Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Time stretch és pitch shift algoritmusok vizsgálata

TDK-dolgozat

Konzulens:
dr. Sujbert László
docens

Készítette:
Galambos Róbert
V. vill.

Kivonat

Manapság digitális világban élünk, és minden területre betörtek a digitális eszközök. Nincs ez máshogy az audio- és stúdiótechnikával sem. A mai lemezlovasok között már szinte megszokottá vált a CD lejátszó, és egyre többen használnak számítógépet a fellépéseikhez. A hangmérnököknél is a vágás, a keverés, a masterelés átkerült számítógépre. Ez új lehetőségekhez nyitotta meg a kapukat: például time stretch algoritmusokat használnak a DJ eszközök a zenék keveréséhez, vagy pitch shift algoritmusokat a hangmérnökök az éneksávok korrigálásra.
A time stretch algoritmusok egy audio jel sebességét, időbeli lefutását befolyásolják (nyújtják, zsugorítják) úgy, hogy ezen jelek spektrális felépítése nem változik meg számottevően, azaz a gyorsított vagy lassított jel hangmagassága változatlan marad. A pitch shift algoritmusok ennek az ellenkezőjét teszik: az időbeli lefutás állandó marad, és a spektrális felépítés változik. Ez annyit jelent, hogy nem gyorsul és nem lassul a jel, de a hangmagassága megváltozik.
Többféle time stretch, és pitch shift algoritmust megvizsgáltam, implementáltam MATLAB-ban, hogy különböző szempontok szerint összehasonlíthassam őket. Ilyen szempontok voltak többek között a tranziens és periodikus jelek megváltozása, a különböző frekvenciájú komponensek más frekvencián ismétlődése, vagy időtartománybeli visszhangosodás. Majd ezen implementációkat megpróbáltam javítani, új módszereket, mint például a frekvenciasávonkénti feldolgozást vagy a wavelet transzformációt felhasználva. Ezek segítségével az idő- és frekvenciafelbontás jobban illeszkedik a jel tulajdonságaihoz, és jobban követi a különböző frekvenciájú komponensek változásait. Így a short time Fourier-transzformációnál hatékonyabban, többféle időfelbontást lehet használni a jel különböző frekvenciasávjaira. Munkám célja az volt, hogy minél nagyobb határok között lehessen állítani mind az időbeli, mind a frekvenciabeli változtatásokat úgy, hogy ez ne rontsa az audio jel minőségét. A módosításokkal sikerült az alap algoritmusoknál hatékonyabb módszereket kidolgozni.

Tartalomjegyzék

1. fejezet

Bevezetés

„ A zene egyszerre fejezi ki a szerelmet és az elválást. Egyszóval az életet fejezi ki. ”

John Minahan

A hallás az emberi érzékelés egyik bonyolultabb formája, és a környezetünkből érkező hang az egyik legnagyobb információforrásunk. A legjobb példa erre a zene, mely egyidős az emberiséggel, és oly sok érzés, hangulat, állapot, mondandó kifejezésére alkalmas.

Többek között én is ezért kezdtem el a zenével foglalkozni, zenészként kipróbálva önmagam. De mint leendő mérnököt, kicsit más szemszögből is érdekelnek az eszközök, algoritmusok. Nem elégszem meg annyival, hogy a zenéléshez ezek adottak, és csak használni kell őket, hanem megpróbálom kiszámolni, reprodukálni ezeket az eszközöket, és, ha tudom, a hibáikat kijavítani.

1.1. Sebességállítás a zenében

A zenének van tempója, egy adott üteme, mely szerint egy periodicitás figyelhető meg. Az alap probléma az, hogy ezt a tempót szeretnénk megváltoztatni, és másik tempó szerint meghallgatni az adott zenét.

A múltban, amikor még nem volt sem analóg, sem digitális hangrögzítés, csak úgy tudtak egy darabot, egy zenét meghallgatni, ha a zenekar, zenész eljátszotta. Ebben az esetben nem beszélhetünk konkrét sebességállításról, hisz a művészek a zenét egyszerűen más ütem szerint játszották el, mindig alkalmazkodva az éppen aktuális hallgatósághoz, környezetükhöz.

Később, amikor megjelentek az analóg hanghordozók, mint például a bakelitlemez, magnószalag, és megszületett a stúdiótechnika, a rádiós műsorszórás és társaik, akkor felvetődött a probléma, hogy a már felvett audio sávok sebességét kellene utólag megváltoztatni. Az indokok különbözőek voltak. A stúdiótechnikában az esetleg rosszul felvett sávokat próbálták korrigálni, a rádióban értékes műsoridőt megspórolni, hogy több zenét tudjanak lejátszani kevesebb idő alatt. Ezt megpróbálták analóg módon úgy megvalósítani, hogy a lejátszók motorjának fordulatszámát változtatták, így lassítva a lemez for-

gását, vagy a magnószalag haladását. A szórakoztatóiparban a lemezlovasok ezt a fajta sebességállítást még ma is használják, a zenék egymás után keverésére.

Ennek a módszernek az a problémája, hogy a sebességállítás következtében, a hangmagasságok is megváltoznak. A lassítás hatására mélyebb, míg a gyorsítás hatására magasabb lesz minden harmonikus, mint az a Fourier-transzformáció hasonlósági tételéből is látható:

$$\mathcal{F}\{x(t/\alpha)\} = |\alpha|\, X(\alpha\omega) \tag{1.1}$$

ezért korlátozottan használható csak. Például, ha egy félhangot elcsúsznak a felharmonikusok, akkor az a stúdiótechnika számára már elfogadhatatlan, hiszen nem lesz összhangban a zene többi részével.

Az 1960-as években Pierre Schaeffer előállt egy analóg megoldással, melyet Phonogenenek hívott. Ő egy szalagós lejátszót módosított úgy, hogy egy fej helyett többet épített bele, és ezek a fejek egy forgórészen voltak, és adott sebességgel forogtak [1]. Így ha a szalag sebességét állították, akkor a hosszát tudták változatni a jelnek, míg a forgás sebességét állítva a hangmagasságok változtak.

A digitális jelfeldolgozás megjelenésével ezt a problémát is megpróbálták megoldani az új technika adta lehetőséggel. A jelfeldolgozó processzorok és számítógépek kapacitása új kapukat nyitott meg, és lehetővé tette, hogy a gyors Fourier-transzformáció segítségével a jeleket a frekvenciatartományban új módszerekkel befolyásoljuk. Ezen lehetőségek segítségével vizsgálom a problémát.

1.2. A dolgozat felépítése

Többféle ismert algoritmusfajta van a sebességállításra, melyet a 2. fejezetben mutatok be. Többek között a Phonogene modelljén alapuló időtartománybeli szegmentálást (2.1 fejezet), ennek egy továbbfejlesztett fajtáját, mely több frekvenciasávon dolgozik különböző paraméterekkel (2.1.5 fejezet), majd rátérek a fázis vokódolás technikájára (2.2 fejezet).

A sebesség állítás után a 3. fejezetben azt vizsgálom, hogy lehet ezeket a sebességállító algoritmusokat arra használni, hogy mint Pierre Schaeffer Phonogene-jénél, ne jel a sebességét állítsák, hanem a hangmagasságot, lehetővé téve ezzel a hangmagasság eltolást, melyet stúdiótechnikában pitch korrekcióra használnak.

Szimulációs eredményeim bemutatásához elérhetővé tettem hanganyagokat, melyek alapján a tárgyalt jelenségeket meg is lehet figyelni.

A dolgozatot rövid összefoglaló és irodalomjegyzék zárja.

1.3. A hiba mérése

A sebességállító és a pitch korrekcióra szolgáló algoritmusok korrekt mérése nehéz, mivel a feldolgozott jelet nem tudjuk mivel összemérni, hisz a sebességállításnál az eredeti és a feldolgozott jel hossza más, így időtartományban hibát számolni nem lehet. Az

emberi fül túl szubjektív ahhoz, hogy mérőműszernek használjuk, ezért megpróbáltam egy módszert, egy eljárást találni arra, hogy miként tudom a hallott hibákat számokkal kifejezni.

1.3.1. Sebességállítás hibabecslése

Mivel a sebességállított jeleket az időtartományban nem lehet összehasonlítani, így megpróbáltam őket a frekvenciatartományban valamilyen módon összemérni. Az első elgondolásom a Fourier-transzformáció használata volt.

Frekvenciakomponensek összehasonlítása

Ez az eljárás az eredeti és a sebességállított jel frekvenciakomponenseit hasonlítja össze. Ezt úgy teszi, hogy az eredeti és a feldolgozott jelek közül az időben rövidebb végét kipótolja nullákkal, ezzel ugyanolyan hosszúvá téve a két jelet. Ezután mind a két jelet transzformálja, ezzel megkapva a két jel frekvenciatartománybeli képét. A két jel viszont nem azonos energiával rendelkezik, így, bár a transzformált párok hasonlítanak egymásra, a skálázás eltérése miatt nem lehet hibát számolni. A két transzformáltat normálni kell. Először megpróbáltam az amplitúdóspektrum maximumához normálni őket, de ez nem vezetett eredményre, ugyanis a sebességállítás esetlegesen egy frekvenciakomponenst nagyon kiemel, ami, bár zavaró a hallgató számára, de nem annyira, mint azt a hiba kiszámításával kapjuk. Ugyanis a kiugró komponens lesz az amplitúdókarakterisztika maximuma, és ehhez normálódik az egész transzformált. Így a frekvenciatengely minden pontján nagy eltérést, nagy hibát mutat a két transzformált, holott a hiba csak a kiugró komponens lenne. Ezért döntöttem úgy, hogy inkább az amplitúdó alatti területtel normálom a két transzformáltat.

$$e_{\text{freq}} = \sum_{j=1}^{N} \left(\frac{|\mathcal{FFT}\left\{\hat{x}[n]\right\}|}{\sum_{i=1}^{N} |\mathcal{FFT}\left\{\hat{x}[n]\right\}|} - \frac{|\mathcal{FFT}\left\{\hat{x}_{\text{proc}}[n]\right\}|}{\sum_{i=1}^{N} |\mathcal{FFT}\left\{\hat{x}_{\text{proc}}[n]\right\}|} \right)^2 \tag{1.2}$$

Az (1.2) képletben a hibaszámítás látható. A $x[n]$ és $x_{\text{proc}}[n]$ az eredeti és a feldolgozott jelet jelentik, és azért jelöltem őket kalappal, mert nullákkal lett kipótolva a rövidebbik vége, hogy $x[n]$ és $x_{\text{proc}}[n]$ ugyanolyan hosszú legyen, és ezáltal az $\mathcal{FFT}$ ugyanolyan hosszú vektort adjon vissza. N az $\hat{x}[n]$ és az $\hat{x}_{\text{proc}}[n]$ vektorok hosszát jelöli.

Az összehasonlítás így még nem teljes, hisz e_{freq} hiba függ még attól, hogy hány pontos $\mathcal{FFT}$-t használtunk. Ezért még le kell osztani az $\mathcal{FFT}$ pontszámával:

$$e'_{\text{freq}} = \frac{e_{\text{freq}}}{l_{\text{fftsize}}} \tag{1.3}$$

Hogy mit hallunk jónak és rossznak, az eltérhet ettől a mérőszámtól, de a szimulációk során, ez a szubjektív tapasztalattal jól egyező eredményt adott.

Az eljárás hibája, hogy nem veszi figyelembe a frekvenciakomponenseknek időbeli változását, csak arról ad információt, hogy az egyes komponensek milyen súllyal szerepelnek a jelben.

Időben változó frekvenciakomponensek összehasonlítása

Ha számunkra az is fontos, hogy melyik időpillanatban, hogyan változik a frekvenciakomponsek értéke, akkor egy másik hibaszámítást kell alkalmazni. Ha vesszük az eredeti és feldolgozott jel spektrogramját (1.4), ami a short-time Fourier transzformáció négyzetével közelíthető, akkor megkapjuk a jel időben változó frekvenciafelbontását. Mivel az eredeti és a feldolgozott jel nem ugyanolyan hosszú, így az időben rövidebb jelnek a spektorgramját interpoláljuk az időtengely mentén úgy, hogy azonos számú egyenletes távolságú osztás keletkezzen. Így a két spektrogramban az összetartozó tranziens események „fedésbe" kerülnek, hisz ezzel az időben rövidebb jel spektrogramját hozzá „nyújtottuk" az időben hosszabbéhoz. Ezeket a jeleket ismét kalappal jelöltem.

$$\begin{aligned} \mathbf{S} &= \mathcal{STFT}\left\{x[n]\right\}^2 \\ \mathbf{S}_{\text{proc}} &= \mathcal{STFT}\left\{x_{\text{proc}}[n]\right\}^2 \end{aligned} \tag{1.4}$$

A két spektrogramot ($\hat{\mathbf{S}}$ és $\hat{\mathbf{S}}_{\text{proc}}$) ismét súlyozom az összegükkel, majd négyzetes hibát számolok a kettő különbségéből:

$$e_{\text{spec}} = \sum_{k=1}^{N}\sum_{l=1}^{M}\left(\frac{\hat{\mathbf{S}}}{\sum_{i=1}^{N}\sum_{j=1}^{M}\hat{\mathbf{S}}_{ij}} - \frac{\hat{\mathbf{S}}_{\text{proc}}}{\sum_{i=1}^{N}\sum_{j=1}^{M}\hat{\mathbf{S}}_{\text{proc}ij}}\right)^2 \tag{1.5}$$

ahol N és M az $\hat{\mathbf{S}}$ és $\hat{\mathbf{S}}_{\text{proc}}$ mátrix méreteit jelölik. Ezt a hibát is normálni kell, méghozzá a spektogram méretével, hogy függetlenné tegyük tőle:

$$e'_{\text{spec}} = \frac{e_{\text{spec}}}{l_{\text{spec}} \cdot w_{\text{spec}}} \tag{1.6}$$

ahol l_{spec} és w_{spec} az $\hat{\mathbf{S}}$ és $\hat{\mathbf{S}}_{\text{proc}}$ méreteit jelölik. Így kaptunk egy másik mérési eljárást, mely valamilyen tulajdonságok szerint összehasonlítja a két jelet. A szimulációk során ez az eljárás is akkor adott nagyobb hibát, ha hallásra is rosszabb volt az eredmény.

Mindkét hibaszámítási módszer MATLAB kódja megtalálható a függelékben.

1.3.2. Hangmagasság eltolásának hibabecslése

Mivel az általam vizsgált hangmagasság eltoló algoritmusok a sebességállító algoritmusokra épülnek, ezért kölön hibamérő eljárást nem használtam, hisz, mint az később a 3 fejezetben tárgyalásra kerül, a hangmagasság eltolásához, egy megfelelően sebességállított jelet újramintavételezünk. Az újramintavéltelezés hibája számolható, és nem kíván mérési eljárást, de a fent említett időben változó frekvenciakomponenseket összehasonlító eljárás a másik irányba, tehát a frekvenciatengely mentén nem ekvidisztánsan interpolálva (hisz nem egyszerű frekvencia eltolás történik, hanem más hangmagasságra tolás) működhet.

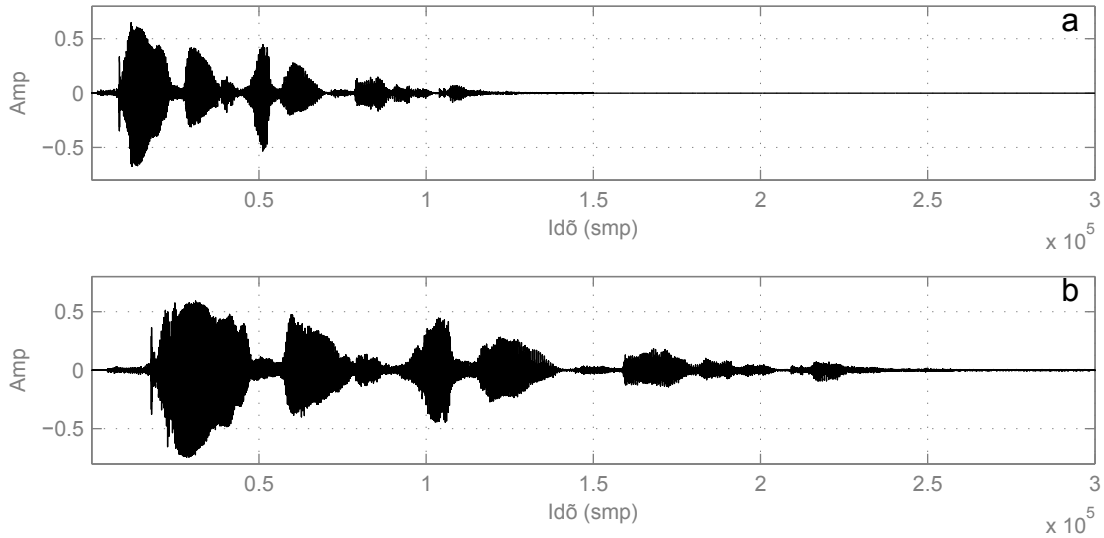
2. fejezet

Time Stretch

„ Az időnek egyetlen oka van: minden nem történhet egyszerre. ”

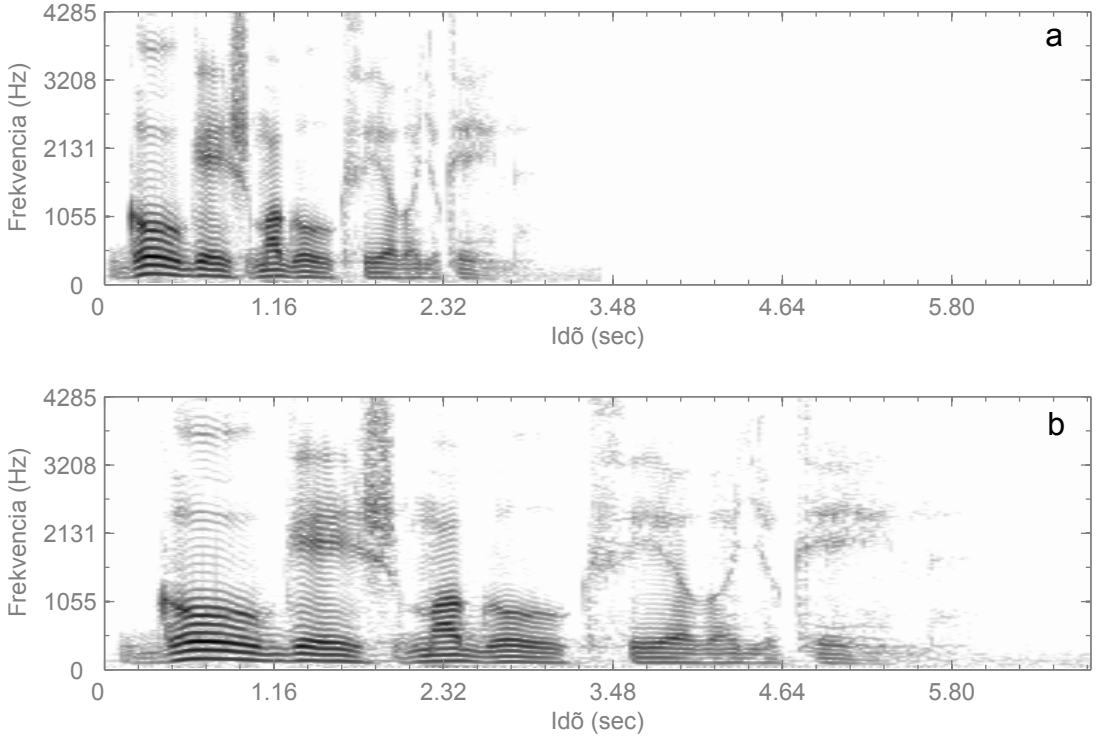
Albert Einstein

Ebben a fejezetben a sebességállító (vagy angol nevén time stretcht) algoritmusokat mutatom be. Ezek az algoritmusok időnyújtást végeznek a jelen, tehát úgy változtatják meg a jel hosszát, hogy annak frekvenciakomponensei nem változnak.



2.1. ábra. Eredeti (a), és time stretchelt jel (b) az időtartományban.

A 2.1. ábrán látható két időfüggvény, az eredeti jel (a) és egy kétszeres hosszra time stretchelt jel (b). Alakjukra nagyon hasonlóak, csak az időbeli hosszuk más.

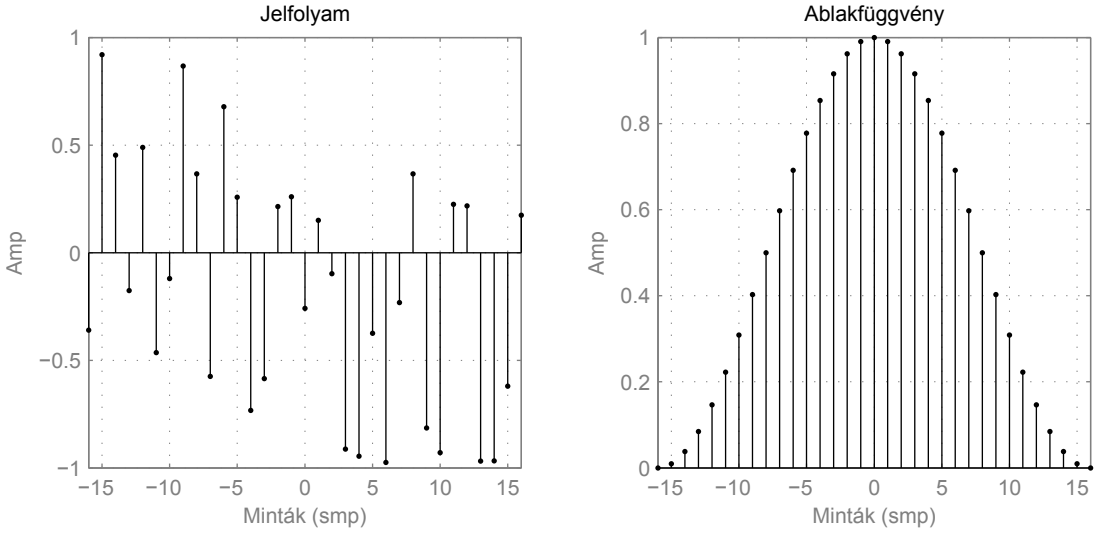


2.2. ábra. Eredeti (a), és time stretchelt jel (b) a frekvenciatartományban.

A 2.2. ábrán látható két spektrogram, az eredeti jel (a) és egy kétszeres hosszra time stretchelt jel (b). A két jel frekvenciakomponensei ugyanott vannak, csak időben máskor kezdődnek, és máskor fejeződnek be.

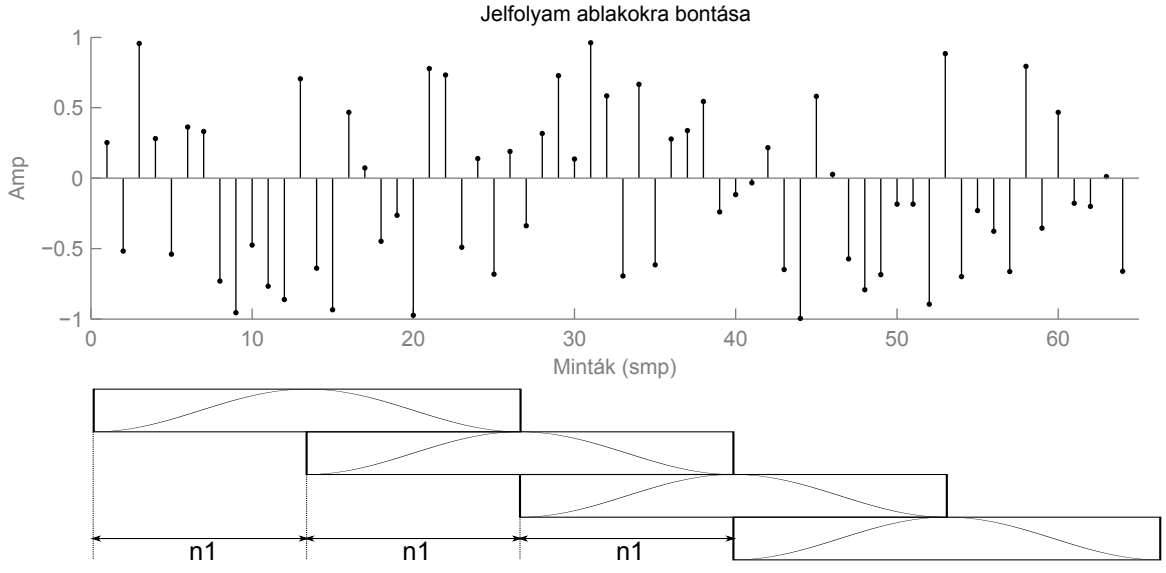
2.1. Időtartománybeli szegmentálás

Az első algoritmus a time stretch algoritmusok közül az időtartománybeli szegmentálás, vagy angol nevén overlap and add (OLA) (és egyik algoritmusverziója a syncronized overlap and add (SOLA) [1]). Az algortimus a jelet ablakokra bontja szét, majd ezeket az ablakokat csúsztatja össze vagy szét, attól függően, hogy nyújtani akarjuk az időfüggvényt, vagy összenyomni. Mint látható, az algoritmus komplexitása kicsi, így könnyen implementálható real-time alkalmazásokban. Talán ezért is van több változata. Minden változat egy-egy speciális felhasználási területre alkalmas. Például a pitch-synchronous overlap and add (PSOLA) [1], vagy a waveform similarity overlap and add (WSOLA) [2], melyek monofonikus jelre működnek, mert kihasználják annak perirriódikusságát, és a periódusokhoz illesztik az ablakokat. De van olyan változat is, amely az érthetőséget helyezi középpontba [3].



2.3. ábra. A jelfolyam, és az abalkfüggvény.

A 2.3. ábrán látható az $x[n]$ digitális jelfolyam, amit fel akarunk dolgozni, valamint a $w[n]$ ablakfüggvény. Az ablakozáshoz meghatározható egy egy n_1 távolság, mely a két ablak kezdete között megadott távolság mintában. Ha ez a távolság kisebb, mint a $w[n]$ ablaksorozat hossza, akkor a két ablak átlapolódik, ha $w[n]$ hosszának felénél is kisebb, akkor többszörösen átlapolódnak az ablakok. Ha $x[n]$, $w[n]$, és n_1 adott, akkor a jelfolyamot a (2.1) szerint lehet $y_k[n]$ ablakok által kivágott darabokra bontani.

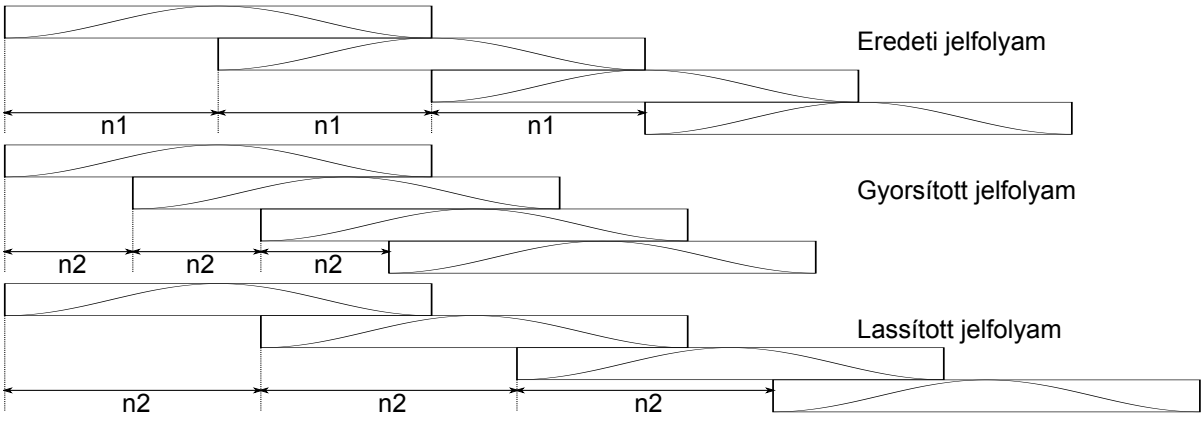


2.4. ábra. A jelfolyam, ablakokra bontása.

A 2.4. ábrán látható, ahogyan az $x[n]$ sorozatot felbontjuk átlapolt ablakokra, ahol n_1 a távolság az ablakok között.

$$\begin{aligned} y_0 &= x[n] \cdot w[n] \\ y_1 &= x[n] \cdot w[n - n_1] \\ y_2 &= x[n] \cdot w[n - 2n_1] \\ ... &= ... \\ y_k &= x[n] \cdot w[n - kn_1] \end{aligned} \tag{2.1}$$

Az $y_k[n]$-kat ezután attól függően, hogy milyen irányban változtatjuk meg a jel sebességét, össze illetve szétcsúsztatjuk, ami azt jelenti, hogy $y_k[n]$-t a megfelelő mintaszámmal eltoljuk, így változtatva meg az eredeti jel hosszát.



2.5. ábra. A jelfolyam gyorsítása, lassítása.

A 2.5. ábrán láthatóak az eredeti jelfolyam ablakai, és a belőle kinyert gyorsított, illetve lassított jel ablakai. Ha n_2 jelöli az elcsúsztatott ablakok kezdete közötti távolságot, akkor az ablakok elcsúsztatása felírható a következőképpen:

$$\begin{aligned} y_k[n + (n_1 - n_2)k] &= x[n + (n_1 - n_2)k] \cdot w[n - kn_1 + (n_1 - n_2)k] \\ &= x[n + (n_1 - n_2)k] \cdot w[n - n_2 k] \end{aligned} \tag{2.2}$$

A kimeneti jelet úgy kapjuk meg, hogy a kivágott elcsúsztatott ablakokat összegezzük a következők szerint:

$$\begin{aligned} z[n] &= \sum_{k=-\infty}^{\infty} y_k[n + (n_1 - n_2)k] \\ &= \sum_{k=-\infty}^{\infty} x[n + (n_1 - n_2)k] \cdot w[n - n_2 k] \end{aligned} \tag{2.3}$$

ahol $z[n]$ a kimeneti jel. Mivel jelfolyamról beszélünk, így nincs eleje és vége, ezért a k index $\in [-\infty, \infty]$ tartománynak. A valóságban természetesen véges hosszú jelekre alkalmazzuk a képletet. Ilyenkor a jelek kezdeténél és végénél a $w[n]$ ablakfüggénnyel való szorzás miatt a jelalak torzul, egy „felhangosodó” rész keletkezik az elején, és egy „elhalkuló” rész a

végén. Ha arra ügyelünk, hogy ne legyen ezeken a helyeken információtartalom, amit úgy érünk el, hogy a jel elé és mögé nullákat szúrunk be, akkor ezt a hatást kiküszöbölhetjük. Ezenkívül a $z[n]$ jelet normálni kell.

A legegyszerűbb megoldás erre, ha a jel mellett számolunk egy $z^{'}[n]$-t a (2.3), ahol feltételezzük, hogy a bemeneti jel konstans egy. Ekkor valójában az ablakfüggvényeket adjuk össze. Ha ezzel a $z^{'}[n]$-el leosztjuk az eredeti $z[n]$ kimeneti jelünket, akkor a jelszintet korrigáltuk. Sőt így még az a jelszintváltozási hiba sem lép fel ami akkor keletkezne, ha az átlapolt ablakok összege nem egy lenne.

2.1.1. Tempóingadozás

Az első hibajalenség amit tapasztaltam, és nem csak a szimulációimban, hanem nagy valószínűséggel időszegmentálás alapú piaci szoftverekben is, az a tempó ingadozás. Ez úgy érzékelhető, hogy a tempó rövid ideig eltér a megadottól.

A fent leírt eljárás bizotsítja, hogy az ablakok kezedete pont oda kerüljön, ahova a sebességállítás miatt kerülnie kell. A probléma gyökere az, hogy egy ablakon belül a sebesség megegyezik az eredeti jelével, hiszen pont ezért nem változik a hangmagasság. Ebből következik, hogy minél nagyobb ablakokat használunk, annál nagyobb lesz az ingadozás. A legnagyobb eltérés az ablak végénél jelentkezhet, mert ekkorra már egy ablaknyi ideje az eredeti sebessége van a jelnek. A hiba a következőképpen felülről becsülhető:

$$e_{\text{tempo}} = (1 - r_{\text{speed}}) \cdot l_{\text{wnd}} \tag{2.4}$$

ahol e_{tempo} egyenlő a mintákban megadott tempóingadozással, r_{speed} jelöli az eredeti és a feldolgozott jelek sebességarányát, l_{wnd} pedig az ablak hosszát.

A tempóingadozás olyan alkalmazásokban kritikus, ahol a jelfolyamok sebességét szinkronizálni kell, hogy össze tudjuk őket keverni. Ilyen alkalmazás lehet például egy DJ keverőprogram, vagy CD lejátszó. Ha ilyen körenyezetben nézzük meg a hibát, akkor át tudjuk számolni BPM (Beat Per Minute) ingadozásba.

Ha egy négy negyedes ütemetü zenét vizsgálunk, akkor a taktus és dobok egybeesnek. Ez általában igaz a mai elektronikus zenék nagy részére. Ilyenkor legrosszabb esetben két dob között e_{tempo} mintányi hiba lép fel. Ebből következik, hogy a BMP ingadozás a következőképpen számolható:

$$\Delta BPM = \frac{60 \cdot f_{\text{s}}}{\frac{60 \cdot f_{\text{s}}}{BPM} + e_{\text{tempo}}} - BPM \tag{2.5}$$

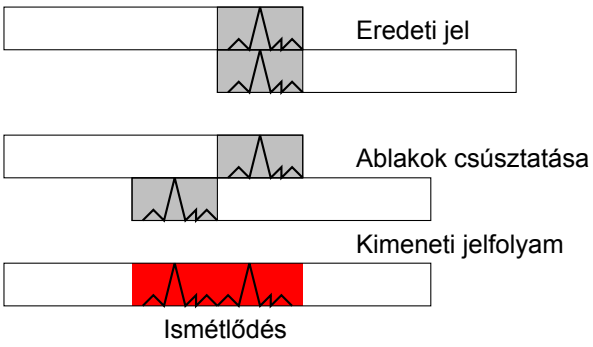
A (2.5) összefüggést használva $f_{\text{s}} = 44100$ Hz, $BPM = 130$, $l_{\text{wnd}} = 2048$ minta, és $r_{\text{speed}} = 1.2$, akkor $e_{\text{tempo}} = 409.6$, és $\Delta BPM = -2.56$. Azaz viszonylag nagy lassítás hatására jelentős eltérés lehet egy rövid időre a tempóban. A becslésünket pontosíthatjuk, hisz az l_{wnd} nem minden esetben az ablak hosszával egyenlő. Ha két ablak átlapolódik, akkor csak az átlapolódásmentes részt kell nézni a pillanatnyi, és a következő ablakkal. Ez alapján felírható (2.6).

$$\begin{aligned} l_{\text{wnd}} &= n_1 \\ e_{\text{tempo}} &= (1 - r_{\text{speed}}) \cdot n_1 \end{aligned} \tag{2.6}$$

Természetesn az átlapolódás növelése nem megoldás, és más problémákat vet fel.

2.1.2. Átlapolódó ablakok

Ha egy ablak átlapolódik a következővel, akkor az átlapolódott részen az információ duplán lesz jelen, és amikor a sebességállítás miatt az ablakokat elcsúsztatjuk, a két ablak azonos információjú része nem lesz fedésben. Így azon a részen ismétlődés tapasztalható. Ez akkor jelentkezik a legjobban, ha a jelfolyam tranziens része belefér az átlapolódásba, és ott a jelfolyam nem tekinthető stacionáriusnak. Ezt szemlélteti a 2.6. ábra.



2.6. ábra. Az ismétlődés szemléltetése

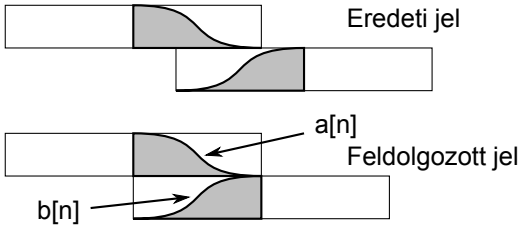
Ha az ismétlődés elkerülése érdekében úgy választanánk meg az ablakokat, hogy ne legyen átlapolódás, akkor kénytelenek lennénk olyan $w[n]$ ablakfüggvényt használni, amely, mivel nem kellően „sima”, így lényegesen megváltoztatná a jel spektrális szerkezetét. Valamint ha nem lapolódnak át az ablakok, akkor nem lehet lassítani, hisz az ablakokat már jobban nem lehet „széthúzni”.

2.1.3. Fésűszűrő hatás

Az átlapolódó ablakok okozta tranziensjel kétszererőzdése is felfogható fésűszűrőnek, hisz a jel időben eltolt önmagával adódott össze, de az előbb a tranziens részét vizsgáltuk a problémának. Ha n_1-et úgy választjuk meg, hogy az ablakok között ne legyen átlapolódás, és n_2-t pedig úgy, hogy gyorsuljon a jel, akkor mivel $n_2 < n_1$, $n_1 - n_2$ minta hosszú rész a feldolgozott jelen átlapolódást szenved, és egy időfüggő fésűszűrő alakul ki, melynek a paraméterei az egymást követő ablakfüggvényektől függenek. Ha n_1-et nem úgy választjuk meg, hogy ne legyen átlapolódás, a fent említett hatás akkor is jelentkezik. Ha feltételezzük, hogy egy szerre csak két ablak lapolódik át, akkor felírható a fésüszűrő egyenlete:

$$y[n] = a[n] \cdot x[n] + b[n] \cdot x[n + (n_1 - n_2)] \tag{2.7}$$

ahol $a[n]$ a pillanatnyi ablakfüggvény azon a része, mely az átlapolódásnál van, és $b[n]$ a következő ablakfüggvény átlapolódásnál lévő része. Hasonló egyenlet felírható több ablak átlapolódására is, csak akkor nagyobb fokszámú lesz a szűrő, és a szűrési hatás is jobban jelentkezik.



2.7. ábra. A fésűszűrő kialakulása

A 2.7. ábrán látható, a fésűszűrő kialakulása, az ablakok összecsúsztatása miatt. Ha $a[n]$ és $b[n]$ „lassan változó" paraméterek, akkor a fésűszűrő felírható a Z-transzformáltjával, és $a[n]$, valamint $b[n]$ felfogható mint időben változó paraméterek. (2.8)

$$\begin{aligned} Y(z) &= a[n] \cdot X(z) + b[n] \cdot X(z) z^{n_1 - n_2} \\ \frac{Y(z)}{X(z)} &= a[n] + b[n] \cdot z^{n_1 - n_2} \\ \frac{Y(\omega)}{X(\omega)} &= a[n] + b[n] \cdot \mathrm{e}^{\mathrm{j}\omega(n_1 - n_2)} \\ \left| \frac{Y(\omega)}{X(\omega)} \right| &= \sqrt{b[n]^2 + 2a[n]b[n]\cos((n_2 - n_1)\omega) + a[n]^2} \end{aligned} \tag{2.8}$$

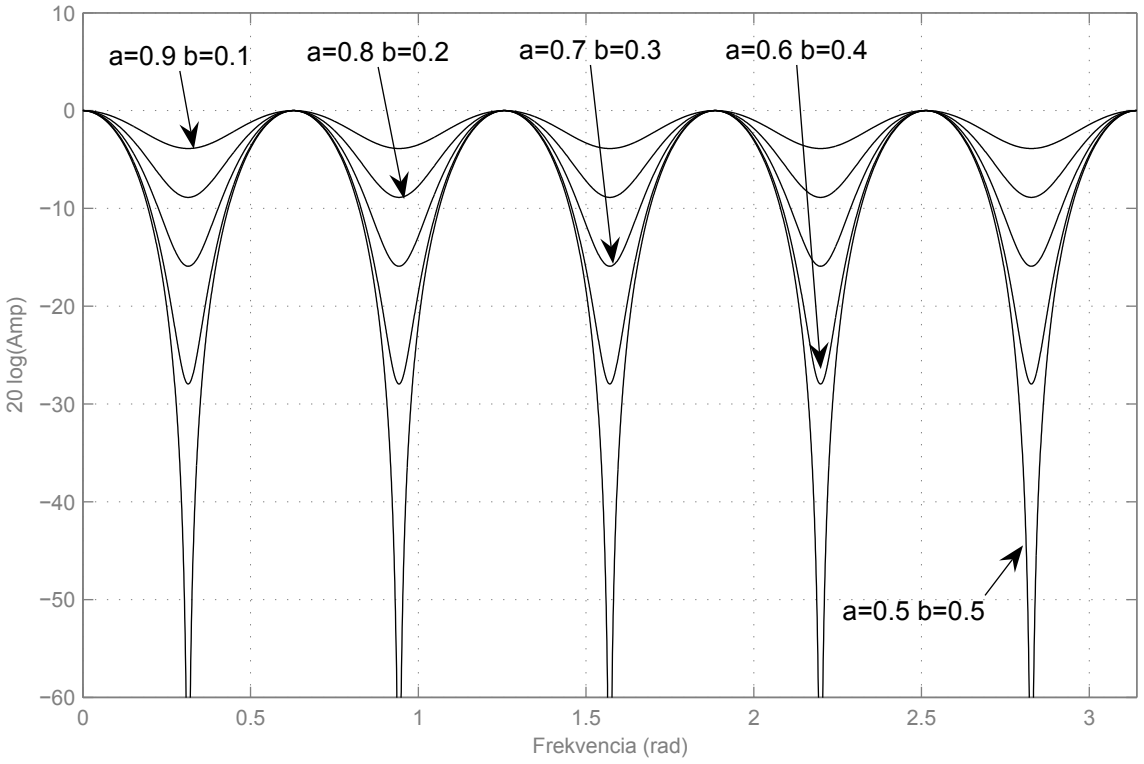
Különböző $a[n]$ és $b[n]$ értékek mellett felrajzolható a fésűszűrő amplitúdókarakterisztikája. n_2-t és n_1-et úgy választottam meg, hogy a 2.8. ábrán jól láthatók legyenek a leszívások, jelen esetben n_2 és n_1 kölünbsége 10 minta, ami tíz pólussal rendelkező szűrőt eredményez.

A 2.8. ábrán látszik a fésűszűrő amplitúdókarakterisztikája. Ha a fejezet elején említett normalizálást elvégezzük a feldolgozott jelen, tehát $z[n]$ helyett már a $z'[n]$-t vizsgáljuk, akkor ebből következik, hogy az átlapolódásnál az ablakok egy adott pillanatban vett értékeinek összege pontosan egyet ad. Ezért a 2.8. ábrán $a[n]$ és $b[n]$ is úgy van megválasztva, hogy összegük egy legyen. Ha $a[n]$ és $b[n]$ értékeit felcseréljük, akkor ugyanazt az amplitúdókarakterisztikát kapjuk.

Ha a 2.7. ábrát is figyelembe vesszük, akkor láthatjuk, hogy az átmenet úgy valósul meg, hogy $a[n]$ értéke folymatosan csökken, míg $b[n]$ értéke nő. Ebből következik, hogy a fésűszűrő leszívásai először nőnek, majd úgyra csökkenek.

2.1.4. AM moduláció

Az előző részben taglalt fésűszűrőnek van egy másik következménye is. A szűrő hatása ugyanis csak az átlapolódó ablakrészekre terjed ki, és az átlapolódásmentes részek érin-

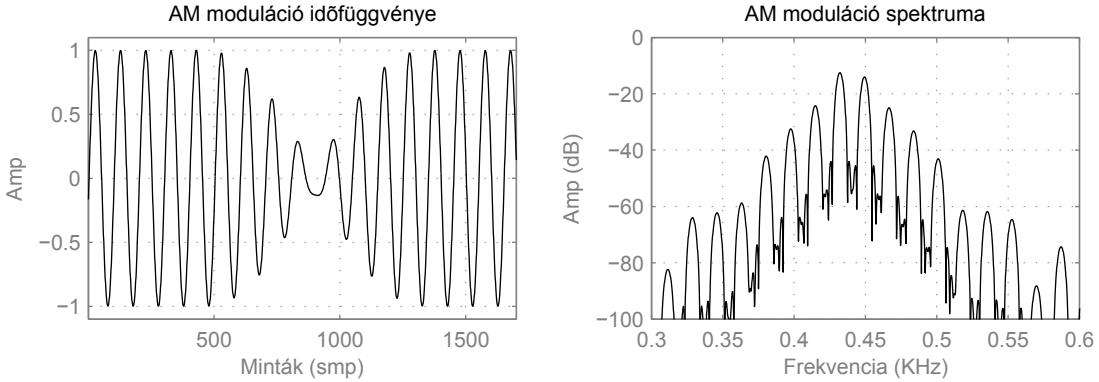


2.8. ábra. A fésűszűrő hatás ábrázolása

tetlenek maradnak. Ha egy másik szemszögből vizsgáljuk az algoritmust, és egy szinuszos tesztjelet dolgozunk fel vele, akkor azt tapasztalni, hogy a szinusz változatlan marad az ablakok nem átlapolódó részénél, viszont az általpolódásoknál a fésűszűrő leszívásai miatt csökkenhet a hangereje, attól függően, hogy milyen frekvencián van maga a szinusz. Ebből az következik, hogy a szinusz időben modulálódik, és az így „ráült" amplitúdómoduláció megváltoztatja a frekvenciakomponenseket. A SOLA algoritmus jobban teljesít az OLE alpváltozatnál, mert a SOLA algoritmusban az átlapolódó részek $a[n]$ és $b[n]$ paramétereit, tehát az aktuális ablak „elhalkulását", és a következő ablak „felhangosodását", mindig úgy választjuk meg, hogy az átlapolt részen megkeressük a legnagyobb keresztkorrelációs értéket, és eszerint állítjuk be a két paramétert. Ez javít a problémán, de teljesen nem szüntetni meg.

A 2.9. ábrán egy ablakátmenet látható, és a jobb oldalon a hozzá tartozó amplitúdó spektrum. A tesztjel egy 440 Hz-es szinusz, az algoritmus 4096 mintás Hanning ablakkal dolgozott, $n_1 = 2048$ és $n_2 = 3686$ paraméterek mellett egy 1.8 szoros lassítást végrehajtva.

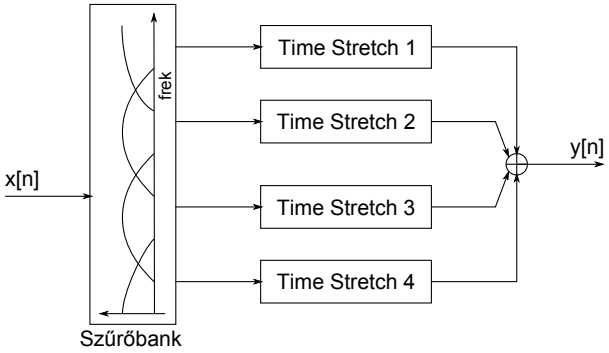
Az amplitúdómoduláció frekvenciája attól függ, mekkora ablakokat használunk. Ha nagyobbak az ablakok, akkor a moduláció frekvenciája is kisebb, ezáltal a spektrum torzulása is. Függ még attól, hogy hány ablak lapolódik át, hisz ilyenkor a fésűszűrő hatás is megváltozik.



2.9. ábra. AM moduláció hatása

2.1.5. Több sávos szegmentálás

Felmerülhet bennünk, hogy a jelfolyamnak magas és alacsony frekvenciás komponensei más sebességgel változnak, így feldolgozhatnánk ezeket a sávokat külön-külön, sávra optimalizált paramétereket használva. Erre egy megoldás, ha a bejövő jelet egy szűrőbank segítségével frekvenciasávokra bontjuk, majd ezeken a frekvenciasávokon végrehajtjuk a sávnak megfelelő paraméterekkel a time stretch algoritmust, és végül ezeket a sávokat ismét egyesítjük.



2.10. ábra. Több sávos szegmentálás

A 2.10. ábrán látható a felbontásnak, feldolgozásnak, és az összegzésnek a blokkvázlata.

Ha összehasonlítjuk a sávokat, a magas frekvenciasávban rövidebb ideig tartanak a tranziens jelek, így ott kisebb ablakokat alkamazhatunk, míg a mély frekvenciasávban nagyobb ablakokat kell használnunk, hogy az AM moduláció ne jelentkezzen annyira érzékelhetően. Az eljárás hátránya, hogy a tempóingadozás minden sávban máshogy jelentkezik, és egy több frekvenciasávot átölelő jelet időben „delokalizál", „szétken" , mivel minden sávban más pillanatnyi értékkel tér el a kívánt tempótól.

Több sávos feldolgozásként kipróbáltam egy 3 sávra bontást FIR szűrőkkel, valamint a diszkrét wavelet-transzformáció „oktávszűrőbankját" [4] [5]. Itt 8-16 sávra is felbontottam a jelet. Az eredmények a két feldolgozás között körülbelül hasonlóak voltak, a feldolgozott

jel minősége némileg javult. A wavelet-es szűrőbank esetében a feldolgozás gyorsabb volt.

2.1.6. Összefoglaló

Az ablak paramétereinek megválasztásánál kompromisszumokra kényszerülünk, hisz az egyes hatások teljes megszüntetése nem lehetséges. Az ablak méretét úgy kell megválasztani, hogy az AM moduláció ne legyen zavaró a jelre nézve. Tehát az ablak hosszát elég nagyra kell választani, de túl nagy sem lehet, hisz akkor a tempó kezd el ingadozni. Az átlapolódó tartománynak elég kicsinek kell lenni, hogy a fésűszűrés időtartamát valamint az információ duplázódást csökkentsük, de túl kicsi sem lehet, mert akkor az ablakfüggvény lesz túl éles változású, és fogja befolyásolni a jelfolyamot.

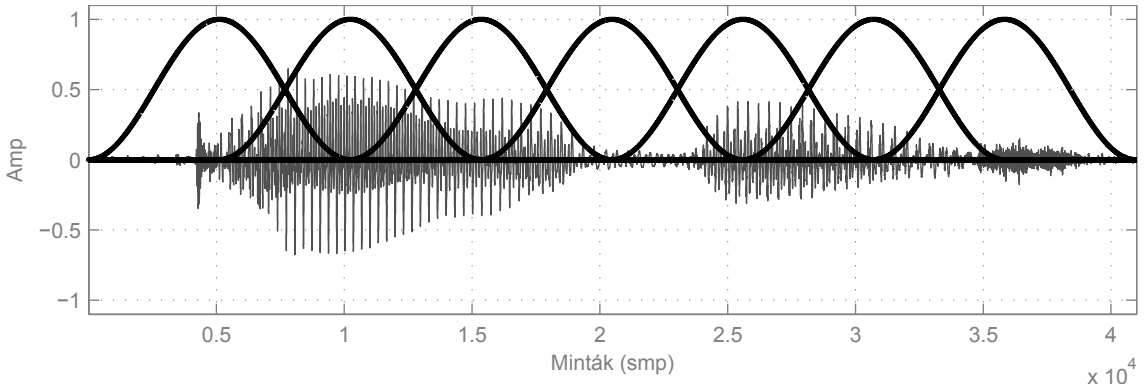
A tipikusan megválasztott ablakméretek magas frekvenciasávokra $1000 - 2000$ minta körül mély frekvenciasávokra $5000 - 10000$ minta körül mozognak $f_s = 44100$ Hz mintavételi frekvencia mellett. Ablakfüggvényként általában a Hanning ablakot használtam, de kipróbáltam egy ötödfokú polinom ablakot is, ahol az ablak deriváltja 25% és 75% -ánál nem 1 volt, mint a Hanning ablaknál, hanem 2, csökkentve ezzel a fésűszűrő hatást olyan módon, hogy $a[n]$ és $b[n]$ paraméterek még rövidebb ideig tartózkodnak 0.5 körül. Ezen kívül Hanning szélű ablakokat is használtam, kis átlapolódású ablakok vizsgálatakor.

2.2. Fázis vokódolás

Egy másik algoritmus a time stretch megvalósítására a fázis vokódolás [1]. Ez a módszer a jelet nem az időtartományban próbálja befolyásolni, hanem a frekvenciatartományban. Az algoritmus lényege, hogy STFT (Short-time Fourier Transform) segítésével a jel időbeli szegmenseiből kinyerjük a különböző frekvenciákhoz tartozó fázis- és amplitúdóértékeket, majd ezen információkkal szintetizáljuk a jelet valamilyen módon, például egy generátorbank használatával. Az analízis és a szintézis más órajelre történik, és így el tudjuk nyújtani, vagy össze tudjuk nyomni a jelfolyamot, tehát meg tudjuk változtani a sebességét.

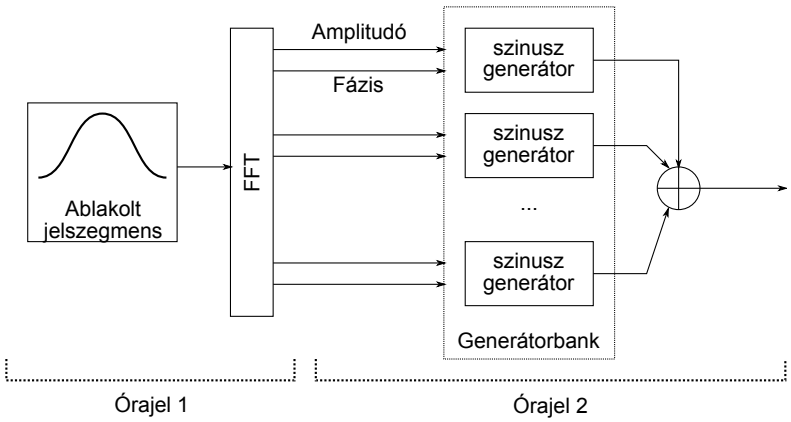
Ennek az algoritmusnak is van több változata, melyek különböző kisebb nagyobb komplexitású javításokat tartalmaznak. Például van tranziensfelismeréses kiegészítése, mely a jelben felismeri a tranzienseket, és nem változtatja meg őket [6]. Vagy a több frekvenciasáv egymásközti fáziskoherenciájának megtartására törekszik [7]. Frekvenciasávok egymásközti fáziskoherenciája azt jelenti, hogy jelben egy adott időpontban a frekvenciasávok mindengyike egy adott fázistolással rendelkezik. Ha sebességállítást végzünk, akkor a fázis vokóder algoritmus egy sávon belül megtarja a fáziskoherenciát, nem lesz fázisugrás, és követi a $\Delta\varphi$ fázisváltozást is, de több sáv egymáshoz képesti fázisváltozásait már nem.

A 2.11. ábrán látható a jelfolyam ablakokra bontása. Az ablakfüggvényt úgy kell megválasztani, hogy ne torzítsa el a jel frekvenciakomponenseit. Ehhez használható például Hanning ablak, vagy más STFT-hez használt ablak. Az átlapolódást ugyanúgy változtattam, mint az időszegmens alapú feldolgozásnál, és általában 80-90%-os átlapolódást



2.11. ábra. A jelfolyam ablakokra bontása

használtam, a pontosabb fázistolás követéséhez.



2.12. ábra. Analízis és szintézis

A 2.12. ábrán látható az analízis és szintézis blokkdiagramja. Ez alapján felírható:

$$\mathbf{G}_k = \mathcal{FFT}\left\{x[n] \cdot w[n - n_1 k]\right\} \tag{2.9}$$

ahol $\mathbf{G}_k$ a k-dik ablakhoz tartozó $\mathcal{FFT}$ vektor. Ebből a vektorból ki lehet számolni a generátorokhoz szükséges amplitúdó- és fázisinformációkat. Mivel az analízis és a szintézis más órajelen működik, így az egymás utáni amplitúdó- és fázisértékek között interpolálni kell, hogy pontosabb értéket kapjunk.

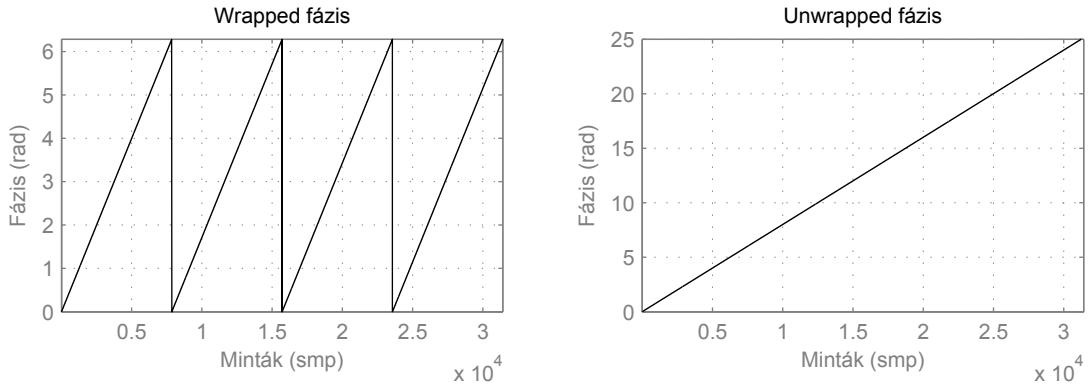
2.2.1. Amplitúdó interpolálása

Az amplitúdó interpolálása egyszerű, hiszen az csak az aktuálisan kiszámolt és a régi amplitúdó között kell valamilyen módon, például lineárisan interpolálni.(2.10)

$$A_k = \frac{A_{\text{now}} - A_{\text{old}}}{k_{\max}} \cdot k \tag{2.10}$$

2.2.2. Fázis interpolálása

A fázis interpolációja már egy kicsit komplikáltabb, hiszen figyelembe kell venni, hogy a szintézisnek eltérő órajele van az analízishez képest, emiatt a szintézis csak a fázis változást tudja reprodukálni, de az abszolút fázist nem tudjuk követni. Ezért van az említett fázis inkoherencia a frekvenciasávok között. Ezenkívül még figyelni kell arra, hogy a fázis értékeit átlapolódáshelyesen kell kezelni az összeadásoknál és kivonásoknál. Ez azt jelenti, hogy a fázist „unwrappolni kell", hogy biztosítsuk a helyes eredményeket. A 2.13. ábrán látható egy fázis-időfüggvény, melyet unwrappoltam. (A szimulációs kódomnál ezt úgy oldottam meg, hogy biztosítottam a kivonásnál azt, hogy nagyobb értékből vonjak ki kisebbet, és moduló képzést használva a megfelelő tartományba került utána az eredmény.)

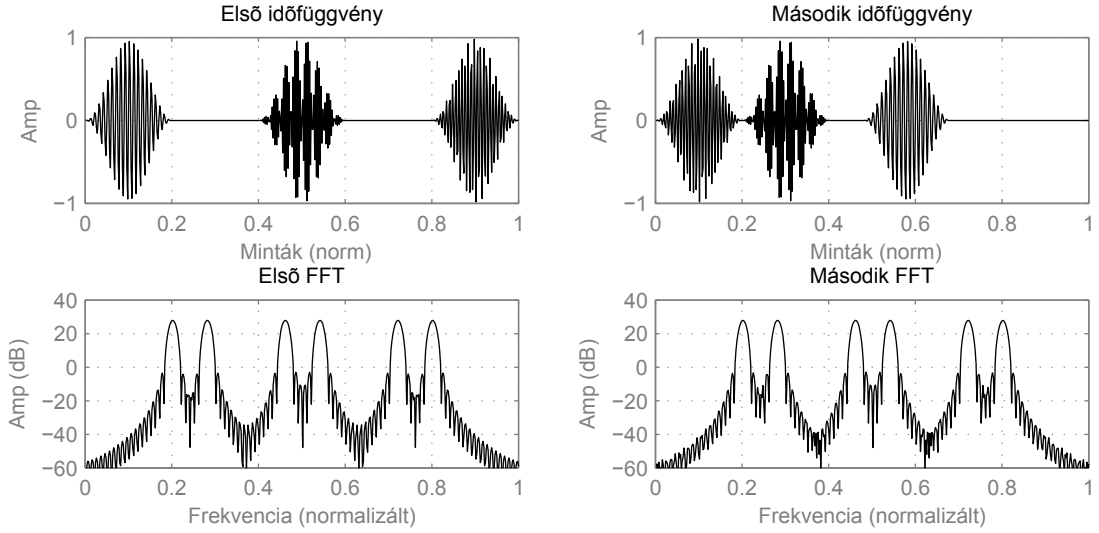


2.13. ábra. Fázis átlapolódás

Tranziens jelek

A tranziens jeleket ez az algoritmus „szétkeni". Ennek az oka az említett frekvenciasávok közti inkoherencia, valamint a $\mathcal{STFT}$ felbontása. Minél nagyobb ablakot használunk az $\mathcal{STFT}$-nél annál pontosabb lesz a frekvenciafelbontás, de annál rosszabb az időbeli felbontás. Hiszen az $\mathcal{STFT}$ egy ablakon belül csak azt határozza meg, milyen frekvenciakomponensek vannak, és mekkora a köztük lévő arány, azt nem tudjuk meg belőle, hogy az adott komponens az ablakon belül hol kezdődött és hol ért véget. Ha kisebb ablakot használunk, akkor ugyan jobb lesz az időtartományban a felbontásunk, de ezzel együtt a frekvenciatartományban csökken.

A 2.14. ábrán látható három jel összege. A három jelhez három különböző frekvenciájú szinuszt használtam, melyeket Hann ablakoltam, és elhelyeztem őket kétféleképpen az időtartományban, majd mindkét jelet $\mathcal{FFT}$-ztem. Az amplitúdókarakterisztikájuk az időfüggvények alatt látszik, és azt mutatja meg, hogy a két amplitúdóspektrum megegyezik.



2.14. ábra. Az $\mathcal{FFT}$ tranziensérzéketlensége

2.2.3. Az algoritmus kihasználatlan lehetőségei

A A.2 függelében mellékelt fázis vokóder implementációnak van még néhány előnye, amit ki lehet használni. A jelet az $\mathcal{STFT}$ segítségével amplitúdó- és fázisértékekre bontjuk, melyek időben változnak. Minden amplitúdó- és fázisérték egy megadott frekvenciához tartozik. De nem kötelező a generátornak is ilyen frekvenciájúnak lenni. Így elérhetjük azt, hogy a frekvenciakomponensek összekeveredjenek, eltolódjanak. Ezzel akár harmonizálni is lehet egy olyan jelet, melynél a felharmonikusok frekvenciája valamilyen oknál fogva elcsúszott[8]. Sőt, pitch shiftelni is lehet a jelet, miközben a sebességét állítjuk. Nincs szükség egy másik algoritmusra, ami ezt megteszni, és az algoritmus számítási igénye sem lesz nagyobb, hiszen csak a bemeneti paramétereket változtatjuk meg.

A másik lehetőség az amplitúdók módosítása. Mivel minden frekvencisávnak megvan az amplitúdóváltozása az idő függvényében, így megvalósíthatunk egy többsávos (ami jelen esetben több ezres nagyságrend) zajszűrőt. Ezt úgy tehetjük meg például, hogy egy másodfokú függvénnyel szorozzuk meg az amplitúdóértékeket, ettől a kis amplitúdók még kisebbek lesznek. Így a spektrumban szétterülő zajt hatékonyan el lehet nyomni, míg a jel kevés torzulást szenved. Egy automatikus hangszínkiegyenlítő, vagy zene lekevérésénél használt többszávos dinamikakompresszor is ugyanígy megvalósítható minden nehézség nélkül.

Ezzel a két tulajdonsággal az algoritmus elég kompakt, és több funkció ellátásra alkalmas egymagában. Bár a zajelnyomással és a felharmonikusok összekeverésével ebben a dolgozatban nem foglalkozom, a függelékben található MATLAB kód kommentezett részébe tettem mindtapéldát ezekre is.

2.2.4. Összefoglaló

A fázis vokódolás sokkal jobb eredményt adó algoritmus, mint az előző részben tárgyalt időtartománybeli szegmentálás. Természetesen a tranziens jelek, és nagy sebességállítás esetén itt is fellépnek nem kívánt hatások, mint a „szétkenődés". Hátránya még, hogy nagyobb számítási kapacitást igényel, mind az $\mathcal{STFT}$, mind a generátorbank megvalósítása. A szimulációim során a 2048 mintás analízis ablakot kielégítőnek találtam. Az 1024 mintás frekvenciafelbontása még kevésnek bizonyult. Ablakfüggvénynek Hanning ablakokat használtam.

2.2.5. Kitekintés

Mivel a hibák egyik oka az $\mathcal{STFT}$ felbontási problémája, ezért az algoritmus javításához ezt kellene megoldani. Egy megoldás a több párhuzamosan futó, más ablakméretet használó $\mathcal{STFT}$ alkalmazása [9]. Egy másik megoldás lehet a folytonos idejű wavelet-transzformáció használata, mely elvileg jobb felbontást biztosít, mint a $\mathcal{STFT}$ [10]. Az algoritmus nagyrésze megegyezik a fentivel, csak a wavelet-transzformációt használja a $\mathcal{STFT}$ helyett, és ekkor a generátorbank sem szinuszokból állna, hanem wavelet-ekből. Ezzel az eljárással szeretnék még foglalkozni a jövőben.

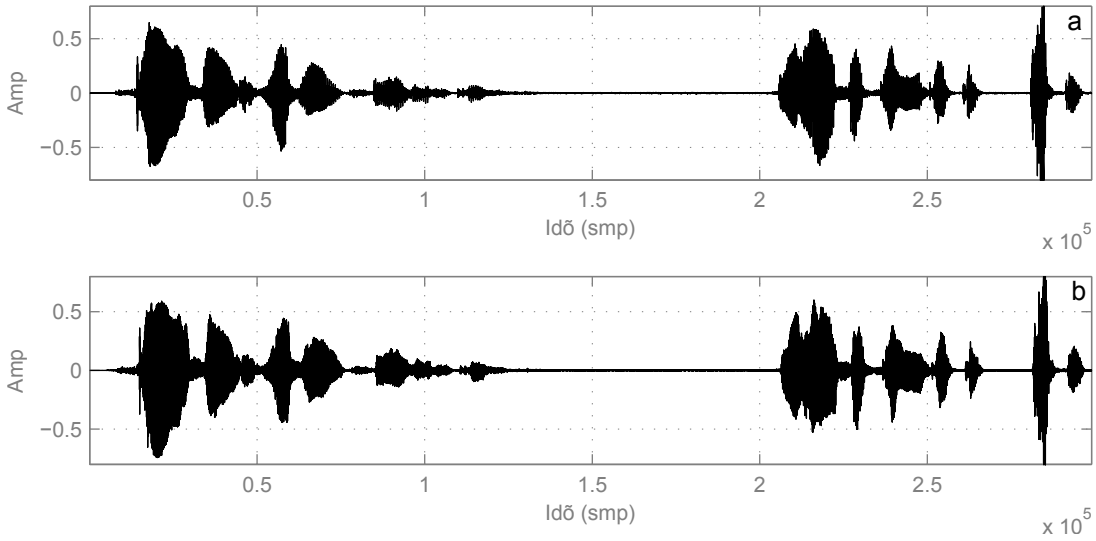
3. fejezet

Pitch Shift

„ Pár hangnyi dallamok, mintha kőbe vésve állták volna századok viharát. ”

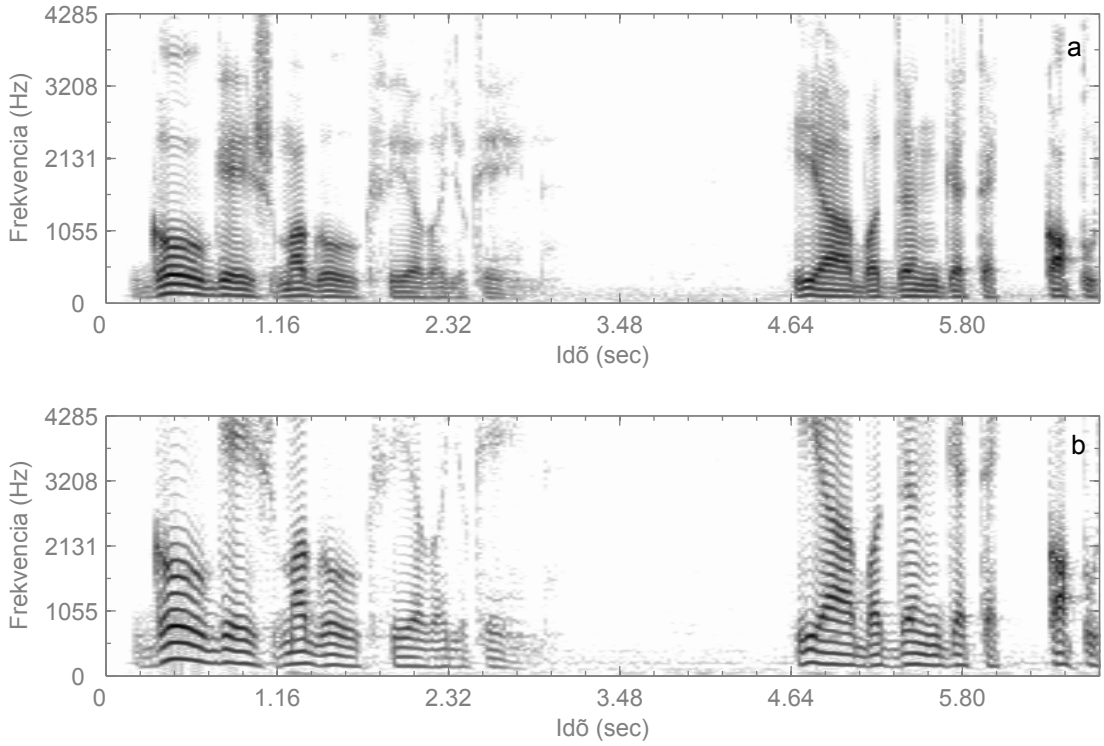
Kodály Zoltán

A time stretch algoritmus rokonalgoritmusa a pitch shift, mely működésében nagyon hasonló. A pitch shift algoritmus a jelfolyam hosszát változatlanul hagyja, míg a frekvenciakomponenseit eltolja, másik hangra emeli. Felhasználási területe stúdiótechnikában főleg ének-, de akár más hangsáv korrigálása, egyszólamú éneksáv többszólamúvá alakítása.



3.1. ábra. Eredeti (a) és pitch shiftelt jel (b) az időtartományban.

A 3.1. ábrán látható, hogy az eredeti és a pitch shiftelt jelnek ugyanaz az időbeli lefutása.



3.2. ábra. Eredeti (a) és pitch shiftelt jel (b) a frekvenciatartományban.

A 3.2. ábrán pedig azt látjuk, hogy a frekvenciatartományban a feldolgozott jel frekvenciakomponensei el vannak csúsztatva.

Hangmagasság állítása

A pitch shift által eltolt frekvenciák nem úgy tolódnak el, hogy minden frekvenciakomponens azonos Δf frekvenciával tolódik arrébb:

$$f_k^{'} \neq f_k + \Delta f \tag{3.1}$$

hiszen ez megegyezne a Fourier-transzformáció modulációs tételével. (3.2)

$$\mathcal{F}\left\{\mathrm{e}^{-\mathrm{j}\omega_0 t} f(t)\right\} = F(\omega - \omega_0) \tag{3.2}$$

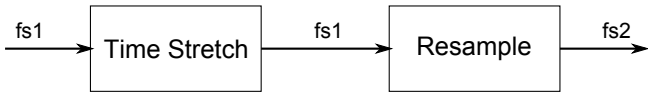
ez esetben nem is kellene hozzá a sebességállítás, hiszen Hilbert-szűrővel és szorzással meg lehetne oldani, oly módon, hogy a beérkező jelet Hilbert-szűrjük, ettől az kilencven fokos fázistolást szenved, majd az eredeti jelet megszorozzuk koszinusszal, a Hilbert-szűrtet pedig szinusszal, majd ezeket összeadjuk. A koszinusz és szinusz azonos ω frekvenciájú. Ez az ω frekvencia lesz az eltolás nagysága. A pitch shift által eltolt frekvenciáknak harmonikusoknak kell maradniuk. Tehát egy adott r_{pitch} értékszeresére kell minden frekvenciának változnia:

$$f_k^{'} = f_k \cdot r_{\mathrm{pitch}} \tag{3.3}$$

Ha az analóg világ példájából indulunk ki, akkor például egy lemezlejátszó lassítást kellene megvalósítani digitálisan. Ezzel az a gond, hogy a lemezlejátszónál a jel sebessége is lassul. De felhasználnánk hozzá az előző fejezetben leírt time stretch algoritmusok valamelyikét, hogy ezt a lassulást ellensúlyozzuk. Már csak a lemezlejátszó lassulását kell reprodukálni, amit meg lehet valósítani újramintavételezéssel.

3.1. Újramintavételezés

Az újramintavételezés felfogható úgy, mint analóg sebességállítás. Ha növeljük a minták számát a jelben, tehát túlmintavételezünk, és az eredeti mintavételi frekvencián játsszuk le őket, akkor a jelünk mélyülni fog, ha csökkentjük a minták számát, és az eredeti mintavételi frekvencián játsszuk le őket, magasabb lesz. Ha a time stretchelt jelünket úgy mintavételezzük újra, hogy az éppen ellensúlyozza a time stretch hatását, például egy kétszer olyan hosszúra timestretchelt jelet újramintavételezünk az eredeti mintavételi frekvenciája felével, akkor az eredeti jel hosszát kapjuk vissza, csak a frekvenciakomponensek máshol lesznek. Mindegyik feleakkora lesz, mint az eredeti jelben, tehát egy oktávval lejjebb toltuk a hangokat. A 3.3. ábrán látható a pitch shift blokk diagramja és benne a time stretch és az újramintavételezés elhelyezkedése. A kimeneten megjelenő f_{s2} jelet az eredeti f_{s1} mintavételi frekvenciával kell lejátszani.



3.3. ábra. A pitch shift blokkdiagramja

Az újramintavételezésre többféle algoritmust használhatunk. Használhatunk olyat, melyhez szükség van a jel összes mintájára, tehát csak offline módon, a times stretch befejeződése után tudjuk végrehajtani, de használhatunk online algoritmust is, melynek segítségével a time stretch algoritmussal egy időben tudjuk végrehajtani a pitch shiftet is[11]. Az alábbiakban ezek közül ismertetek néhányat.

3.1.1. Offline újramintavételezés

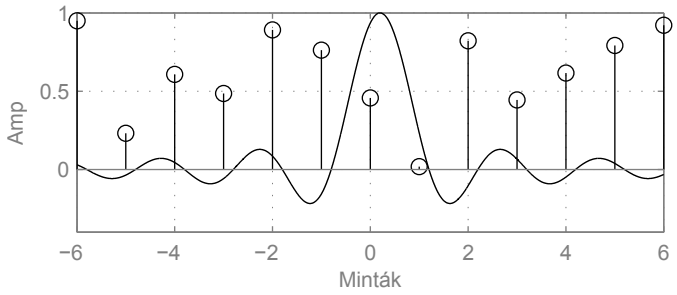
Az offline algoritmusok előnye, hogy jobb minőségű újramintavételezést produkálnak, hiszen nem időkritikus már a feladat, és rendelkezésre áll az összes minta a jelből.

Sinc alapú újramintavételezés

A sinc alapú interpoláció egy sinc alakú jellel súlyozza a mintákat, mely súlyozott összegből számítja ki az aktuális mintát. A (3.4) képletből ez látható, ahol $T = 1/f_{\mathrm{d}}$, f_{s} a mintavételezési frekvencia, valamint sinc a normalizált sinc függvény. Ha a sinc-et jól választjuk meg, akkor az alacsonyabb mintavételi sebességre való áttéréskor sem történik átlapolódás, mert a sinc a jelet egyben meg is szűri.

$$x(t) = \sum_{n=-\infty}^{\infty} x[n] \cdot sinc\left(\frac{t - nT}{T}\right) \tag{3.4}$$

A sinc interpolációt a 3.4. ábra szemlélteti.



3.4. ábra. Sinc alapú újramintavételezés

Spline alapú újramintavételezés

A spline alapú interpolációt használó újramintavételezés a megadott minták közé mintapáronként egy-egy polinom (általában harmadfokú) függvényt illeszt, úgy, hogy a mintáknál az illesztett polinom folytonos legyen a függvény egy megadott deriváltjáig. Ha az új mintavételi frekvencia alacsonyabb a réginél, akkor szükséges átlapolódásgátló szűrő a spline interpoláció előtt, hogy a frekvenciatartományban ne legyen átlapolódás. A spline interpolációt a 3.5. ábra szemlélteti.

3.5. ábra. Spline alapú interpoláció

3.1.2. Online újramintavételezés

Az online algoritmusok előnye, hogy real-time alkamazásokban is használhatóak, hátrányuk általban, hogy jobb minőségű újramintavételezés eléréséhez nagyobb számítási kapacitás szükséges.

Interpoláció alapú újramintavételezés

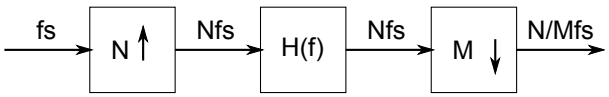
A fenti két offline újramintavételezés is felfogható mint interpoláció, de ezek azért nem valósíthatók meg online módon, mert mind a sinc, mind a spline-nál egy minta kiszámolásához szükség van a jel összes mintájára, ezért ha online módon akarnánk implementálni, akauzális hálózatot kellene megvalósítanunk.

Az interpoláció alapú újramintavételek megpróbálnak az eddig ismert mintákra (vagy inkább a minták részhalmazára) egy illesztést elvégezni és ezzel kiszámolni az éppen szükséges aktuális új mintavételi frekvenciájú mintát.

Szükség lehet átlapolásgátló szűrőre a mintavételi frekvencia csökkentése esetén.

Szűrős újramintavételezés

A szűrő alapú újramintavételezés, a jel racionális arányú újramintavételezésére alkalmas. Például ha $\frac{2}{3}$-ára akarjuk megváltoztatni a mintavéltei sebességet, akkor először kétszeresen túlmintavételezzük a jelet, ami annyit jelent, hogy minden második minta után beszúrunk egy nullát, majd a jelet megszűrjük egy átlapolódás gátló (anti-alias) szűrővel, melynek törésponti frekvenciája a túlmintavételezett mintavéltei frekvenciának a harmadánál van, ezután csak minden harmadik mintát tartjuk meg. A szűrés és minták eldobása összevonható egy polifázisos szűrővé, amelynek következtében kevesebb számítási kapacitást igényel az eljárás. A blokkdiagram a 3.6. ábrán látható.



3.6. ábra. Racionális arányú újramintavételezés

4. fejezet

Eredmények

„ Nem csak az ember tanulmányozza a természetet; a természet is tanulmányozza az embert; úgy, hogy próbára teszi. ”

Osvát Ernő

Dolgozatomban ismertettem az alapvető time strech és pitch shift algoritmusokat. Ezeket a függelékben található kódokkal szimuláltam is. Szimulációim bemenő paramétereit és eredményeit elérhetővé tettem az interneten a következő címen, hogy meg lehessen őket hallgatni.

`http://users.hszk.bme.hu/~gr602/honlap/TDK/`

A tárhely kapacitás és az esetleges lassú internet kapcsolat miatt a jeleket átalakítottam 192kbit/s tömörítésü $f_s = 44100$ Hz mintavételi frekvenciájúak monó MP3-makká.

A pitch shift minták előállításához egy spline alapú újramintavételező algoritmus lett használva, mely a jelet a frekvenciatartományban elhanyagolhatóan befolyásolja csak.

A jelek $\mathcal{STFT}$ hibabecsléséhez 8192 minta hosszú 50% átlapolódású Hamming ablakkal lett a spectrogram kiszámítva.

4.1. Tesztjelek

A tesztjelek mind $f_s = 44100$ Hz mintavételi frekvenciájúak, 16 bit, monó Windows PCM wav formátúak. A teszteléshez különböző tulajdonságokkal rendelkező jeleket választottam.

4.1.1. Beszédjel

Az első tesztjel beszédjel. Egy részlet Ady Endre - Góg és Magóg fia vagyok én... c. verséből. Azért került kiválasztásra, mert jól megfigyelhetők rajta, hogy viselkednek az algoritmusok monofonikus jelen, valamint a jel elég tiszta, olyan szempontból, hogy nem sok visszhang, vagy teremzaj van rajta. Igaz viszonylag zajosnak mondható, mert a zajszint peak értéke $-43 - -50$ dB körül mozog.

Egysávos SOLA

`http://users.hszk.bme.hu/~gr602/honlap/TDK/beszed_Simple_SOLA_0.7.mp3`

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Ablak méret	8192 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = 0.5 wnd_{\text{lng}}$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	2.110863e-012
Hibabecslés ($\mathcal{STFT}$)	3.877344e-005

`http://users.hszk.bme.hu/~gr602/honlap/TDK/beszed_Simple_SOLA_1.5.mp3`

Hosszmegváltozás	150% ($r_{\text{length}} = 1.5$)
Ablak méret	8192 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = 0.25 wnd_{\text{lng}}$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	1.723582e-012
Hibabecslés ($\mathcal{STFT}$)	2.661885e-005

Többsávos SOLA

`http://users.hszk.bme.hu/~gr602/honlap/TDK/beszed_Multi_SOLA_0.7.mp3`

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Szűrők típusa	8-tap FIR
Szűrők száma	3
Low - Frekvencia tartomány	0-500 Hz
Low - Ablak típus és méret	HS-2048 O-4096 HE-2048
Low - Bemeneti ablaktávolság	$n_1 = 0.5 wnd_{\text{lng}}$
Low - Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Mid - Frekvencia tartomány	500-5000 Hz
Mid - Ablak típus és méret	HS-1024 O-4096 HE-1024
Mid - Bemeneti ablaktávolság	$n_1 = 0.5 wnd_{\text{lng}}$
Mid - Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
High- Frekvencia tartomány	5000-22050 Hz
High- Ablak típus és méret	HS-512 O-2048 HE-512
High- Bemeneti ablaktávolság	$n_1 = 0.5 wnd_{\text{lng}}$
High- Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	2.552878e-012
Hibabecslés ($\mathcal{STFT}$)	5.847221e-005

Az ablak típus és méretnél a HS-2048 O-4096 HE-2048 jelentése a következő. HS azt jelenti „Hanning start”, ez egy Hanning ablak első fele, ezzel kezdődik az ablak. Az utána

következő szám a félablak hosszát adja meg mintában. Az O a ones szóból keletkezett, és azt jelenti, hogy ezen a szakaszon csupa egyes az ablakfüggvény. A HE „Hanning end”-et jelent, és egy Hanning második fele, ezzel van vége az ablaknak. Ez a jelölésforma a többi táblázatra is igaz.

`http://users.hszk.bme.hu/~gr602/honlap/TDK/beszed_Multi_SOLA_1.5.mp3`

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Szűrők típusa	8-tap FIR
Szűrők száma	3
Low - Frekvencia tartomány	0-500 Hz
Low - Ablak típus és méret	HS-2048 O-4096 HE-2048
Low - Bemeneti ablaktávolság	$n_1 = 0.3 wnd_{\text{lng}}$
Low - Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Mid - Frekvencia tartomány	500-5000 Hz
Mid - Ablak típus és méret	HS-1024 O-4096 HE-1024
Mid - Bemeneti ablaktávolság	$n_1 = 0.3 wnd_{\text{lng}}$
Mid - Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
High- Frekvencia tartomány	5000-22050 Hz
High- Ablak típus és méret	HS-512 O-2048 HE-512
High- Bemeneti ablaktávolság	$n_1 = 0.3 wnd_{\text{lng}}$
High- Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	1.934009e-012
Hibabecslés ($\mathcal{STFT}$)	1.893983e-003

Wavelet-szűrőbankos SOLA

`http://users.hszk.bme.hu/~gr602/honlap/TDK/beszed_Wavelet_0.7.mp3`

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Ablak méret	5120 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = wnd_{\text{lng}} k/8$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	2.387873e-012
Hibabecslés ($\mathcal{STFT}$)	1.037630e-004

`http://users.hszk.bme.hu/~gr602/honlap/TDK/beszed_Wavelet_1.5.mp3`

Hosszmegváltozás	150% ($r_{\text{length}} = 1.5$)
Ablak méret	5120 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = wnd_{\text{lng}} k/16$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	2.325414e-012
Hibabecslés ($\mathcal{STFT}$)	2.005662e-005

Ahol $k \in [1;5]$ a frekvenciasáv indexét jelenti. Az egyes index a magas frekvenciasáv, az ötös a mély frekvenciasávot jelenti. Ez a többi teszjelre is érvényes.

Fázis vokóder

`http://users.hszk.bme.hu/~gr602/honlap/TDK/beszed_PhaseV_0.7.mp3`

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Ablak méret	2048 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = 256$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	2.178275e-012
Hibabecslés ($\mathcal{STFT}$)	9.199633e-005

`http://users.hszk.bme.hu/~gr602/honlap/TDK/beszed_PhaseV_1.5.mp3`

Hosszmegváltozás	150% ($r_{\text{length}} = 1.5$)
Ablak méret	2048 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = 256$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	9.045182e-013
Hibabecslés ($\mathcal{STFT}$)	2.129713e-005

Értékelés

Lassításnál a fázis vokódernek volt a legkisebb becsült $\mathcal{STFT}$ hibája, de gyorsításnál elég rossz eredményeket produkált. A gyorsításnál produkált rossz eredmények valószínűleg a jel szétkenődése okozhatta, hisz a beszédben viszonylag sok tranziens jel van.

4.1.2. Klasszikus zene

A második tesztjel egy klasszikus zene, a Carmina Burana-ból egy részlet. Azért került kiválasztásra, mert sok folyamatos hang van benne.

Egysávos SOLA

`http://users.hszk.bme.hu/~gr602/honlap/TDK/classic_Simple_SOLA_0.7.mp3`

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Ablak méret	8192 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = 0.5 wnd_{\text{lng}}$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	2.632045e-011
Hibabecslés ($\mathcal{STFT}$)	3.395466e-004

http://users.hszk.bme.hu/~gr602/honlap/TDK/classic_Simple_SOLA_1.5.mp3

Hosszmegváltozás	150% ($r_{\text{length}} = 1.5$)
Ablak méret	8192 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = 0.23 wnd_{\text{lng}}$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	2.189545e-011
Hibabecslés ($\mathcal{STFT}$)	6.755824e-005

Többsávos SOLA

http://users.hszk.bme.hu/~gr602/honlap/TDK/classic_Multi_SOLA_0.7.mp3

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Szűrők típusa	8-tap FIR
Szűrők száma	3
Low - Frekvencia tartomány	0-500 Hz
Low - Ablak típus és méret	HS-2048 O-4096 HE-2048
Low - Bemeneti ablaktávolság	$n_1 = 0.5 wnd_{\text{lng}}$
Low - Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Mid - Frekvencia tartomány	500-5000 Hz
Mid - Ablak típus és méret	HS-1024 O-4096 HE-1024
Mid - Bemeneti ablaktávolság	$n_1 = 0.5 wnd_{\text{lng}}$
Mid - Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
High- Frekvencia tartomány	5000-22050 Hz
High- Ablak típus és méret	HS-512 O-2048 HE-512
High- Bemeneti ablaktávolság	$n_1 = 0.5 wnd_{\text{lng}}$
High- Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	2.361463e-011
Hibabecslés ($\mathcal{STFT}$)	2.182197e-004

http://users.hszk.bme.hu/~gr602/honlap/TDK/classic_Multi_SOLA_1.5.mp3

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Szűrők típusa	8-tap FIR
Szűrők száma	3
Low - Frekvencia tartomány	0-500 Hz
Low - Ablak típus és méret	HS-2048 O-4096 HE-2048
Low - Bemeneti ablaktávolság	$n_1 = 0.3wnd_{\text{lng}}$
Low - Kimeneti ablaktávolság	$n_2 = r_{\text{length}}n_1$
Mid - Frekvencia tartomány	500-5000 Hz
Mid - Ablak típus és méret	HS-1024 O-4096 HE-1024
Mid - Bemeneti ablaktávolság	$n_1 = 0.3wnd_{\text{lng}}$
Mid - Kimeneti ablaktávolság	$n_2 = r_{\text{length}}n_1$
High- Frekvencia tartomány	5000-22050 Hz
High- Ablak típus és méret	HS-512 O-2048 HE-512
High- Bemeneti ablaktávolság	$n_1 = 0.3wnd_{\text{lng}}$
High- Kimeneti ablaktávolság	$n_2 = r_{\text{length}}n_1$
Hibabecslés ($\mathcal{FFT}$)	1.734076e-011
Hibabecslés ($\mathcal{STFT}$)	6.675221e-005

Wavelet-szűrőbankos SOLA

`http://users.hszk.bme.hu/~gr602/honlap/TDK/classic_Wavelet_0.7.mp3`

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Ablak méret	5120 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = wnd_{\text{lng}}k/8$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}}n_1$
Hibabecslés ($\mathcal{FFT}$)	2.810553e-011
Hibabecslés ($\mathcal{STFT}$)	3.803565e-004

`http://users.hszk.bme.hu/~gr602/honlap/TDK/classic_Wavelet_1.5.mp3`

Hosszmegváltozás	150% ($r_{\text{length}} = 1.5$)
Ablak méret	5120 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = wnd_{\text{lng}}k/16$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}}n_1$
Hibabecslés ($\mathcal{FFT}$)	2.425069e-011
Hibabecslés ($\mathcal{STFT}$)	8.773230e-005

Fázis vokóder

`http://users.hszk.bme.hu/~gr602/honlap/TDK/classic_PhaseV_0.7.mp3`

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Ablak méret	2048 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = 256$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	1.670774e-011
Hibabecslés ($\mathcal{STFT}$)	2.053745e-004

`http://users.hszk.bme.hu/~gr602/honlap/TDK/classic_PhaseV_1.5.mp3`

Hosszmegváltozás	150% ($r_{\text{length}} = 1.5$)
Ablak méret	2048 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = 256$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	7.201861e-012
Hibabecslés ($\mathcal{STFT}$)	2.392710e-005

Értékelés

A klasszikus zene kisebb tranzienstartalma, és a hosszan kitartott hangok miatt, a fázisvokóder ebben az esetben a gyorsításnál is a legjobb $\mathcal{STFT}$ becsült hibával rendelkezett.

4.1.3. Elektronikus zene

A harmadik tesztjel Kosheen - Overkill c. számának egy részlete. Ebben a zenében mind tranziens, mind folyamatos hangok előfordulnak, ezért esett erre a választás.

Egysávos SOLA

`http://users.hszk.bme.hu/~gr602/honlap/TDK/electro_Simple_SOLA_0.7.mp3`

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Ablak méret	8192 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = 0.5 wnd_{\text{lng}}$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	3.831304e-012
Hibabecslés ($\mathcal{STFT}$)	5.076439e-005

`http://users.hszk.bme.hu/~gr602/honlap/TDK/electro_Simple_SOLA_1.5.mp3`

Hosszmegváltozás	150% ($r_{\text{length}} = 1.5$)
Ablak méret	8192 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = 0.25 wnd_{\text{lng}}$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	2.800395e-012
Hibabecslés ($\mathcal{STFT}$)	3.646257e-005

Többsávos SOLA

`http://users.hszk.bme.hu/~gr602/honlap/TDK/electro_Multi_SOLA_0.7.mp3`

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Szűrők típusa	8-tap FIR
Szűrők száma	3
Low - Frekvencia tartomány	0-500 Hz
Low - Ablak típus és méret	HS-2048 O-4096 HE-2048
Low - Bemeneti ablaktávolság	$n_1 = 0.5 wnd_{\text{lng}}$
Low - Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Mid - Frekvencia tartomány	500-5000 Hz
Mid - Ablak típus és méret	HS-1024 O-4096 HE-1024
Mid - Bemeneti ablaktávolság	$n_1 = 0.5 wnd_{\text{lng}}$
Mid - Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
High- Frekvencia tartomány	5000-22050 Hz
High- Ablak típus és méret	HS-512 O-2048 HE-512
High- Bemeneti ablaktávolság	$n_1 = 0.5 wnd_{\text{lng}}$
High- Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	4.169972e-012
Hibabecslés ($\mathcal{STFT}$)	4.994683e-005

`http://users.hszk.bme.hu/~gr602/honlap/TDK/electro_Multi_SOLA_1.5.mp3`

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Szűrők típusa	8-tap FIR
Szűrők száma	3
Low - Frekvencia tartomány	0-500 Hz
Low - Ablak típus és méret	HS-2048 O-4096 HE-2048
Low - Bemeneti ablaktávolság	$n_1 = 0.3wnd_{\text{lng}}$
Low - Kimeneti ablaktávolság	$n_2 = r_{\text{length}}n_1$
Mid - Frekvencia tartomány	500-5000 Hz
Mid - Ablak típus és méret	HS-1024 O-4096 HE-1024
Mid - Bemeneti ablaktávolság	$n_1 = 0.3wnd_{\text{lng}}$
Mid - Kimeneti ablaktávolság	$n_2 = r_{\text{length}}n_1$
High- Frekvencia tartomány	5000-22050 Hz
High- Ablak típus és méret	HS-512 O-2048 HE-512
High- Bemeneti ablaktávolság	$n_1 = 0.3wnd_{\text{lng}}$
High- Kimeneti ablaktávolság	$n_2 = r_{\text{length}}n_1$
Hibabecslés ($\mathcal{FFT}$)	2.583829e-012
Hibabecslés ($\mathcal{STFT}$)	6.481955e-005

Wavelet-szűrőbankos SOLA

`http://users.hszk.bme.hu/~gr602/honlap/TDK/electro_Wavelet_0.7.mp3`

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Ablak méret	5120 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = wnd_{\text{lng}}k/8$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}}n_1$
Hibabecslés ($\mathcal{FFT}$)	4.550796e-012
Hibabecslés ($\mathcal{STFT}$)	7.786901e-003

`http://users.hszk.bme.hu/~gr602/honlap/TDK/electro_Wavelet_1.5.mp3`

Hosszmegváltozás	150% ($r_{\text{length}} = 1.5$)
Ablak méret	5120 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = wnd_{\text{lng}}k/16$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}}n_1$
Hibabecslés ($\mathcal{FFT}$)	3.074589e-012
Hibabecslés ($\mathcal{STFT}$)	2.428338e-005

Fázis vokóder

`http://users.hszk.bme.hu/~gr602/honlap/TDK/electro_PhaseV_0.7.mp3`

Hosszmegváltozás	70% ($r_{\text{length}} = 0.7$)
Ablak méret	2048 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = 256$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	2.492068e-012
Hibabecslés ($\mathcal{STFT}$)	7.059289e-005

`http://users.hszk.bme.hu/~gr602/honlap/TDK/electro_PhaseV_1.5.mp3`

Hosszmegváltozás	150% ($r_{\text{length}} = 1.5$)
Ablak méret	2048 minta
Ablak típus	Hanning
Bemeneti ablaktávolság	$n_1 = 256$
Kimeneti ablaktávolság	$n_2 = r_{\text{length}} n_1$
Hibabecslés ($\mathcal{FFT}$)	1.124620e-012
Hibabecslés ($\mathcal{STFT}$)	1.331132e-005

Értékelés

Mivel ebben a jelben mind tranziensek, mind folyamatos jelkomponensek megtalálhatóak, várható volt, hogy a fázisvokóder esetében a gyorsításnál kapott eredmények nem lesznek olyan rosszak, mint a beszéd gyorsításánál kapottak.

5. fejezet

Összefoglalás

„ Ha elértél a véghez, és nem tudod, hogyan tovább, akkor kezdd elölről. ”

Peter Freund

Dolgozatomban igyekeztem átfogó képet adni a time stretch és pitch shift algoritmusokról, kezdve egy rövid történelmi áttekintéssel, amiben az analóg bakelitlemez és magnószalag világtól, eljutottam a digitális jelfeldolgozó processzorokig és számítógépekig. Közben igyekeztem végigvezetni, hogy mikor és hol volt, és lehet szükség time stetch és pitch shift algoritmusokra, milyen módon tudja őket használni a stúdiótechnika, vagy a szórakoztatóipar.

A következő fejezet a sebességállító algoritmusok bemutatásáról szólt, részletesebben kifejtve az időtartománybeli szegmentálás és a fázis vokódolás technikáját, valamint kitérve néhány érdekességre.

Ezután a pitch shift és a time stretch algoritmusok rokonságára mutattam rá, és kifejtettem, hogyan lehet a time stretch algoritmusokat módosítani úgy, hogy azok pitch shift algoritmusokként működjenek.

Ezt követően a bemutatott módszerek MATLAB implementációin futtatott szimulációs eredményeim felsorolása következett, ahol összehasonlító eljárások segítségével igyekeztem rangsorolni az algoritmusokat. Megállapítottam, hogy az általam vizsgált összes algoritmus alkalmas time strecth és pitch shift megvalósítására. Minőség szempontjából ez a következők szerint részletezhető: Lassításra a leginkább a fázis vokóder alkalmazható, gyorsítás esetén a legjobb módszer függ a feldolgozandó jeltől. Zenei anyagok esetén, amelyekben kevés tranizens szakasz fordul elő, ugyancsak a fázisvokódolás a legjobb megoldás, míg beszédjel, illetve sok tranziens részt tartalmazó jelek esetén, az időtartománybeli szegmentálás alkalmazható. Az algoritmus jellegéből adódóan ugyanez a minősítés adható pitch shift algoritmusok esetén is. Szimulációim hanganyagát az interneten elérhetővé tettem.

Irodalomjegyzék

[1] U. Zölzer, *Digital Audio Effects*. John Wiley & Sons Ltd, 2002. ISBN 0-471-49078-4.

[2] M. Verhelst, W.; Roelands, „An overlap-add technique based on waveform similarity (wsola) for high quality time-scale modification of speech," in *Acoustics, Speech, and Signal Processing 1993 IEEE International Conference*, vol. 2, pp. 554 – 557, 27-30 Apr 1993.

[3] O. W. J. L. W. Wong, P.H.W.; Au, „On improving the intelligibility of synchronized over-lap-and-add(sola) at low tsm factor," in *Speech and Image Technologies for Computing and Telecommunications., Proceedings of IEEE*, vol. 2, pp. 487 – 490, 2-4 Dec 1997.

[4] C. K. Chui, *An Introduction to Wavelets*. Academic Press, 1992. Library of Congress Catalog Card Number: 91-58831.

[5] F. Keinert, *Wavelets and Multiwavelets*. Chapman & HallCRC, 2004. ISBN 1-58488-304-9.

[6] A. Röbel, „A new approach to transient processing in the phase vocoder," in *Proc. of the 6th Int. Conference on Digital Audio Effects (DAFx-03)*, 8-11 Sept 2003.

[7] J. L. M. Dolson, „Improved phase vocoder time-scale modification of audio," in *Speech and Audio Processing, IEEE Transactions*, vol. 7, pp. 323 – 332, May 1999.

[8] L. J. D. M., „New phase-vocoder techniques for pitch-shifting, harmonizing andother exotic effects," in *Applications of Signal Processing to Audio and Acoustics, 1999 IEEE Workshop*, no. 10.1109/ASPAA.1999.810857, pp. 91 – 94, 1999.

[9] B. J., „Automatic technique in frequency domain for near-lossless time-scale modification of audio," in *International Computer Music Conference*, pp. 396 – 399, 2000.

[10] D. G. P. D. M. B. M. M., „Applications of the continuous wavelet transform in the processingof musical signals," in *Digital Signal Processing Proceedings*, vol. 2, pp. 563 – 566, 2-4 Jul 1997.

[11] U. Zölzer, *Digital Audio Signal Processing*. John Wiley & Sons Ltd, 1997. ISBN 0-471-97226-6.

Internetes hanganyag

http://users.hszk.bme.hu/~gr602/honlap/TDK/

A. Függelék

Implementált kódok

A.1. Időtartománybeli szegmentálás

A.1.1. A SOLA algoritmust megvalósító MATLAB függvény

```
0  function Y = SOLA(X,wnd,n1,n2)
   % SOLA Megvalositas
   %
   % X      - Bemeneti vektor
   % wnd    - Ablak fuggveny
5  % n1     - Bemeneti ugras
   % n2     - Kimeneti ugras

   Y = zeros(ceil(n2/n1*length(X)),1);
   wndsize = length(wnd);
10 D = zeros(ceil(wndsize/n2)+1,2)-1;

   disp(sprintf('Length Factor: %d',n2/n1));

   wndcnt = 0;
15 for k=1:length(Y)-wndsize

       % Uj ablak kezdodik
       if (mod(k-1,n2)==0)
           l = 1;
20         while(D(l,1)~=-1) l=l+1; end
           D(l,1) = wndcnt;
           D(l,2) = 0;
           wndcnt = wndcnt + 1;
       end
25
       % Osszegzes
       V = 0;
       for l=1:length(D)
           if (D(l,1)~=-1)
30             Y(k)=Y(k)+X(D(l,1)*n1+D(l,2)+1)*wnd(D(l,2)+1);
               V = V + wnd(D(l,2)+1);
               D(l,2)=D(l,2)+1;
               % Ablak vege
               if (D(l,2)+2>wndsize)
35                 D(l,1) = -1;
                   D(l,2) = -1;
               end
           end
       end
```

```
40
    % Volume Normalizalas
    if (V~=0) Y(k) = Y(k)./V; end
end
```

A.1.2. Egysávos SOLA algoritmus fő MATLAB fájlja

```
0  clc;
   clear all;
   format long;
   format compact;

5  speed = 0.8;
   [X,Fs,bit] = wavread('Input.wav');

   wnd = hann(1024*1);
   wnd = wnd(1:round(length(wnd)/2));
10 wnd = [wnd; ones(1024*5,1); 1-wnd];
   n1  = round(length(wnd)*(1-0.7));
   n2  = round(speed*n1);
   Y   = SOLA(X,wnd,n1,n2);

15 wavwrite(Y/max(Y),Fs,bit,'Output.wav');
```

A.1.3. Többsávos SOLA algoritmus fő MATLAB fájlja

```
0  clc;
   clear all;
   format long;
   format compact;

5  speed = 1.5;
   N     = 512;
   win   = hann(N+1);
   flag  = 'scale';
   [X,Fs,bit] = wavread('Input.wav');
10
   % Mely frekvenciasav
   Fc   = 500;
   filt1 = fir1(N, Fc/(Fs/2), 'low', win, flag);
   X1   = filter(filt1,1,X);
15 wnd = hann(1024*5);
   wnd = wnd(1:round(length(wnd)/2));
   wnd = [wnd; ones(1024*10,1); 1-wnd];
   n1  = round(length(wnd)*0.5);
   n2  = round(speed*n1);
20 Y1  = SOLA(X1,wnd,n1,n2);

   % Kozep frekvenciasav
   Fc1 = 500;
   Fc2 = 5000;
25 filt2 = fir1(N, [Fc1 Fc2]/(Fs/2), 'bandpass', win, flag);
   X2  = filter(filt2,1,X);
   wnd = hann(1024*3);
   wnd = wnd(1:round(length(wnd)/2));
   wnd = [wnd; ones(1024*5,1); 1-wnd];
30 n1  = round(length(wnd)*0.5);
   n2  = round(speed*n1);
   Y2  = SOLA(X2,wnd,n1,n2);

   % Magas frekvenciasav
35 Fc   = 5000;
   filt3 = fir1(N, Fc/(Fs/2), 'high', win, flag);
   X3   = filter(filt3,1,X);
```

```
wnd = hann(1024*1);
wnd = wnd(1:round(length(wnd)/2));
wnd = [wnd; ones(1024*2,1); 1-wnd];
n1  = round(length(wnd)*0.5);
n2  = round(speed*n1);
Y3  = SOLA(X3,wnd,n1,n2);

% Elteresek lehetnek a vektorok kozott, es ezt ki kell egyenliteni
Y = Y1(1:min([length(Y1) length(Y2) length(Y3)])) ...
  + Y2(1:min([length(Y1) length(Y2) length(Y3)])) ...
  + Y3(1:min([length(Y1) length(Y2) length(Y3)]));

wavwrite(Y/max(Y),Fs,bit,'Output.wav');
```

A.2. Fázis vokódolás

A.2.1. A fázisvokódálst megvalósító MATLAB függvény

```
function Y = PhaseVocoder(X,wnd,n1,n2)
% PhaseVocoder
%
% X     - Bemeneti vektor
% wnd   - Ablak fuggveny
% n1    - Bemeneti ugras
% n2    - Kimeneti ugras

Y = zeros(ceil(n2/n1*length(X)),1);
wndsize = length(wnd);
disp(sprintf('Length Factor: %d',n2/n1));

Freq = [0:floor(wndsize/2)-1]'./wndsize;
% ========================================
% Frekvenci beavatkozasok (pl harmonizalas)
% Freq = 0.995*Freq+0.005;
% ========================================

Amp0 = zeros(floor(wndsize/2),1);
Phs0 = zeros(floor(wndsize/2),1);
Phsh = zeros(floor(wndsize/2),1);
k = 1;
m = 1;
while (k<length(X)-length(wnd))
  F = fft(X(k:k+wndsize-1).*wnd);
  Amp = abs(F(1:floor(wndsize/2)));

  % ========================================
  % Amplitudo beavatkozasok (pl zajszures)
  % Amp = (Amp.*(1+3.*[1:floor(wndsize/2)]'./floor(wndsize/2))).^1.6;
  % ========================================

  Phs = angle(F(1:floor(wndsize/2)));

  Ampd = PVIntAmp(Amp0,Amp,n2);
  Amph = Amp0;

  Phsd = PVIntPhs(Phs0,Phs,n1,n2,Freq);
  for l=1:n2
      Amph = Amph+Ampd;
      Phsh = Phsh+Phsd;
      Gen(l) = Amph'*cos(Phsh);
  end
  Y(m:m+n2-1) = Gen';
```

```
45    k = k+n1;
      m = m+n2;
      Amp0 = Amp;
      Phs0 = Phs;
      Phsh = PVPhsWrap(Phsh);
50  end
```

A.2.2. Az amplitúdót interpoláló segédfüggvény

```
0   function Ampd = PVIntAmp(Amp0,Amp,n2)
    % Amplitudo interpolalas
    %
    % Amp0  - Regi amplitudo ertek
    % Amp   - Mostani amplitudo ertek
5   % n2    - Kimeneti ablak hossza
    % Ampd  - Interpolalt amp vektor
    Ampd = (Amp-Amp0)/n2;
```

A.2.3. A fázist interpoláló segédfüggvény

```
0   function Phsd = PVIntPhs(Phs0,Phs,n1,n2,Freq)
    % Fazis interpolalas
    %
    % Phs0  - Regi fazis ertek
    % Phs   - Mostani fazis ertek
5   % n1    - Bemeneti ablak hossza
    % n2    - Kimeneti ablak hossza
    % Freq  - Frekvencia vektor (f/fs)
    % PhsN  - Interpolalt phs vektor (x=Frek y=Time)

10  WndPhs = 2*pi*Freq*n1;
    Phsd = (WndPhs + PVPhsWrap(Phs - Phs0 - WndPhs))./n1;
```

A.2.4. Fázis átlapoló segédfüggvény

```
0   function Y = PVPhsWrap(X)
    % fazis-wrap
    Y = mod(X+pi,2*pi)-pi;
```

A.2.5. A fázis vokóder algoritmus fő MATLAB fájlja

```
0   clc;
    clear all;
    format long;
    format compact;

5   [X,Fs,bit] = wavread('Input.wav');
    X = X(1:100000);

    speed = 2;
    wnd = hann(1024*2);
10  n1  = 256;
    n2  = round(n1*speed);

    tic;
    Y = PhaseVocoder(X,wnd,n1,n2);
15  toc;

    wavwrite(Y./max(Y),Fs,bit,'Output.wav');
```

A.3. Wavelet-szűrőbank

A.3.1. A wavelet-alapú SOLA-t megvalósító MATLAB függvény

```
clc;
clear all;
format long;
format compact;
%warning off;

% nyujtasi arany beallitasa
speed = 0.7;

[X,Fs,bit] = wavread('electro.wav');
Sum = zeros(2*length(X),1);

% 4 szintu struktura Daubechies wavelettel
[C,L] = wavedec(X,4,'db8');

wnd = hann(1024*5);
for k=1:length(L)-2
    n1 = round(length(wnd)*k./8);
    n2 = round(n1*speed);

    % Sav informacio kinyerese
    XX = upcoef('d',detcoef(C,L,k),'db8',k);
    YY = SOLA(XX,wnd,n1,n2);
    Sum(1:length(YY)) = Sum(1:length(YY)) + YY;
end

XX = upcoef('a',appcoef(C,L,'db8',k),'db8',k);
YY = SOLA(XX,wnd,n1,n2);
Sum(1:length(YY)) = Sum(1:length(YY)) + YY;

Sum = Sum./max(Sum);
wavwrite(Sum,Fs,bit,'electro_Wavelet_1.5.wav');
```

A.4. Time stretch jósága

A.4.1. Az $\mathcal{FFT}$ alapú mérés megvalósítása

```
function [FF F] = Meres2(X1,X2)
% Egy hibaerteket szamol a ket jel kozott
%
% X1 - Egyik jel (celszeru a rovidebbet)
% X2 - Masik jel
if length(X2)>length(X1)
    X1 = [X1; zeros(length(X2)-length(X1),1)];
else
    X2 = [X2; zeros(length(X1)-length(X2),1)];
end

F1 = abs(fft(X1));
F2 = abs(fft(X2));

F1 = F1./sum(F1);
F2 = F2./sum(F2);

F = (F1-F2).^2;
FF = sum(F)./length(F1);
F = F./max(F);
```

A.4.2. A spektrogram alapú mérés megvalósítása

```
function [SS S] = Meres(X1,X2,n)
% Egy hibaerteket szamol a ket jel kozott
%
% X1 - Egyik jel (celszeru a rovidebbet)
% X2 - Masik jel
% n  - Ablak hossza
[S1] = spectrogram(X1,2*n,n,2*n,44100);
[S2] = spectrogram(X2,2*n,n,2*n,44100);

ss1 = size(S1);
ss2 = size(S2);

xx1 = [0:1:ss1(2)-1]./(ss1(2)-1);
xx2 = [0:1:ss2(2)-1]./(ss2(2)-1);

S1 = spline(xx1,S1,xx2);

S1 = abs(S1./sum(sum(S1)));
S2 = abs(S2./sum(sum(S2)));

S = (S1-S2).^2;
SS = sum(sum(S))./(ss2(1)*ss2(2));

S = 128*(S./max(max(S)));
```

A.4.3. A mérés fő MATLAB fájlja

```
clc;
clear all;
format long;
format compact;

n = 1024*8;
[X1,Fs,bit] = wavread('Mer1.wav');

[X2,Fs,bit] = wavread('Mer2_SOLA_1.wav');
[SS1 S1] = Meres(X1,X2,n);
[SS2 S2] = Meres2(X1,X2);
disp(sprintf('SOLA1: %d \t %d',SS1,SS2));

[X2,Fs,bit] = wavread('Mer2_SOLA_2.wav');
[SS1 S1] = Meres(X1,X2,n);
[SS2 S2] = Meres2(X1,X2);
disp(sprintf('SOLA2: %d \t %d',SS1,SS2));

[X2,Fs,bit] = wavread('Mer2_PhaseV.wav');
[SS1 S1] = Meres(X1,X2,n);
[SS2 S2] = Meres2(X1,X2);
disp(sprintf('PhsV : %d \t %d',SS1,SS2));

[X2,Fs,bit] = wavread('Mer2_Cubase.wav');
[SS1 S1] = Meres(X1,X2,n);
[SS2 S2] = Meres2(X1,X2);
disp(sprintf('CSX3 : %d \t %d',SS1,SS2));
```