Phase Vocoder without Occurring the Phase Unwrapping Problem

Cristian Constantin 341C2

Abstract:

Un phase vocoder este un procesor de semnal folosit de exemplu in industria audio pentru a obține diverse efecte precum auto-tune sau aplicații mai simple precum schimbarea vitezei de redare. Problema cu un astfel de phase vocoder este ca uneori pot apărea probleme precum o distorsiune a semnalului când poate vrem sa redam un podcast la viteza 1.5x, iar asta nu este o situatie ideală pentru ca am pierde informație astfel si dintr-un semnal audio pe care îl putem auzi fără problem ajungem in situația in care sa nu putem înțelege cuvintele rostite in semnalul inițial.

Introducere:

Articolul Time-Scale and Pitch-Scale Modification by the Phase Vocoder without Occurring the Phase Unwrapping Problem scris de Ryoichi Yoneguchi (Meiji University Graduate School of Science and Technology) si Takahiro Murakami (Meiji University School of Science and Technology) prezintă o varianta de phase-vocoder care evita problema de phase unwrapping intalnita intr-un phase vocoder normal, problema care apare atunci când vocoderul estimează frecvența unghiulară instantanee, iar din cauza limitărilor în calculul fazei rezultă distorsiuni în forma de undă a semnalului modificat. Varianta propusă este o abordare diferită care in loc să se bazeze pe estimarea frecvenței unghiulare instantanee, aceasta folosește direct diferența de faze între cadrele de analiză adiacente pentru a modifica faza semnalului si astfel evită necesitatea de a estima frecvența unghiulară instantanee, acest lucru ajutând la evitarea problemei de tip phase unwrapping.

Pentru acest proiect am pornit de la aceasta abordare descrisă in articol pentru a obține un phase vocoder care sa ofere rezultate mai bune fata de unul convențional, astfel proiectul include 2 implementări: o implementare pentru un phase vocoder normal si o implementare care pornește de la un phase vocoder normal si face modificările pentru semnal urmând ideea din articol.

Un phase vocoder simplu implica in esență 3 pași principali pentru a obține un nou semnal: aplicarea FFT(Fast Fourier Transform) asupra semnalului astfel încât sa fie posibilă procesarea sa pentru ca astfel trecem semnalul in domeniul frecvenței, procesarea fazei si a magnitudinii in cadrul căreia ajungem sa aplicam parametrii primiți precum rata de redare sau rata pentru pitch si in final aplicam un IFFT(Inverse Fast Fourier Transform) pentru a trece semnalul in domeniul timpului așa cum era trimis inițial către FFT.

Implementare:

Implementarea proiectului a presupus 3 etape: Implementarea funcțiilor de tip helper, Implementarea celor 2 variante de phase vocoder si Implementarea secțiunii de testare pentru rezultatele generate.

Functii Helper - Pentru a putea utiliza un phase vocoder in python in mod inițial am nevoie sa pot trimite semnalul audio către funcția de phase vocoder si ulterior sa pot scrie într-un fiiser audio semnalul rezultat. In cadrul implementării exista 2 funcții helper: write wave si read_wave. Write_wave folosește modulul wave din python si setea parametrii necesari pentru a scrie fișierul wave. Read-wave este o functie cu unicul scop de citi fișierul audio pentru phase vocoder si începe prin a verifica sample_width-ul care in esență de referă la calitatea audio si am 2 cazuri pe care le tratez: 16 bit si 32 bit. In cazul proiectului meu, generez datele din intrare in GarageBand si export-ul pentru fișiere are setarea de 16-bit care mai este cunoscut si drept “CD

Quality”. Daca avem unul din cele 2 cazuri trebuie sa verificam daca avem un semnal mono sau stereo pentru ca avem nevoie de un semnal mono si astfel daca este cazul transformam semnalul stereo in semnal mono si in cazul proiectului meu acest lucru are loc pentru înregistrez semnalul in mod stereo in GarageBand folosind mai multe microfoane.

Implementare phase vocoder simplu - implementarea pentru phase vocoder începe prin a verifica daca semnalul primit este unul de tip mono si in cazul in care da efectuam STFT pe semnal cu un window size pentru transformata Fourier de 1024 si hop size de 256. Dupa aplicarea STFT avem nevoie de matricea in care stocam varianta prelucrata a matricei date de STFT pentru ca in funcție de rata pentru timp e posibil sa avem un semnal de o durata diferită si astfel avem nevoie si de un vector de timp nou. In următoarea secțiune a funcției putem modifica scara temporala, reprezentând astfel secțiunea din cod in care schimbam viteza de redare a semnalului nostru si in aceasta secțiune realizam diferențele de faza specifice phase vocoder si tot aici este secțiunea in care poate apărea phase unwrapping si este important acest aspect pentru ca o sa revenim la el pentru implementarea vocoder-ului din articol. Parcurgem fiecare coloană a matricei STFT care va fi modificată temporal cu un for si pentru fiecare moment `t` în matricea STFT întinsă temporal, se calculează `col_idx`, un index al coloanei din matricea STFT originală. Acest index este modificat în funcție de factorul de întindere temporală, apoi pentru fiecare coloană, se calculează diferența de fază între cadrele succesive ale matricei STFT **[1]** si in final aplicăm modificările în matricea STFT. Pentru modificarea pitch-ului ne trebuie o noua matrice STFT in care o sa avem rezultatul final si parcurgem toate frecvențele din matricea STFT modificată temporal si pentru fiecare frecvență `k`, se calculează un nou index `k_psm`, care este determinat de factorul de scalare a înălțimii (`psm_rate`). La final folosim ISTFT pentru a obține semnalul audio modificat.

Implementare phase vocoder articol - scopul phase vocoder-ului din articol este acela de a elimina problema de phase unwrapping si o prima abordare cât mai simplă pentru aceasta problema este secțiunea amintită la punctul **[1]** unde după ce fac diferența de faza folosind numpy.angle aș putea foarte simplu sa folosesc biblioteca numpy si funcția unwrap asupra diferenței de faza pentru a rezolva aceasta problema, dar am ales sa urmez articolul si sa implementez algoritmii propuși. Diferenta in implementare fata de phase vocoder-ul simplu este data de modificarea fazelor, astfel in următorul cod: stretched_stft_matrix_tsm[:, t] = np.abs(stft_matrix[:, col_idx]) * np.exp(1j * (np.angle(stretched_stft_matrix_tsm[:, t - 1]) + phase_diff)) folosim stretched_stft_matrix_tsm in loc de stft_matrix ca in phase vocoder-ul simplu ca sa obținem astfel ceea ce propune articolul: folosirea directă diferenței de faze între cadrele de analiză adiacente. Alta diferenta fata de abordarea simplă este vizibila tot in acest cod pentru ca articolul menționează faptul ca pentru valori ale tsm_rate care nu sunt întregi o sa fie accentuată problema pentru ca in phase vocoder-ul normal in loc sa adunam phase_dif direct o sa înmulțim cu tsm_rate.

Implemntare funcții de testare - Pentru aceasta secțiune folosesc spectogram din scipy.signal pentru a genera spectograme pentru semnalele audio necesare pentru analiza cantitativă si funcția de plot din matplotlib.pyplot pentru a obține un waveform care sa ne arate problema de phase unwrapping. Pentru a evalua numeric performanta folosesc o funcție care generează raportul SNR intre cele 2 semnale.

Date:

Pentru secțiunea de date am ales sa îmi generez 5 fișiere audio in GarageBand, 4 sunt înregistrări ale vocii mele in care număr de la 1 la 10(acest aspect o sa fie detaliat in discuția despre Spectograme) si 3 dintre aceste 4 fișiere au deja un filtru aplicat asupra lor precum deeper voice, robot voice sau telephone voice pentru a putea evidenția diferența de performanta intre cele 2 metode. Fisierul numărul 5 urmează modelul de testare amintit in articolul inițial si reprezintă o înregistrare de aproximativ 30 de secunde a ciripitului de păsări pentru ca un astfel de sunet este predispus la problema de phase unwrapping. De mention este faptul ca evaluările au fost făcute pe 3 fișiere: fișierul cu vocea mea fără un filtru drept caz de baza, fișierul cu filtru deeper voice din GarageBand întrucât este cel mai dificil din cele 3 filtre pentru phase vocoder in urma testării si fișierul cu ciripitul de păsări pentru a putea reproduce situația mentioanata in articol.

Tot pentru date am folosit inițial pentru testare si audio_sample-urile din următorul proiect pentru autotune in python: https://github.com/PyExplained/Autotune-using-Python/tree/main/audio_samples si o sa includ rezultatele obținute pentru Sample-ul C Major Vocal pentru ca are o calitate mai ridicată.

Evaluare:

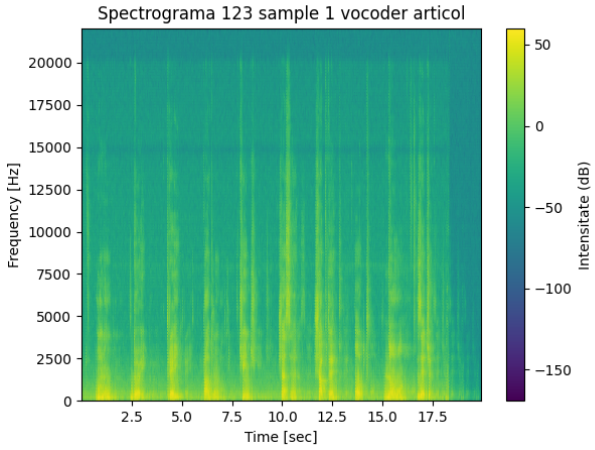
Evaluare cantitativă:

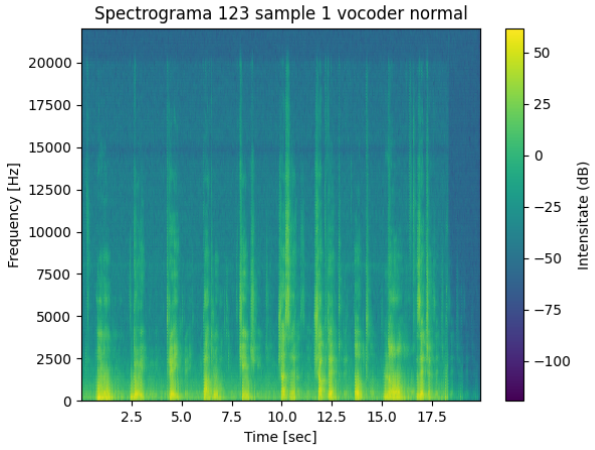
Pentru evaluarea cantitativă am folosit 3 indicii: Spectograme, Waveform si Raport SNR si am sa menționez sample-urile pentru care se pot obține concluzii mai usor. De menționat pentru aceste valori este faptul ca pentru Sample 1, 2 si C Major parametrii de rulare pentru phase vocoder au fost 1.5 pentru tsm si 1 pentru psm, iar pentru Sample 5 valorile au fost 1.5 si 3 pentru a urma exemplul de simulare din articol.

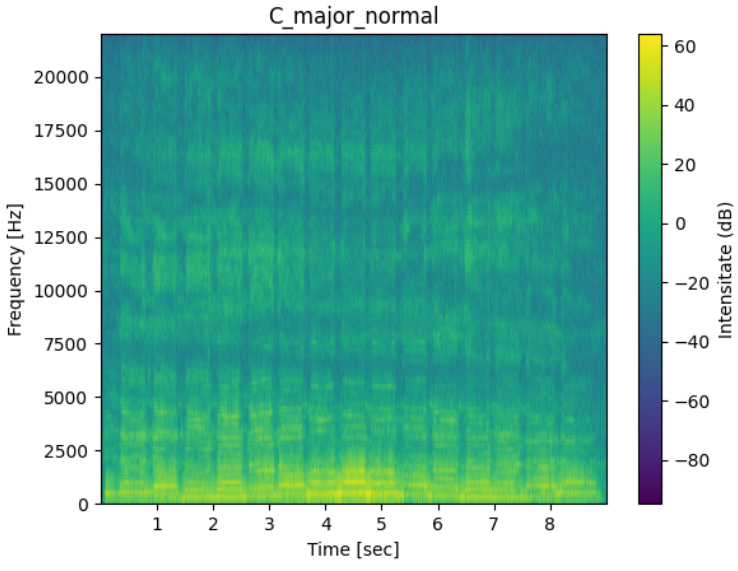
1. Valori SNR (normal/articol):
-SNR Sample1: 1.741101442008811 dB
-SNR Sample2: -0.3418349227733592 dB
-SNR Sample5: 0.7324084431242194 dB
-SNR Sample C major: 2.729103844368449 dB

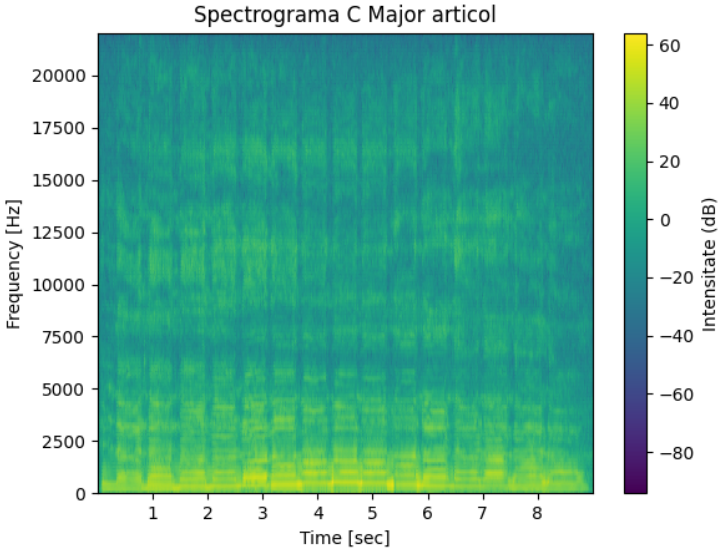
Rezultatele obținute pentru primele 3 sample-uri indica o îmbunătățire daca luam in context si evaluarea calitativă pt ca la sample-ul 1 reusim sa auzim sunetul mai clar, dar pentru sample-ul 2 care este puternic afectat de phase unwrapping deși reușim sa auzim mai bine cifrele avem o valoare negativa si asta se datorează calității reduse a înregistrării, astfel am inclus si Sample-ul C Major de calitate ridicată care indica in mod clar o îmbunătățire cu o valoare de peste 2 DB.

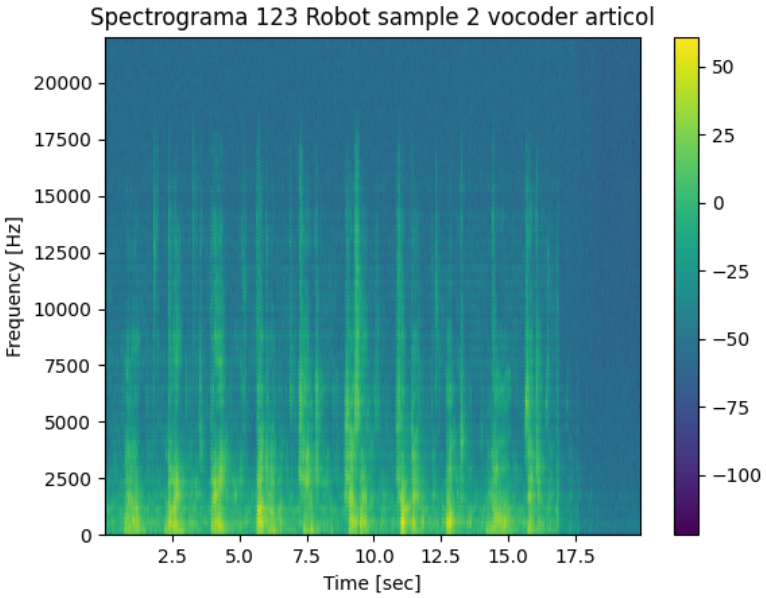
2. Spectograme:

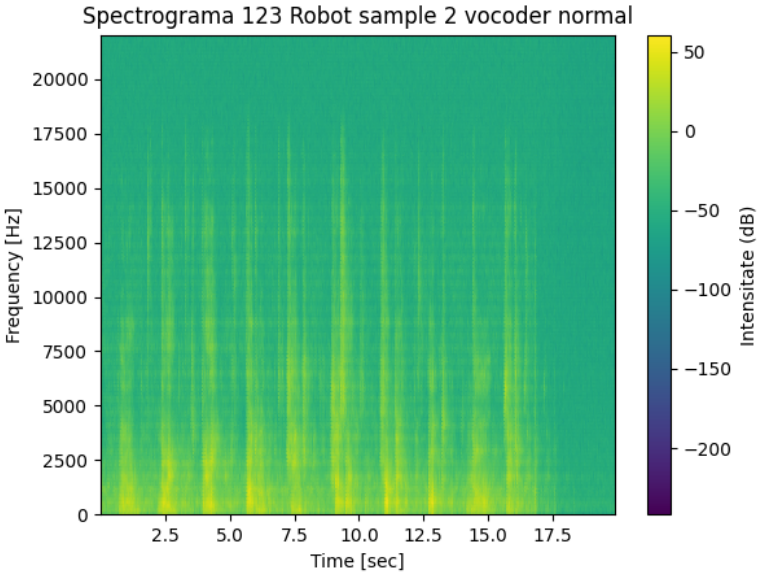


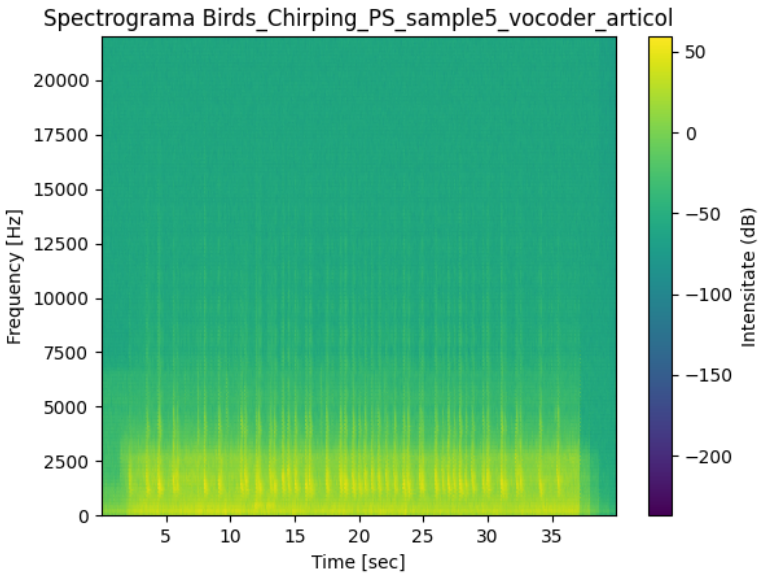


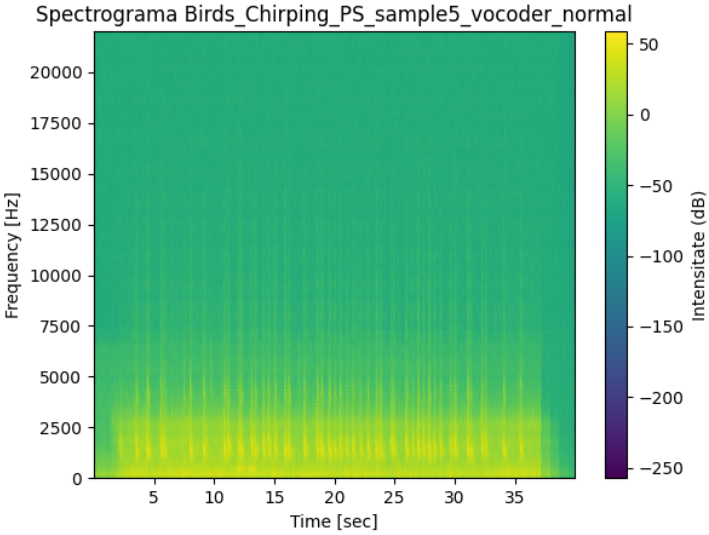










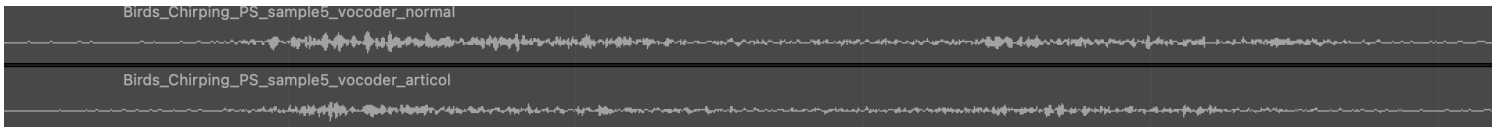


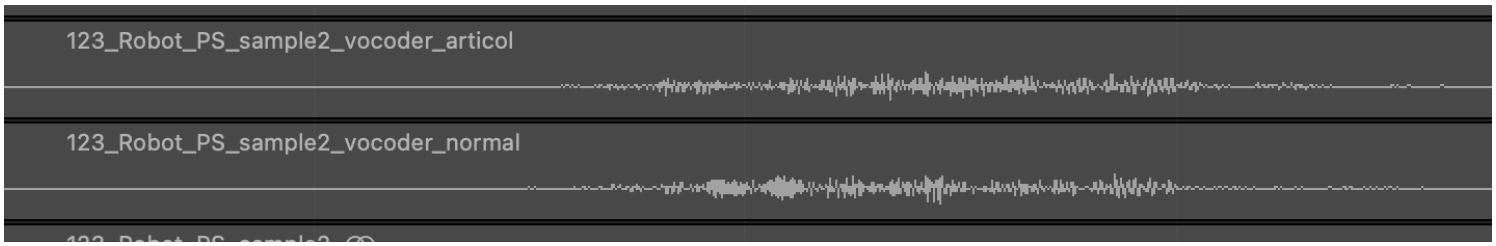
Explicatii Spectograme:

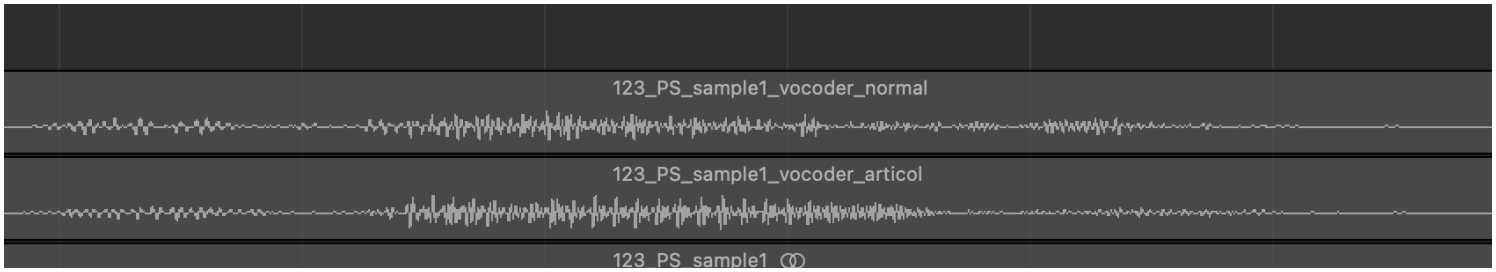
Pentru Sample 1 se poate observa faptul ca diferența este neglijabila, dar pentru Sample 2 in spectograma pentru varianta din articol observam clar o îmbunătățire pentru undele asociate numărătorii sunt clar vizibile fata de zgomot. Pentru Sample-ul C Major si Sample 5 nu putem utiliza spectograma drept un indicator foarte clar al perfomantei.

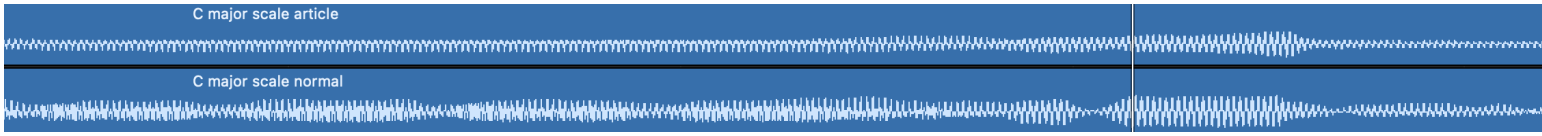
3. Waveforms:

Pentru aceasta evaluare am atașat capturi de ecran din GarageBand unde am importat rezultatele întrucât interfața îmi permite sa vad mult mai in detaliu fenomenul de phase unwrapping. Generez in același timp si waveform-uri in Python așa cum am menționat in secțiunea de implementare si acestea se ragesesc in arhiva. Pentru Sample 1 se observa clar avem mai mult detaliu, iar pentru Sample C Major observam ca avem un număr mai redus de distorsiuni prin uniformitatea mai ridicată a waveform-ului fata de versiunea normală.









Evaluare calitativă

Pentru a evalua calitativ rezultatele am importat in GarageBand fișierele obținute si am rulat succesiv fișierele audio si am obținut următoarele concluzii:

Sample 1 - Pentru acest sample rezultatul generat de versiunea din articol rezulta într-un sunet mai clar pentru cifre fata de versiunea normală, dar in situația in care exista un ecou in semnal varianta din articol va amplifica acel efect in lipsa altor prelucrări.

Sample 2 - Acest sample este cel mai dificil pentru ambele vocodere din cele 4 sample-uri cu vocea mea pentru ca zgomotul este mai ridicat după procesare, volumul pare mai scăzut pentru ambele, dar in ciuda acestor aspecte versiunea din articol este clar cea preferată. De mentionat este faptul ca înregistrarea inițială aplica filtrul de tip deeper voice asupra întregului input ceea ce generează mult zgomot de fundal si acest aspect s-a reflectat in scorul negativ al raportului SNR.

Sample 5 - Pentru acest sample diferența nu este una foarte sesizabilă.

Sample C Major - Acest sample prezintă cea mai mare diferența fata de varianta normală, sunetul este foarte clar, vocalele sunt izolate si se apropie in calitate de sample-ul original, iar acest aspect este posibil influențat de calitatea buna a sample-ului inițial care nu prezintă zgomot de fundal inițial precum sample-urile generate de mine.

Concluzii

Metoda propusa in articolul ales este o soluție foarte eficienta pentru implementarea unui phase vocoder pentru diferența in implementare nu este deloc mare fata de un phase vocoder, deci nu presupune un trade-off in acest sens si rezultatele calitative, dar si cantitative pentru anumite situații susțin asta.

Un aspect de mentionat este faptul ca metoda propusă rezolva doar problema de phase unwrapping si nu abordează problema de zgomot de fundal, astfel se explica faptul ca in cazul sample-urilor generate de mine care conțin si ecou, dar si zgomot de fundal rezultatele pentru evaluarea cantitativă nu au fost așa de ușor de observat inițial precum in cazul sample-ului C Major care are zgomot de fundal minim si astfel reprezintă un caz ideal de aplicare.