciju audioWrite() i funkciju gScope.log() koja prikazuje podatke na osciloskopu.

```
for(unsigned int n = 0; n < context->audioFrames; n++) {
    // Read the next sample from the buffer
    float in = audioRead(context, n, 0) + audioRead(context, n, 1) + audioRead(context, n, 2) + audioRead(context, n, 3) +
    audioRead(context, n, 4) + audioRead(context, n, 5) + audioRead(context, n, 6) + audioRead(context, n, 7);
    // Store the sample ("in") in a buffer for the FFT
    // Increment the pointer and when full window has been
    // assembled, call process_fft()
    gInputBuffer[gInputBufferPointer++] = in;
    if(gInputBufferPointer >= gBufferSize) {
        // Wrap the circular buffer
        // Notice: this is not the condition for starting a new FFT
        gInputBufferPointer = 0;
    }
```

Slika 3.3 Unos ulaznih podataka

```
    // Write the audio to the output
    for(unsigned int channel = 0; channel < context->audioOutChannels; channel++) {
        audioWrite(context, n, channel, out);
    }

    // Log to the Scope
    gScope.log(in, out);
}
```

Slika 3.4 Ispis izlaznih podataka



Slika 3.5 Kružni međuspremnik [3]

Funkcija recalculate_window(unsigned int length) implementira promjenu veličine prozora unutar GUI-a, stoga mijenja parametre u ovisnosti o veličini međuspremnika za analizu i sintezu te ponovno računa Hann funkciju za prozoriranje. Implementacija Hann funkcije prikazana je *Slika 3.6*.

```
    for(int n = 0; n < length; n++){
        // Hann window, split across analysis and synthesis window
        gAnalysisWindowBuffer[n] = 0.5f * (1.0f - cosf(2.0 * M_PI * n / (float)(length-1)));
        gSynthesisWindowBuffer[n] = gAnalysisWindowBuffer[n];
    }
}
```

Slika 3.6 Implementacija Hann funkcije

S obzirom na to da nakon što se provedu FFT i promjene odabranog efekta na signalu, signal se mora ponovno prozorirati kako bi se ispravno zbrojio na izlazni signal bez artefakata.

Glavni dio faznog vokodera nalazi se u funkciji void process_fft(std::vector<float> const& inBuffer, unsigned int inPointer, std::vector<float>& outBuffer, unsigned int outPointer) koja ovisno o varijabli koji primjenjuje odgovarajući efekt na signal. Ako je varijabla koji jednaka nuli, primjenjuje se promjena tonaliteta; ako je varijabla koji jednaka jedinici, primjenjuje se efekt šaptanja; a ako je jednaka dvojci, primjenjuje se efekt robotizacije.

Efekt promjene tonaliteta implementiran je koristeći niz vektora koji sadržavaju informacije o vremenskim i frekvencijskim karakteristikama trenutnog i prošlog uzorka. Kada primijenimo FFT, donja polovica FFT binova se prepisuje u fftCurrentOut međuspremnik. Gornja polovica FFT binova je samo kompleksno konjugirani broj donjeg dijela te ne sadrži nikakvu novu informaciju, stoga se ne upisuje zbog očuvanja memorije i brzine programa.

Slijedi primjena promjene tonaliteta. Za svaki od binova uzima se njegova amplituda i faza. U varijabli phaseDiff nalazi se razlika trenutne i prijašnje faze kako bi se koristila u izračunu trenutne modificirane faze. Na kraju spremamo novo izračunate frekvencije, a u analysisMagnitudes i lastInputPhases spremaju se amplituda i faza za daljnje korištenje u procesu sinteze. Na *Slika 3.7* prikazan je kod za izračun novih frekvencija te spremanje važnih podataka.

```
for(int n = 0; n <= gFftSize/2; n++) {
    // Turn real and imaginary components into amplitude and phase
    float amplitude = gFft.fda(n);
    float phase = atan2f(gFft.fdi(n), gFft.fdr(n));

    // Calculate the phase difference in this bin between the last
    // hop and this one, which will indirectly give us the exact frequency
    float phaseDiff = phase - lastInputPhases[n];

    // Subtract the amount of phase increment we'd expect to see based
    // on the centre frequency of this bin (2*pi*n/gFftSize) for this
    // hop size, then wrap to the range -pi to pi
    float binCentreFrequency = 2.0 * M_PI * (float)n / (float)gFftSize;
    phaseDiff = wrapPhase(phaseDiff - binCentreFrequency * gHopSize);

    // Find deviation in (fractional) number of bins from the centre frequency
    float binDeviation = phaseDiff * (float)gFftSize / (float)gHopSize / (2.0 * M_PI);

    // Add the original bin number to get the fractional bin where this partial belongs
    analysisFrequencies[n] = (float)n + binDeviation;

    // Save the magnitude for later and for the GUI
    analysisMagnitudes[n] = amplitude;

    // Save the phase for next hop
    lastInputPhases[n] = phase;
}
```

Slika 3.7 Implementacija analize promjene tonaliteta

Sljedeći korak jest sinteza frekvencija u nove magnitude i faze kako bi se pravilno primijenila inverzna Fourierova transformacija bez artefakata. Proces sinteze prikazan je na *Slika 3.8*. U posljednjem koraku se spremaju vrijednosti novih amplituda i faza na izlaz, a za gornju polovicu ulaza upisuju se kompleksno konjugirani brojevi.

```
    float outPhase = wrapPhase(lastOutputPhases[n] + phaseDiff);
    // TODO: Now convert magnitude and phase back to real and imaginary c
    gFft.fdr(n) = amplitude * cosf(outPhase);   // change me
    gFft.fdi(n) = amplitude * sinf(outPhase);   // change me

    // Also store the complex conjugate in the upper half of the spectrum
    if(n > 0 && n < gFftSize / 2) {
        gFft.fdr(gFftSize - n) = gFft.fdr(n);
        gFft.fdi(gFftSize - n) = -gFft.fdi(n);
    }

    // TODO: Save the phase for the next hop
    lastOutputPhases[n] = outPhase;
}
```

Slika 3.8 Implementacija sinteze promjene tonaliteta

Funkcija wrapPhase osigurava da se faza nalazi u rasponu - π do π.

```
// Wrap the phase to the range -pi to pi
float wrapPhase(float phaseIn)
{
    if (phaseIn >= 0)
        return fmodf(phaseIn + M_PI, 2.0 * M_PI) - M_PI;
    else
        return fmodf(phaseIn - M_PI, -2.0 * M_PI) + M_PI;
}
```

Kod 3.1 – program za osigurati da se faza nalazi u rasponu –π do π.

Implementacija šaptanja je nešto jednostavnija jer sve što treba napraviti jest provesti ulazne podatke kroz FFT, binove spremiti u međuspremnik te za svaki bin promijeniti fazu na slučajno generiran broj kako bi dobili efekt diskontinuiteta podataka, što zapravo kao rezultat daje efekt šaptanja.

```
// Whisperise the output
for(int n = 0; n < gFftSize; n++){
      float amplitude = gFft.fda(n);
      float phase = 2.0 * M_PI * (float)rand()/(float)RAND_MAX;
      gFft.fdr(n) = amplitude * cosf_neon(phase);
      gFft.fdi(n) = amplitude * sinf_neon(phase);
}
```

Kod 3.2 – program za implementaciju efekta šaptanja

Robotizacija se obrađuje identično, ali se faza postavlja u 0. Zapravo, imaginarni dio bina se briše, te kao rezultat izlazni signal nema izražen ton i visinu glasa pa kao izlazni signal zvuči robotizirano.

```
// Robotise the output
for(int n = 0; n < gFftSize; n++) {
      float amplitude = gFft.fda(n);
      gFft.fdr(n) = amplitude;
      gFft.fdi(n) = 0;
}
```

Kod 3.3 – program za implementaciju efekta robotizacije

Nakon obrade efekta, provodi se inverzna Fourierova transformacija kako bi se podaci vratili u vremensku domenu i bili spremni za reprodukciju kao što je prikazano na *Slika 3.9*.

```
// Run the inverse FFT
gFft.ifft();

// Add timeDomainOut into the output buffer
for(int n = 0; n < gFftSize; n++) {
    int circularBufferIndex = (outPointer + n - gFftSize + gBufferSize) % gBufferSize;
    outBuffer[circularBufferIndex] += gFft.td(n) * gSynthesisWindowBuffer[n];
}
```

Slika 3.9 Isječak koda s inverznom FFT

Dodatno, u implementaciji se koristio kod prikazan *Slika 3.10* kako bi se ispisali podatci o kašnjenju, ali s obzirom da se na GUI može slati ograničen broj podataka, zbog brzine izvođenja kod je zakomentiran te se lako može testirati u kratkom periodu trajanja programa.

```
if(++gTestCounter >=gTestInterval) {
    gTestCounter = 0;

    if(gInTest){
        rt_printf("Pulse timeout\n");
    }
    gInTest = true;
}

// If we are counting samples, then look for a rising edge above the threshold
if(gInTest) {
    if(!gInputAboveThreshold && in >= kInputHiThreshold) {
        // Edge found: stop counting samples
        gInTest = false;
        gInputAboveThreshold = true;

        int normalisedSamples = gTestCounter - 2*context->audioFrames;

        // Print the calculated latency
        rt_printf("Latency: %.2fms (%d samples) --- hardware: %.2fms (%d samples)\n",
                  1000.0 * gTestCounter / context->audioSampleRate,
                  gTestCounter,
                  1000.0 * normalisedSamples / context->audioSampleRate,
                  normalisedSamples);
    }

    // Update the previous input
    gLastInput = in;
}


// If the input falls below the lower threshold, set the flag accordingly
if(in < kInputLoThreshold) {
    gInputAboveThreshold = false;
}
```

Slika 3.10 Kod za testiranje ukupne latencije

GUI je implementiran unutar datoteke *sketch.js*. Funkcija setup() postavlja inicijalne vrijednosti klizača, njihovu poziciju i vrijednosti koje se mogu odabrati kako je prikazano na *Slika 3.11*.

```
function setup() {
    createCanvas(windowWidth, windowHeight);

    // Create and position the sliders
    pitchSlider = createSlider(-12, 12, 0, 0.1);
    pitchSlider.position(10, 10);
    pitchSlider.style('width', '2000px'); // Set width to 2000px

    hopSizeSlider = createSlider(64, 1024, 256, 1);
    hopSizeSlider.position(10, 70); // Adjust position to avoid overlap
    hopSizeSlider.style('width', '2000px'); // Set width to 2000px

    effectSlider = createSlider(0, 2, 0, 1);
    effectSlider.position(10, 130); // Adjust position to avoid overlap
    effectSlider.style('width', '2000px'); // Set width to 2000px

    // Attach input handlers
    pitchSlider.input(sendToBela);
    hopSizeSlider.input(sendToBela);
    effectSlider.input(sendToBela);
}
```

Slika 3.11 Inicijalizacija klizača

Funkcija draw(), prikazana na *Slika 3.12*, koristi se za slanje vrijednosti sa GUI-a na Belu, kako bi promjenom nekog od klizača Bela pravovremeno promijenila vrijednosti varijabli visine tonaliteta, veličine prozora skoka i varijable koji koja određuje koji efekt se primjenjuje.

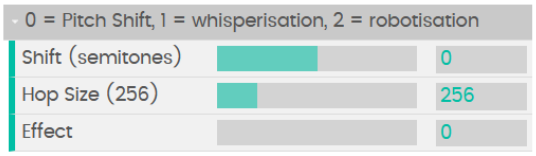
```
function draw() {
    background(255);

    // Display labels for sliders
    textSize(32); // Increase text size for better readability
    fill(0); // Set text color to black for better contrast
    text('Pitch Shift:', 2020, 35); // Adjust label position
    text('Hop Size:', 2020, 95); // Adjust label position
    text('Effect:', 2020, 155); // Adjust label position

    // Get slider value
    let pitchValue = pi  let hopSizeSlider: any
    let hopSizeValue = hopSizeSlider.value();
    let effectValue = effectSlider.value();

    // Send slider values to Bela
    Bela.data.sendValue(0, pitchValue);
    Bela.data.sendValue(1, hopSizeValue);
    Bela.data.sendValue(2, effectValue);
}
```

Slika 3.12 Slanje vrijednosti sa GUI-a

Funkcija sendToBela() implementira komunikaciju GUI-a i Bele, ali ona je zadana unutar Bela IDE sustava.

4. Rezultati

Bela korisničko grafičko sučelje za implementaciju željenih efekata, implementirano u ovom radu prikazano je Slika *4.1*.



Slika 4.1 Korisničko grafičko sučelje

Jednostavnim dizajnom osigurana je poboljšana performansa faznog vokodera jer Bela ne troši previše vremena na komunikaciju s GUI-jem, čime se također osigurava jednostavnost korištenja. Klizač za pomak tonaliteta (*Shift*) koristi se za pomak tonaliteta i ne utječe na druge efekte. Klizač za veličinu skoka (*Hop size*) utječe na sve efekte, ali se prilikom korištenja efekta promjene tonaliteta ne koristi jer veličina prozora za kvalitetnu primjenu tog efekta mora biti fiksna. Zadnji klizač (*Effect*) koristi se za promjenu efekta koji se koristi, a pri vrhu je dana legenda za korištenje tog klizača.

4.1. Kašnjenje

Sljedeće slike prikazuju kašnjenje u ovisnosti o primijenjenom efektu te veličini prozora skoka. U ispisu se može vidjeti prikaz ukupnog kašnjenja i kašnjenja opreme pa se kašnjenje faznog vokodera može uzeti kao razlika tih dviju vrijednosti.

```
bela: Latency: 211.25ms (9316 samples) --- hardware: 210.52ms (9284 samples)
bela: Latency: 1.11ms (49 samples) --- hardware: 0.39ms (17 samples)
bela: Pulse timeout
bela: Pulse timeout
bela: Pulse timeout
bela: Latency: 123.22ms (5434 samples) --- hardware: 122.49ms (5402 samples)
bela: Latency: 0.16ms (7 samples) --- hardware: -0.57ms (-25 samples)
bela: Latency: 5.74ms (253 samples) --- hardware: 5.01ms (221 samples)
bela: Latency: 0.61ms (27 samples) --- hardware: -0.11ms (-5 samples)
bela: Latency: 1.04ms (46 samples) --- hardware: 0.32ms (14 samples)
bela: Latency: 3.22ms (142 samples) --- hardware: 2.49ms (110 samples)
```

Slika 4.2 Kašnjenje prilikom korištenja efekta promjene tonaliteta

```
bela: Latency: 1.63ms (72 samples) --- hardware: 0.91ms (40 samples)
bela: Latency: 2.00ms (88 samples) --- hardware: 1.27ms (56 samples)
bela: Latency: 1.56ms (69 samples) --- hardware: 0.84ms (37 samples)
bela: Latency: 5.58ms (246 samples) --- hardware: 4.85ms (214 samples)
bela: Latency: 6.42ms (283 samples) --- hardware: 5.69ms (251 samples)
bela: Latency: 3.15ms (139 samples) --- hardware: 2.43ms (107 samples)
bela: Latency: 7.51ms (331 samples) --- hardware: 6.78ms (299 samples)
bela: Latency: 0.00ms (0 samples) --- hardware: -0.73ms (-32 samples)
bela: Latency: 1.56ms (69 samples) --- hardware: 0.84ms (37 samples)
bela: Latency: 5.94ms (262 samples) --- hardware: 5.22ms (230 samples)
bela: Latency: 0.36ms (16 samples) --- hardware: -0.36ms (-16 samples)
bela: Latency: 0.00ms (0 samples) --- hardware: -0.73ms (-32 samples)
bela: Latency: 4.72ms (208 samples) --- hardware: 3.99ms (176 samples)
```

Slika 4.3 Kašnjenje prilikom korištenja efekta robotizacije s veličinom prozora skoka 256

```
bela: Pulse timeout
bela: Latency: 174.92ms (7714 samples) --- hardware: 174.20ms (7682 samples)
bela: Latency: 3.06ms (135 samples) --- hardware: 2.34ms (103 samples)
bela: Pulse timeout
bela: Pulse timeout
bela: Latency: 165.40ms (7294 samples) --- hardware: 164.67ms (7262 samples)
bela: Latency: 0.00ms (0 samples) --- hardware: -0.73ms (-32 samples)
bela: Pulse timeout
bela: Pulse timeout
bela: Latency: 24.13ms (1064 samples) --- hardware: 23.40ms (1032 samples)
bela: Latency: 0.00ms (0 samples) --- hardware: -0.73ms (-32 samples)
bela: Pulse timeout
bela: Pulse timeout
bela: Pulse timeout
```

Slika 4.4 Kašnjenje prilikom korištenja efekta robotizacije s veličinom prozora skoka 1024

```
bela: Latency: 1.04ms (46 samples) --- hardware: 0.32ms (14 samples)
bela: Latency: 0.00ms (0 samples) --- hardware: -0.73ms (-32 samples)
bela: Latency: 0.00ms (0 samples) --- hardware: -0.73ms (-32 samples)
bela: Latency: 2.06ms (91 samples) --- hardware: 1.34ms (59 samples)
bela: Latency: 1.36ms (60 samples) --- hardware: 0.63ms (28 samples)
bela: Latency: 0.09ms (4 samples) --- hardware: -0.63ms (-28 samples)
bela: Latency: 2.02ms (89 samples) --- hardware: 1.29ms (57 samples)
bela: Latency: 0.00ms (0 samples) --- hardware: -0.73ms (-32 samples)
bela: Latency: 2.29ms (101 samples) --- hardware: 1.56ms (69 samples)
bela: Latency: 0.00ms (0 samples) --- hardware: -0.73ms (-32 samples)
bela: Latency: 0.34ms (15 samples) --- hardware: -0.39ms (-17 samples)
bela: Latency: 0.00ms (0 samples) --- hardware: -0.73ms (-32 samples)
bela: Latency: 0.00ms (0 samples) --- hardware: -0.73ms (-32 samples)
```

Slika 4.5 Kašnjenje prilikom korištenja efekta šaptanja s veličinom prozora skoka 256

```
bela: Latency: 0.00ms (0 samples) --- hardware: -0.73ms (-32 samples)
bela: Pulse timeout
bela: Latency: 145.26ms (6406 samples) --- hardware: 144.54ms (6374 samples)
bela: Latency: 0.00ms (0 samples) --- hardware: -0.73ms (-32 samples)
bela: Pulse timeout
bela: Latency: 110.79ms (4886 samples) --- hardware: 110.07ms (4854 samples)
bela: Latency: 0.00ms (0 samples) --- hardware: -0.73ms (-32 samples)
bela: Pulse timeout
bela: Latency: 75.17ms (3315 samples) --- hardware: 74.44ms (3283 samples)
bela: Latency: 4.85ms (214 samples) --- hardware: 4.13ms (182 samples)
bela: Pulse timeout
bela: Latency: 41.52ms (1831 samples) --- hardware: 40.79ms (1799 samples)
```

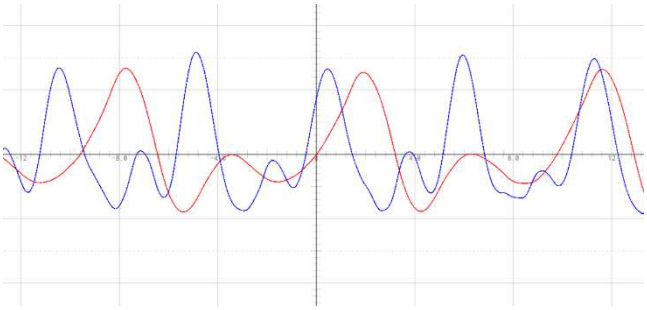
Slika 4.6 Kašnjenje prilikom korištenja efekta šaptanja s veličinom prozora skoka 128

```
bela: Latency: 1.63ms (72 samples) --- hardware: 0.91ms (40 samples)
bela: Latency: 3.65ms (161 samples) --- hardware: 2.93ms (129 samples)
bela: Latency: 1.77ms (78 samples) --- hardware: 1.04ms (46 samples)
bela: Latency: 3.40ms (150 samples) --- hardware: 2.68ms (118 samples)
bela: Latency: 0.29ms (13 samples) --- hardware: -0.43ms (-19 samples)
bela: Latency: 1.93ms (85 samples) --- hardware: 1.20ms (53 samples)
bela: Latency: 2.47ms (109 samples) --- hardware: 1.75ms (77 samples)
```

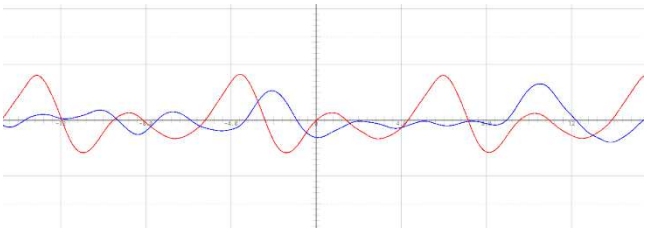
Slika 4.7 Kašnjenje prilikom korištenja efekta šaptanja s veličinom prozora skoka 1024

4.2. Vremenska i frekvencijska domena

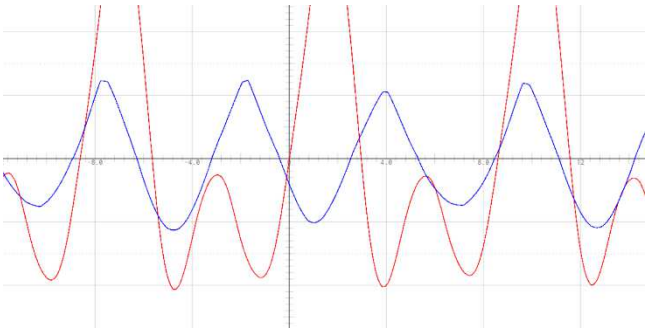
Sljedeće slike prikazuju prikaz pojedinih efekata u vremenskoj i frekvencijskoj domeni snimljene pomoću osciloskopa koji je uključen u Bela IDE.



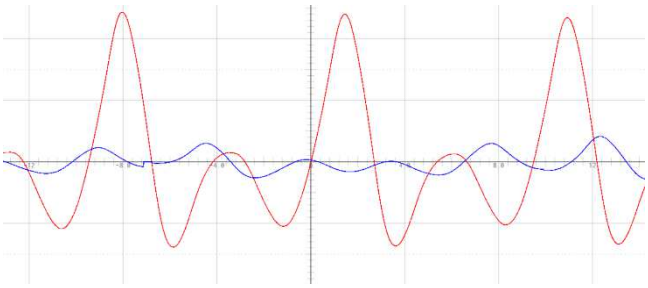
Slika 4.8 Prikaz promjene tonaliteta u vremenskoj domeni



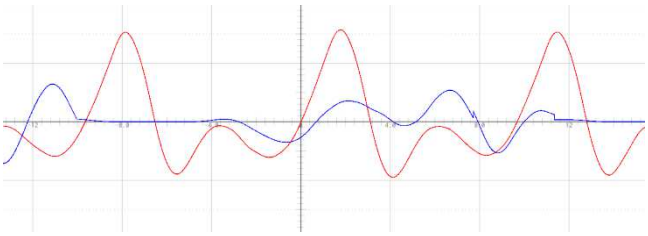
Slika 4.9 Prikaz efekta šaptanja u vremenskoj domeni s veličinom prozora 256



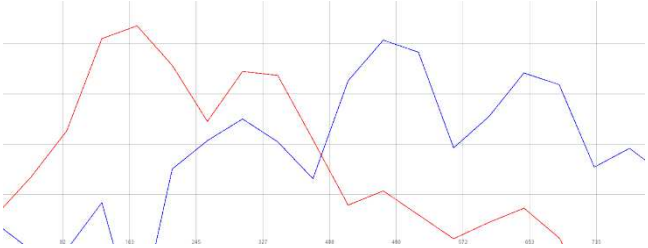
Slika 4.10 Prikaz efekta robotizacije u vremenskoj domeni s veličinom prozora 256



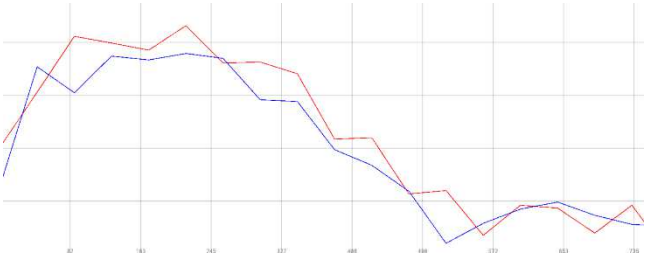
Slika 4.11 Prikaz efekta robotizacije u vremenskoj domeni s veličinom prozora 1024



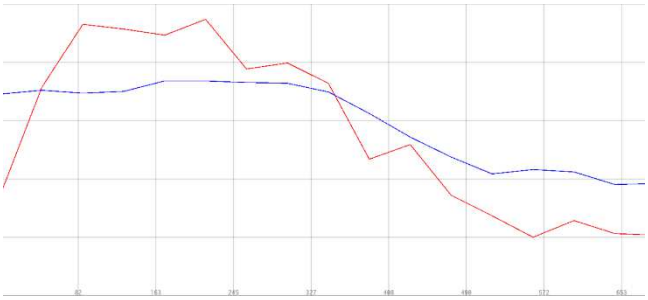
Slika 4.12 Prikaz efekta šaptanja u vremenskoj domeni s veličinom prozora 1024



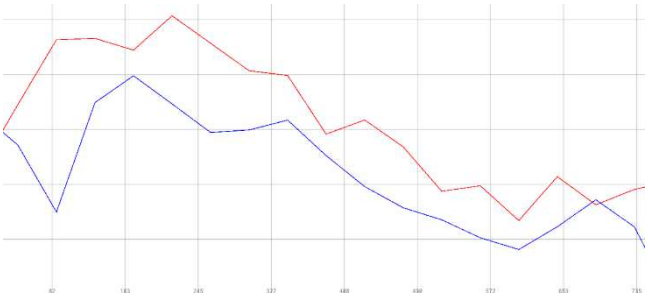
Slika 4.13 Prikaz promjene tonaliteta u frekvencijskoj domeni



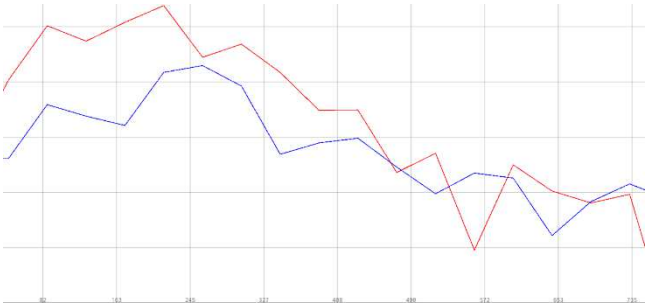
Slika 4.14 Prikaz efekta šaptanja u frekvencijskoj domeni s veličinom prozora 256



Slika 4.15 Prikaz efekta šaptanja u frekvencijskoj domeni s veličinom prozora 1024



Slika 4.16 Prikaz efekta robotizacije u frekvencijskoj domeni s veličinom prozora 256



Slika 4.17 Prikaz efekta robotizacije u frekvencijskoj domeni s veličinom prozora 1024

Zaključak

Zaključak ovog završnog rada donosi sažetak teorijskih i praktičnih aspekata faznog vokodera te demonstrira implementaciju istog na Bela platformi. Fazni vokoder predstavlja moćnu tehniku obrade zvuka koja manipulira frekvencijom i fazom signala kako bi postigla različite efekte poput promjene tonaliteta, robotizacije i šaptanja. Za razliku od obrade u vremenskoj domeni, fazni vokoder omogućuje lakše postizanje željenih rezultata u frekvencijskoj domeni.

Kroz implementaciju faznog vokodera na Bela platformi, pokazuje se kako se ova tehnika može primijeniti u stvaranju visokokvalitetnih audio efekata s malom latencijom. Osim toga, korištenjem Bela IDE-a olakšava se proces razvoja i testiranja, a grafičko korisničko sučelje dodatno olakšava korištenje faznog vokodera.

Analizom rezultata, uočava se utjecaj veličine prozora skoka na latenciju i kvalitetu generiranih efekata. Manje veličine prozora skoka obično rezultiraju manjom latencijom, dok veće veličine prozora skoka mogu povećati latenciju.

U konačnici, ovaj rad pruža uvid u teorijske i praktične aspekte faznog vokodera te pokazuje kako se ova tehnika može koristiti za stvaranje raznovrsnih zvučnih efekata s visokom kvalitetom i fleksibilnošću.

Literatura

[1] Reiss, J. D., McPherson, A. *Audio Effects: Theory, Implementation and Application*. 1. izdanje. Boca Raton: CRC Press, 2015.

[2] Zölzer, U., Amatriain, X., Arfib, D., Bonada, J., et al. *DAFX: Digital Audio Effects*. 1. izdanje. Chichester: John Wiley & Sons, 2002.

[3] *learn.bela.io*. Bela Platform. Poveznica: https://learn.bela.io/; pristupljeno 10.6.2024.

[4] *freesound.org*. Freesound. Poveznica: https://freesound.org/; pristupljeno 10.6.2024.

Sažetak

Implementacija faznog vokodera

Ovaj rad istražuje fazni vokoder, tehniku za analizu i sintezu zvuka koja omogućava visokokvalitetnu obradu audio signala. Objašnjeni su teorijski koncepti faznog vokodera, uključujući Fourierove transformacije, prozoriranje te pojedini efekti, i praktični aspekti implementacije faznog vokodera. Rad uključuje primjer stvaranja tri odabrana zvučna efekta koristeći vokoder unutar interaktivnog sučelja i programske potpore Bela.io. Time se demonstriraju njegove mogućnosti u tehničkoj audio produkciji. Na kraju, rad zaključuje kako fazni vokoder predstavlja moćan alat za naprednu manipulaciju zvuka.

Ključne riječi: fazni vokoder, zvučni efekti, Bela, obrada audio signala

Summary

Phase Vocoder Implementation

This work explores the phase vocoder, a technique for sound analysis and synthesis that enables high-quality audio signal processing. The theoretical concepts of the phase vocoder, including Fourier transformations, windowing and various effects, are explained along with the practical aspects of vocoder implementation. The paper includes an example of creating three selected sound effects using the vocoder within an interactive interface and software support Bela.io. This demonstrates its capabilities in technical audio production. Finally, the paper concludes that the phase vocoder is a powerful tool for advanced sound manipulation.

Keyword: phase vocoder, sound effects, Bela, audio signal processing

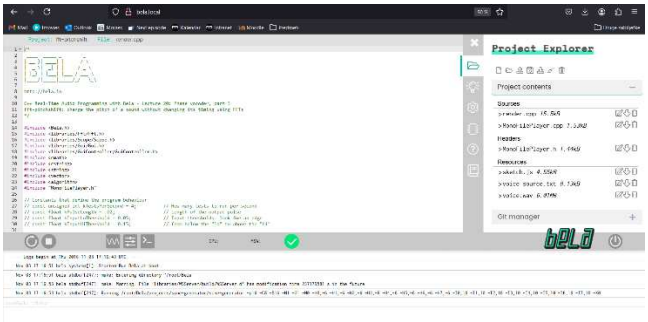
Privitak

Upute za korištenje sklopovske i programske potpore

Za korištenje faznog vokodera trebate imati Bela Starter Kit ili Bela Mini Starter Kit prikazan na *Slika 0-1*. Priključite Belu pomoću USB kabela u vaš laptop te povežite Belu na zvučnike i mikrofon. Kada lampice na Beli krenu treptati sačekajte 15 sekundi te nastavite dalje (lampice će nastaviti treptati, ali 15 sekundi je dovoljno da se Bela spoji na sustav).

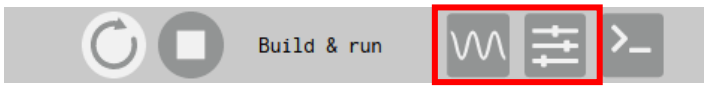
Slika 0-1 Bela Mini Starter Kit [3]

Zatim u nekom od internetskih preglednika otiđite na http://bela.local ili http://192.168.7.2 (Linux) i http://192.168.6.2 (Windows). Sada bi trebao biti otvoren Bela IDE prikazan Slika *0-2*.



Slika 0-2 Bela IDE

Nakon toga, trebate učitati .zip datoteku s https://github.com/Zl54017/implementation-of-phase-vocoder-with-Bela-IDE.git te ju povući (drag-and-drop) unutar Bela IDE. Sada bi sa strane trebali vidjeti učitan otpakirani projekt. Program se pokreće klikom na *Build and run* gumb, *a osciloskop i GUI se pokreću preko gumbova prikazanih na Slici 0-3.*



Slici 0-3 Gumbovi za pokretanje osciloskopa i GUI-a