Inhoudsopgave

Inleiding

Wanneer geluid wordt opgenomen is het soms handig om de opname te versnellen of te vertragen. Dit heeft tal van toepassingen, zoals bij het presenteren van het nieuws of bij het beatmatchen van muzieknummers voor een dj. Een nieuwsanker moet veel informatie geven binnen een bepaalde tijdsspanne en af en toe gaat dit ten koste van de verstaanbaarheid. Het kan dus nuttig zijn om te snelle spraak te vertragen. Deze techniek wordt eveneens toegepast bij live muzikale optredens om op de maat te spelen. Dit gebeurt ook wanneer het geluid bij een beeld moet gezet worden, wanneer ze niet samen zijn opgenomen. Een andere toepassing is om teksten die in een normaal tempo zijn voorgelezen te versnellen zodat blinden, die een verscherpt gehoor hebben, ook boeken kunnen 'lezen'. Het omgekeerde gebeurt voor mensen die een vreemde taal leren. [1],[2]

Als het geluid sneller of trager wordt afgespeeld door het veranderen van de bemonsteringsfrequentie, zal de toon hoger of later gaan klinken. Via *time scaling* of *time stretching* kan men een audiosignaal versnellen of vertragen zonder dat er een verschil in toonhoogte is. Time-scaling kan op verschillende manieren gebeuren. Algemeen worden de algoritmes opgedeeld in drie verschillende categorieën: Time domain, Phase-vocoder en signal models. [1]. In dit verslag wordt enkel tijdsdomein algoritmes besproken, meer bepaald OLA (*Overlap and Add*), SOLA (*Synchronous Overlap and Add*) en PSOLA (*Pitch-Synchronous Overlap and Add*). Het basisidee van dit algoritme is eerst het splitsen in overlappende frames van het signaal en het daarna hercombineren van deze frames met telkens een bepaalde overlap om een output signaal te geven. Ten slotte worden deze algoritmes verder aangepast, zodat de time scaling ook in real time kan gebeuren.

Time Stretching Algoritmes

0.1 Introductie

Hoofdstuk 2 beschrijft de verschillende algoritmes voor het versnellen en vertragen van geluidsbestanden. Het hoofdstuk begint met het beschrijven van de *sample rate conversion*, een slechte methode, en gaat verder met de OLA-methode (*Overlap and Add*), het eenvoudigste algoritme. Vervolgens beschrijft deze tekst het SOLA-algoritme (*Synchronous Overlap and Add*), wat het OLA-principe verbetert door de maximaal overeenkomende overlap te zoeken tussen de opeenvolgende frames. Een nog verbeterde versie van dit algoritme is het PSOLA (*Pith-Synchronous Overlap and Add*), die gebruik maakt van de pith of toonhoogte van het signaal. Daarna wordt er een vergelijking gemaakt tussen deze algoritmes door het signaal te plotten en de frequentiespectra te vergelijking en deze te laten beoordelen in een enquete.

0.2 Sample Rate Change

Sample rate change is een minder goede en naïeve benadering van time scaling om het geluidssignaal simpelweg te vertragen of te versnellen. Deze methode kan beschreven worden als de *sample rate conversie* en versnelt of vertraagt het audiosignaal door het uit te rekken of in te krimpen over de hele lengte van het signaal. Het uitgangssignaal wordt volledig vervormd en stemmen zullen niet langer natuurlijk klinken. De voornaamste reden hiervoor is dat sample rate conversie geen rekening houdt met de verandering van de *toonhoogte* of *pitch*. De toonhoogte is de frequentie die het meest bepalend is voor het geluid dat de luisteraar ervaart. Zo krijg je een vervormd uitgangssignaal dat ofwel een lagere toonhoogte krijgt wanneer het fragment vertraagd wordt, ofwel een hogere toonhoogte krijgt wanneer het fragment

Figuur 1: Principe van het OLA-algoritme

versneld wordt.

Sample rate conversie kan eenvoudig geïmplementeerd worden in MATLAB door gebruik te maken van de ingebouwde functie *resample*. **HIER GRAFIEKJES VAN DE OPDRACHT DIE IN DE BUNDEL STAAT OM MEE AAN TE TONEN WAAROM SAMPLE RATE CONVERSION SLECHT IS.**

Methodes als OLA, SOLA en PSOLA veranderen de snelheid van de geluidssignalen zonder dat informatie over de toonhoogte verloren gaat. Deze time stretching algoritmes verhinderen het negatief effect dat optreedt bij sample rate conversion en zullen een natuurlijker uitgangssignaal creëren.

0.3 Overlap and Add (OLA)

0.3.1 Concept

Een zeer eenvoudig time stretching algoritme is het geluidssignaal te verdelen in overlappende deeltjes van een vaste lengte N. Deze originele fragmenten worden opgeschoven met een tijdsverschuiving van S_a samples. Vervolgens **worden** de blokjes gepositioneerd met een tijdsverschuiving van $S_s = \alpha.S_a$, waarbij α een variabele is die vrij gekozen mag worden. Alpha is de variabele die bepaalt in welke mate het geluidsfragment vertraagd of versneld wordt en ligt typisch ... **binnen een bepaald interval...)**

FIGUUR MOET NOG GEHERPOSITIONEERD WORDEN

Door deze verschuiving overlappen kleine delen van de opeenvolgende fragmenten en op dit overlappend blok **wordt** een fade-in en fade-out functie geïmplementeerd **(and summed sample-by-sample?)**. Uiteindelijk schakelt het algoritme de uiteengehaalde fragmenten weer aan elkaar om op deze manier een tijdsgestrekt signaal te bekomen.

0.3.2 Uitwerking

Bijlage A ??? *bevat de volledige MATLAB-code voorzien van toelichting.*

Deze paragraaf geeft een beschrijving van de implementatie van het OLA-algoritme in `MATLAB`. De OLA-functie gebruikt vijf argumenten, waarvan het eerste verwijst naar het geluidssignaal dat ingeladen moet worden.

```
function timeshifted_signal = timeshift_OLA(filename, sample_rate,
                                overlap, fps, alpha)
```

De sample-rate staat voor het aantal frames per seconde waaruit het geluidssignaal bestaat en staat standaard op 44100. **DIT MOET ER EIGENLIJK NIET BIJ**

Het OLA-algoritme bestaat uit drie grote delen. Het eerste deel laadt het audiosignaal binnen en splitst het simpelweg in twee kanalen, een linker- en een rechtersignaal. Een audiosignaal is namelijk een vector van dimensie `nx2`, waarbij het aantal rijen `n` afhankelijk is van de lengte van de opname. De twee kolommen stelt het geluid voor van respectievelijk het linker- en rechtersignaal. Het programma past verder apart de code toe op beide signalen.

In een tweede gedeelte worden zowel het linker- als het rechtersignaal opgesplitst in frames van een vaste lengte `frame_size`. De lengte `frame_size` is gelijk aan de `sample_rate` (het aantal samples per seconde) gedeeld door de `fps` (het aantal frames per seconde) en heeft als eenheid samples per frame. Het algoritme gebruikt voor het opdelen van het audiosignaal een apart geschreven functie `make_frames` (cfr. section 0.9), waarvan ook het SOLA-algoritme gebruik maakt. De functie `make_frames` heeft als output een lijst `frames` waarvan elke rij de data bevat voor één van de aparte opgesplitste fragmenten. Om deze aparte fragmenten aan te maken, hanteert de functie een `for`-lus. Concreet zorgt array-slicing van het inputsignaal **(juiste terminologie?)** in elke `for`-lus voor de aanmaak van een fragment. Hierbij wordt meteen rekening gehouden met de gewilde overlap en dus zullen het begin en einde van opeenvolgende frames dezelfde data bevatten. Standaard staat de overlap op 50%, wat inhoudt dat de tweede helft van een eerste frame volledig dezelfde data bevat als de eerste helft van het daaropvolgende frame (zie ook fig. 1). Belangrijk is dat het algoritme bij de `for`-lus het concept preallocatie toepast. Preallocatie houdt in dat **je** op voorhand de juiste lengte bepaalt van de lijst waarvan in elke `for`-lus informatie wordt toegevoegd. Zo wordt tijdens elke iteratie de elementen van de lijst overschreven met deze informatie in plaats van toegevoegd zonder dat de lengte van de lijst verandert. Specifiek gaat het hier om de outputlijst `frames`. Deze lijst

heeft voor de lus enkel nullen, die elke iteratie overschreven worden met de data van één afgesplitst fragment. De snelheid van het algoritme verhoogt hierdoor aanzienlijk; het duurt niet langer 15 seconden om het programma uit te voeren, maar slechts 0,3 seconden.

Het derde en laatste deel bestaat erin om de opgesplitste fragmenten, aanwezig in de lijst `frames` (of `frames_left` en `frames_right`???), te recombineren, wat resulteert in de outputlijst `timeshifted_signal` die de data voor het tijdsverschoven geluidssignaal bevat. Dit deel omvat nogmaals een `for`-lus, waarbij ook hier preallocatie wordt toegepast. Concreet **worden** er tijdens elke iteratie twee indices bijgehouden, `index1` en `index2`. *'index1' is the index in the output signal where the overlap with the next-to-be-added frame (framel/framer) starts. 'index2' is the index of the last sample in the output singal, where the overlap stops of course.*

Fade-in en fade-out: Het algoritme werkt nu reeds "degelijk"maar een verbetering zou zijn om ...

0.3.3 Interpretatie

Positieve punten: beter dan resample. geen verandering van de toonhoogte.

Negatieve punten: - veel ruis zeker als geen crossfade functie. maar OLA maakt geen gebruik van de inhoud van de data/informatie over de pitch enzovoort, het past het algoritme op alle data toe...

0.4 Synchronous Overlap and Add (SOLA)

0.4.1 Concept

SOLA en OLA vertonen sterke gelijkenissen, maar het grote verschil is dat SOLA gebruikt maakt van kruiscorrelatie technieken. Dit zorgt voor minder ruis in de geluidsfragmenten. Het algoritme verschuift de blokken met de tijdsfactor α, en zoekt daarna naar gelijkenissen in het overlappingsgebied. Het algoritme zoekt de maximale gelijkenis en verschuift de blokken zodat deze overlappingen samenvallen. Het algoritme telt dan de uiteindelijk overlappingen weer op, zoals in het OLA-algoritme, met een crossfade.

Figuur 2: Principe van cross- en autocorrelatie.

0.4.2 Uitwerking

De kruiscorrelatie is een manier om gelijkenissen tussen twee geluidsgolven te vinden. Het is een veelgebruikte manier bij signaalverwerking om kleinere geluidsgolven in een langer sample te herkennen, wat leidt tot patroonherkenning. Hier leidt de kruiscorrelatie tot het vinden van de index waarbij de overlapping tussen de twee frames een maximale overeenkomst vertoont. De kruiscorrelatie wordt gegeven door :

$$(x_{L1} * x_{L2})[m] = \frac{1}{L} \sum_{n=0}^{L-m-1} x_{L1}^{*}[n] \cdot x_{L2}[n+m], 0 \leq m < L$$

met $x_{L1}[n]$ en $x_{L2}[n+m]$ de delen van $x_1[n]$ en $x_2[n]$ in het overlappende interval met lengte L. De index van de maximale overlap wordt gebruikt om de signalen te overlappen. In het algoritme gebeurt de bepaling van de maximale overlapping door voor elke mogelijke overlapping een getal, de correlatie, te berekenen. Dit is de som van alle elementen in een matrix, gedeeld door de lengte van de overlapping. De matrix is gelijk aan de element-per-element vermenigvuldiging van het eerste deeltje en het tweede overlappende deel. In de code staat dit als: **FIGUUR MOET NOG GEHERPOSITIONEERD WORDEN**

```
$correlation = sum(overlapping_part_of_output_signal .* ...
overlapping_part_of_frame) / length_overlap;
```

Als dit getal groter is dan `max_correlation`, de maximale waarde tot nu toe vervangen door de nieuwe maximale waarde, dan wordt de corresponderende `max_correlation_index` ook vervangen. Zo wordt de grootste waarde voor de kruiscorrelatie bepaald, alsook op welke positie van overlapping dit gebeurd. De twee delen worden dan zo verschuiven zodat het tweede stukje begint op de index van `max_correlation_index`. De frames worden opgeteld met een kruisfade en dit wordt herhaald voor alle frames. Dit wordt ook grafisch weergegeven in **grafiek???**

0.5 PSOLA

0.5.1 Concept

0.5.2 Uitwerking

0.5.3 Zero-rate crossing

Pitch detectie met auto-correlatie

Een meer robuuste manier om pitch detectie uit te voeren is door gebruik te maken van de autocorrelatie van een signaal. De autocorrelatie is gelijkend op de kruiscorrelatie in (?? verwijzing), maar bij de autocorrelatie wordt het algoritme toegepast op hetzelfde signaal. Een venster van twee maal de lengte van de langste periode wordt gebruikt om accuraat de pitch te bepalen. Als de shift in de auto-correlatie functie de fundamentele frequentie, bereikt de autocorrelatie een maximum. Dit maximum kan beschouwd worden als de pitch periode. Zo is het mogelijk om de pitch periode te bepalen van het signaal.

0.6 Vergelijking verschillende algoritmes

0.6.1 Frequentie en signaal plots

0.6.2 Enquete

Real-time implementatie

In het implementeren van de voorgaande algoritmes werd steeds gebruikt gemaakt van reeds opgenomen signalen, waarvan de karakteristieken gekend waren. In real-time-implementatie zijn de signaal karakteristieken ongekend en veranderen ze doorheen de tijd. Het signaal kan niet versneld worden omdat er niet in de toekomst kan gekeken worden en de daarna komende frames niet gekend zijn. Het signaal vertragen kan echter wel, omdat er stukjes van het gekende signaal moeten herhaald worden. Dit is de volgende stap die nog verder uitgewerkt moet worden in de volgende teamzittingen. Dit zal gebeuren in simulink. Wanneer het algoritme in simulink werkt, wordt het samengevoegd met de wim de vilder filter, die gemaakt werd door een ander team, om de ademhalingsruis eruit te halen.

Werkwijze en vakintegratie

0.7 Werkwijze

Elke PO zitting beginnen we met een korte vergadering. Hierin bespreken we eerst de eventuele huistaken en hoe deze verlopen zijn. Dan passen we indien nodig de algemene planning aan en stellen we de dagplanning op met de taakverdeling. Dit gebeurt in Asana, een handige webinterface diealle projecttaken makkelijk laat inplannen. Vervolgens maken we via Instagantt een Gant-chart voor een duidelijk overzicht. Als er moet geprogrammeerd worden proberen we in groepjes van twee te werken. Volgens het Pair Programming-systeem kan dan n persoon de code typen en de andere kijkt onmiddellijk na op fouten.

0.8 Vakintegratie

Bij dit project wordt de kennis uit verschillende vakken toegepast. Het project werkt met geluid. Een matrix stelt het geluid voor. Bewerkingen op matrices werd aangeleerd in het vak toegpaste algebra. Het frequentiespectrum kan berekend worden door een fourier transformatie. Dit werd gezien in het vak Informatie-Overdracht en Verwerking en in Elektrische Netwerken. Het schrijven van de algoritmes in Matlab vergt kennis opgedaan uit het vak methodiek van de informatica. In de algoritmes wordt ook gebruikt gemaakt van een crosscorrelatie die werkt met een convolutie. Dit staat uitgelegd in de cursus Algebra en Analyse. Daarnaast wordt de kennis van PO 1 en 2 toegepast in de vergaderingen.

Besluit

Bijlages

0.9 Bijlage A

0.10 Bijlage B

Hoofdstuk 1

Tijdelijke Commentaar

1.1 Informele inhoudstafel

Wat moet er zeker in het verslag?

1. Inleiding
    - Situering van het probleem in context
    - (Verband met literatuur/evt. vorige studies)
    - Heel kort opsommen van voornaamste besluiten
    - Structuur/opbouw van het verslag duiden
2. Middenstuk
    - Sample Rate change
    - Time Stretching Algorithms: OLA, SOLA, PSOLA, Pitch Detection
        (a) **Concept:** uitleg algemeen principe (conceptueel schetsen)
        (b) **Uitwerking:** code in MATLAB + duiding (specifieke uitleg)
        (c) **Interpretatie:** reflectie/interpretatie/besluit: is dit een goed algoritme of niet? Waarom wel/niet? Wat kan beter?
    - Real-time implementatie/Simulink
3. Slot
    - Integratie andere vakken

    - Reflectie planning
    - Conclusie
4. Referentielijst
5. Bijlages

1.2 Algemene opmerkingen

- Chapters vernederlandsen
- Cross-referenties goed krijgen
- Formule crosscorelatie
- Foto crosscorelatie
- Foto's invoegen
- titels terug toevoegen?
- interpretatie?
- Foto's op de juiste plaats zien te krijgen.
- zien wat er niet mee upgedatet is.