Stefan Richardt
Matr.nr.: 0331129

Benedikt Bengler
Matr.nr.: 0331028

Graz, den 12.4.2008

Seminararbeit

Algorithmen in Akustik und Computermusik 02

WS/SS 07/08

Thema:

Pitch Shifting & Time Stretching

O.Univ.Prof. Mag.art. DI Dr.techn. Robert Höldrich
DI Franz Zotter

Inhaltsverzeichnis:

Abstract:

In der folgenden Arbeit möchten wir mehrere Algorithmen zur Zeit - und Tonhöhenänderung von Audiomaterial vorstellen. Als Ausgangspunkt dient eine kurze geschichtliche Betrachtung, die die elektroakustische Realisierungen von Zeit - und Tonhöhenänderung vorstellt. Ausgehend von der grundlegenden Overlap and Add Methode werden dann mehrere Verbesserungen und Weiterentwicklungen dieses Grundprinzips beschrieben. Hierbei soll jeweils die Funktionsweise, die Implementierung in Matlab und die klanglichen Eigenschaften der einzelnen Varianten dargestellt werden.

1. Historische Betrachtung von Pitch Shifting und Time Stretching

Die Änderung von Zeit oder Tonhöhe spielen seit den Anfängen der elektroakustischen Tonaufzeichnung eine wichtige Rolle. Die erste Möglichkeit Tonhöhe und Abspieldauer zu verändern war die Variation der Abspielgeschwindigkeit bei Schallplattenwiedergabe. Die dadurch entstehende, gleichzeitige Veränderung von Dauer und Tonhöhe wurde bald schon ein wichtiges Hilfsmittel zur Manipulation von Audiomaterial. So wurden viele Effektsounds in den Anfängen des elektroakustischen Sounddesigns im Filmbereich mit Hilfe dieser Technik realisiert. Auf Grund der Größe und schwierigen Transportmöglichkeiten der ersten Recorder wurden aus einem Field Recording im Studio ein Vielzahl von Effektsounds hergestellt. So verwendete beispielsweise der amerikanische SFX Spezialist Robert Mott eine Aufnahme der Mongambi Wasserfälle um daraus Pistolenschüsse, Verkehrsgeräusche, Fluglärm oder die Explosion einer Atombombe zu gewinnen. Mit der Entwicklung des Re-Recordings wurde der Einsatzbereich der Audiomanipulation durch Tempovariation deutlich erweitert. So entstanden 1933 die Soundeffekte für King Kong auf Basis eines auf halber Geschwindigkeit abgespielten Löwengebrülls, das mit einer rückwärts abgespielten Tigeraufnahme gemischt wurde [1]. Auch in der elektronischen Musik wurde von Beginn mit Variation der Abspielgeschwindigkeit experimentiert. Neben der Verwendung von Schallplattenspielern führte ab Anfang der 50ger Jahre die Verbreitung der Tonbandaufzeichnung zur Erweiterung der technischen Möglichkeiten. So wurden die aufkommenden Tonbandmaschinen für den Einsatz in den Studios für elektronische Musik vielseitig modifiziert und erweitert. Ein Beispiel hierfür ist das "phonogène chromatique ", ein von Pierre Schaeffer entwickeltes Gerät, das es ermöglicht eine in sich geschlossene Bandschleife in 12 verschiedenen Geschwindigkeitsstufen wiederzugeben die so abgestuft sind, das sich eine Transposition über eine gleichmäßig temperierte Skala möglich ist. Das Gerät wurde über ein eine Oktave umfassendes Keyboard gesteuert. Allen angewendeten Verfahren war jedoch gemein, dass sie eine gemeinsame Änderung von Abspieldauer und Tonhöhe bewirken, da nur die Abspielgeschwindigkeit verändert wird. Mitte der 50ger Jahre wurde erstmals die unabhängige Änderung von Tonhöhe und Abspieldauer möglich. Das im Folgende beschriebene Verfahren wurde 1954 von Fairbanks, Everitt und Jaeger verwendet und wurde auch im zur gleichen Zeit entwickelten Gerät von Springer eingesetzt, das unter dem Namen akustischer Zeitregler, Zeitdehner oder Tempophon bekannt wurde. Hierbei handelt es sich um ein zusätzliches Gerät für die Tonbandwiedergabe, dass über mehrere, auf einer rotierenden Trommel angeordnete Tonköpfe verfügt. Das Magnetband wird an diesem rotierenden Kopf vorbei geführt. Neben der Geschwindigkeit des Tonbands kann auch die Rotationsgeschwindigkeit des Rotierkopfes geregelt werden. So ist es beispielsweise möglich das Band schneller

abzuspielen um eine Verringerung der Abspieldauer zu erreichen. Eine Erhöhung der Tonhöhe wird jedoch dadurch vermieden dass durch die Drehung des Rotierkopfes die Relativgeschwindigkeit zwischen Band und Magnetkopf gleich den Normalbetrieb gehalten wird. Dies führt dazu, dass Teile des Bandes ausgelassen werden. Das Ausgangssignal wird durch die Summation der Signale der einzelnen Tonköpfe gebildet. Ebenso ist beispielsweise beim Konstanthalten der Bandgeschwindigkeit eine Erhöhung der Tonhöhe durch Erhöhung der Relativgeschwindigkeit zwischen Magnetband und Kopftrommel möglich, wobei Teile des Tonbands doppelt wiedergegeben werden. Auf diese Weise können beide Parameter unabhängig voneinander erhöht oder verringert werden [2].

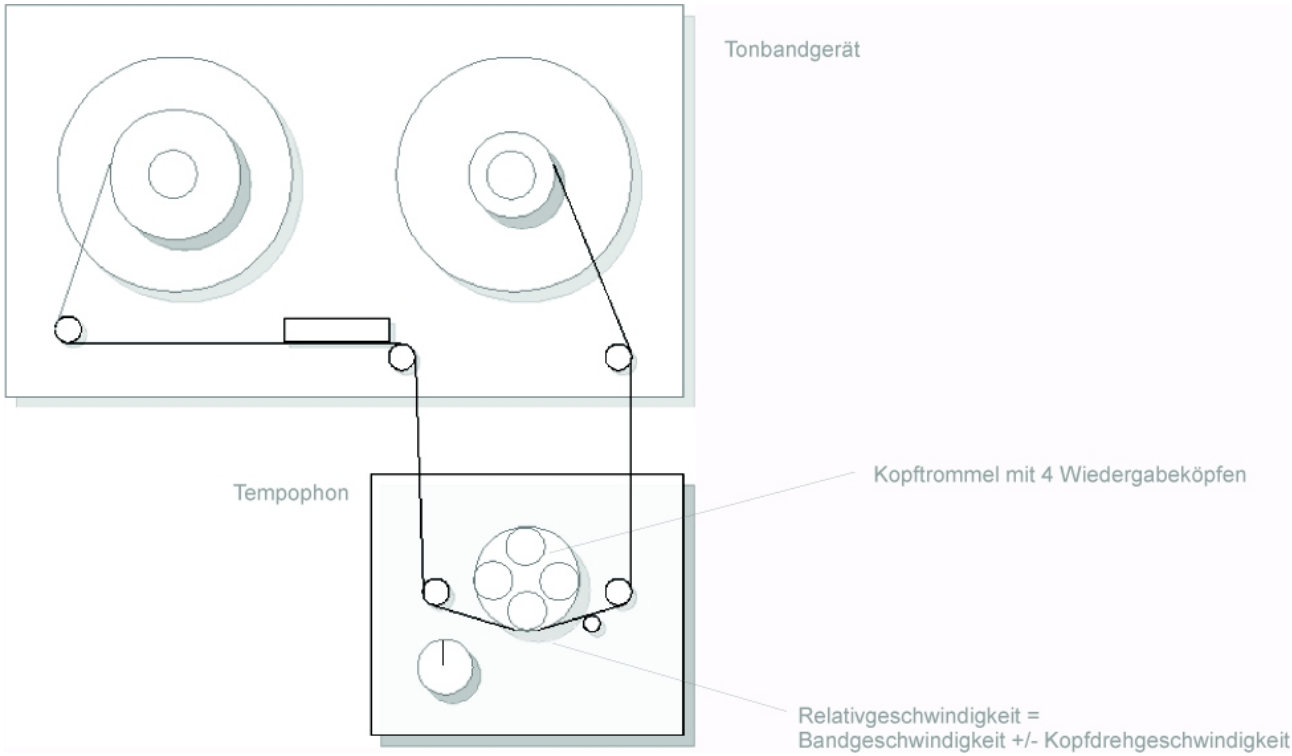


Bild 1: Funktionsprinzip der Springer Tempophons

2. Overlap and Add (OLA)

Die grundlegenste Realisierung eines Pitch/Timestrech - Algorithmus in digitaler Domäne entspricht weitgehend dem oben vorgestellten Prinzips des Tempophons. Für die Segmentierung des Signals, die beim Tempophon durch die Abnahme mit den rotierenden Köpfen erreicht wird, scheidet man das Ausgangssignal in sich überlappende Blöcke. Der Parameter hop size (x_hop) gibt an, wann der nächste Block beginnt. Soll nun das Gesamtsignal zeitlich um den Faktor a gedehnt werden, wird jeder einzelne Block um die nun mit dem Faktor a skalierte hop size (x_a) nach hinten verschoben. Dies entspräche beim Tempophon einer Rotation entgegen der Laufrichtung des Magnetbandes. Das

Überblenden zwischen den einzelnen Segmente wird beim Tempophon durch die Summation der Ausgangssignale der rotierenden Köpfe erreicht. Beim OLA - Algorithmus wird hierfür ein linearer Crossfade mit definierter Länge verwendet.

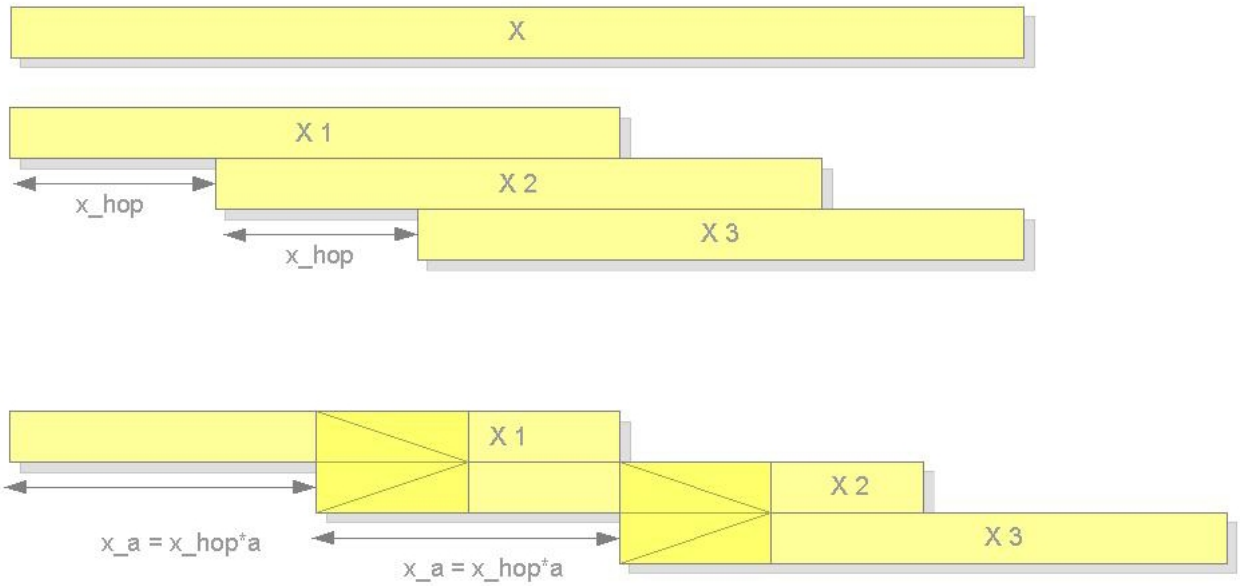


Bild 2: Overlap and Add

Ist a kleiner 1 werden die Segmente zusammen geschoben und so Teile des Originalsignals ausgelassen. Eine Tonhöhenänderung ohne zeitliche Änderung kann durch Resampling mit nachfolgender Zeitkorrektur erreicht werden. Soll die Tonhöhe um den Faktor b (b>1) erhöht werden, wird das Ausgangssignal mit der Samplingfrequenz $f_{pitch} = b * f_{old}$ re-gesampelt. Dieses Signal wird nachfolgend mit dem OLA - Algorithmus um den Faktor b zeitlich gedehnt.
Die Implementierung des OLA-Algorithmus stellt kein größeres Problem dar. Wir haben bei unserer Implementierung die Blocklänge N der einzelnen Segmente auf 3*x_hop festgelegt. Die Länge eines Fadebereichs beträgt x_a/2. Wie oben beschrieben werden die jeweiligen Signalteile mit einfachen linearen Fades zusammengefügt. Der Vektor x_out wird jeden Schleifendurchgang um das aktuelle Signalteil erweitert und stellt am Ende das Ergebnis dar.
Die hörbaren Artefakte hängen stark von der hop size und vom Eingangssignal ab. Bei hohem Stretchfaktor kann man eine Art Flattern hören. Bei einer hopsize von 2000 und einem Stretchfaktor a=1.8 wird das Signal mit einer Verzögerung von 1600 Samples (entspricht 36 ms) wiederholt, was das Flattern erklärt. Ebenso wahrnehmbar ist ein mit steigendem a stark ausgeprägter metallischer Klang.

3. Synchronous Overlap and Add (SOLA)

Hierbei handelt es sich um eine Erweiterung des oben beschriebenen OLA - Algorithmus. Die überlappenden Blöcke werden bezüglich Ähnlichkeiten im Signal analysiert. Der wesentliche Unterschied zum OLA-Algorithmus besteht in der Flexibilität der Verschiebung der Signalabschnitte zueinander. Block X 1 wird nicht mit konstanter Verschiebung an der Stelle a*x_hop in Block X 2 übergeblendet, vielmehr wird die Position der Signale und damit die des Fadebereichs jedes mal neu errechnet. Ziel ist es die Verschiebung so zu wählen, dass der Crossfade zwischen den Segmenten an der Position der größten Ähnlichkeit zu liegen kommt und so deutlich weniger wahrnehmbar ist.

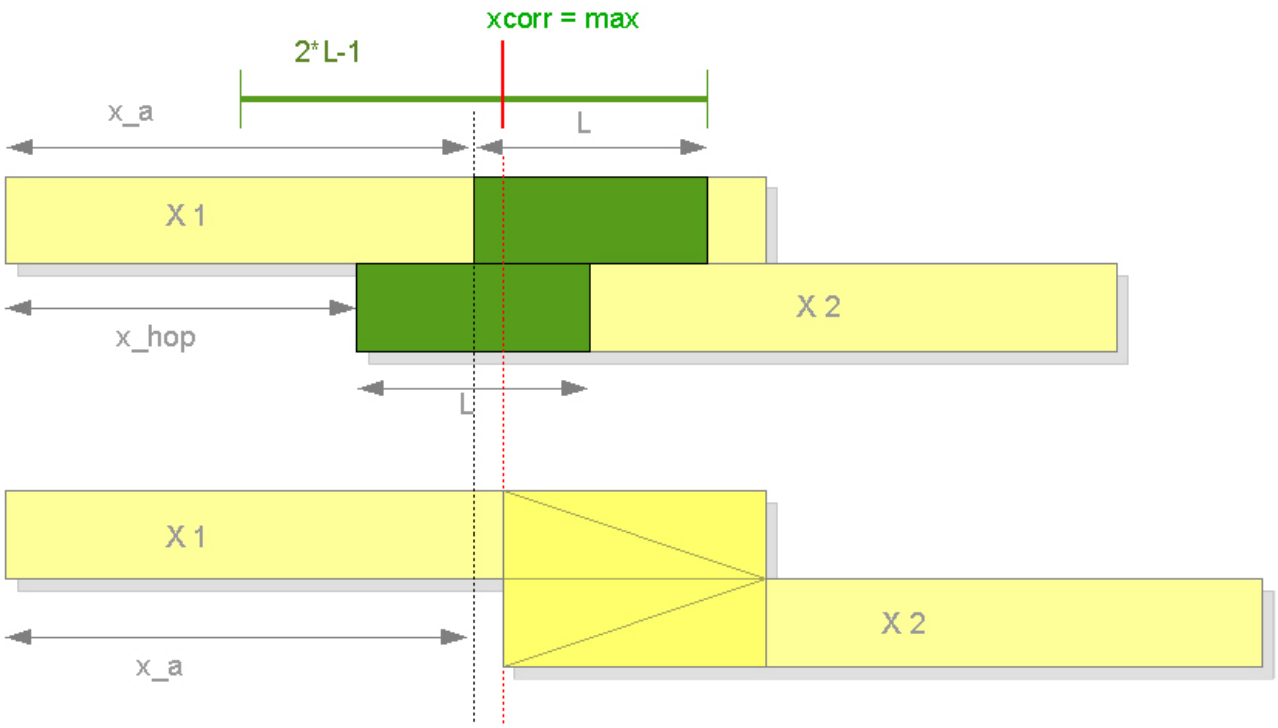


Bild 3:Funktionsweise Synchronous Overlap and Add

Im oberen Abschnitt von Bild 3 ist die Analyse der Segmente dargestellt. Hierbei wird ein Segment der Länge L zu Beginn des Blocks X 2 mit einem Segment der gleichen Länge, das im Block X 1 an der Stelle x_a = x_hop*a beginnt, kreuzkorreliert. Die Länge L ist mit x_a/2 festgelegt. Im erhaltene Kreuzkorrelationsvektor wird nun das Maximum ermittelt, das im grün gekennzeichneten Korrelationsbereich der Länge 2*L-1 liegt. Liest man den zum Maximum des Kreuzkorrelationsvektor gehörigen Index aus, kann die Position des Maximas in Bezug auf den Block X 1 bestimmt werden. Dies ist die ermittelte Startposition für den Block X 2. Die Fades der beiden Blöcke haben im Gegensatz zur OLA - Implementierung eine variable Länge. Diese wird für jeden Crossfadevorgang neu, in Abhängigkeit vom in der Analyse ermittelten Startwert des neuen Blocks ermittelt. Die Startpositionen der Blöcke können sowohl vor als auch nach der exakten Verschiebung x_a zu liegen kommen. Es zeigt sich jedoch, dass der geforderte Strechfaktor a über einen längeren Signalabschnitt mit einer gewissen Abweichung erreicht wird. Die Genauigkeit ist maßgeblich von der Länge des bearbeiteten Signals abhängig. Mit zunehmender Länge

wird durch die statistische Gleichverteilung der Abweichung vom tatsächlichen Stretchfaktor, dieser immer besser angenähert.
Wie zu erwarten sind auch die Artefakte ähnlich, wobei eine Verbesserung gegenüber SOLA durchaus hörbar ist. Bei ungünstiger Wahl der Parameter kann eine Art Phasing entstehen, dass durch Kammfiltereffekte hervorgerufen wird, die beim Überblenden zweier Signalteile an der Position der größten Gleichheit entstehen können.

4. Time Stretching und Pitch Shifting mit Berücksichtigung der Tonhöhe

Bei den vorgestellten Verfahren OLA und SOLA wird die Tonhöhe des bearbeiteten Signals nicht berücksichtigt. Sprachsignale und monophone Instrumente werden jedoch weitgehend durch ihre Tonhöhe charakterisiert. Der PSOLA - Algorithmus (Pitch-synchronous Overlap and Add) soll deshalb die Zuordnung der einzelnen Zeitsegmente auf Basis der ermittelten Tonhöheninformation vornehmen. Damit soll erreicht werden, dass die dem Signal zu Grunde liegende Frequenz nicht beim Zusammenfügen der Segmente beeinträchtigt wird. Hierfür ist jedoch eine Analyse des Eingangssignals nötig um daraus die Tonhöhe und die pitch marks zu ermitteln. Diese liegen äquidistant auf dem jeweiligen Maxima jeder Periode und dienen zur pitchangemessenen Segmentierung des Signals.

4.1 Pitch Mark Dedection durch Linear Prediction Coding

Im Folgenden soll ein Analyseverfahren vorgestellt werden, um die nötigen Pitchinformationen für den PSOLA - Algorithmus zu erhalten. Das Linear Prediction Coding ist einen Verfahren aus dem Bereich der Quelle-Filter-Signalverarbeitung. Die Grundlage dieses Teilbereichs ist die Aufspaltung eines Signals in einen Quelle - und einen Filterbestandteil. Der Filterbestandteil beinhaltet die spektrale Einhüllende des Signals wohingegen das Quellensignal zeitliche Informationen wie die Tonhöhe beinhaltet. Nach der Trennung können beide Bestandteile unabhängig voneinander bearbeitet und in der Synthesephase wieder zusammengefügt werden. Der Hauptanwendungsbereich des Quelle-Filter Modells ist die Spachsignalverarbeitung. So wird diese Prinzip beispielsweise in der Sprachsynthese genutzt um durch die Analyse von Sprachsamples Filtermodelle zu entwickeln die dem menschlichen Vokaltrakt entsprechen.
Die ursprünglichen Sprachlaute können durch eine angemessenen Anregung dieser Filter (Impulsfolge und Rauschen) synthetisiert werden. Für die Realisierung des PSOLA - Algorithmus möchten wir nun mit Hilfe des LPC Verfahrens die Tonhöheninformationen aus dem Quellensignal der Quelle - Filter Analyse erhalten.

Funktionsweise des LPC

Die Grundlage der LPC ist die Schätzung des aktuellen Werts eines Signals aus den vorhergegangenen Werten. Der Schätzer x'(n) wird mit Hilfe eines FIR Filters aus der Linearkombination der vergangenen Samples ermittelt.

$$x'(n) = \sum_{k=1}^{p} a_k * x(n-k)$$

Hierbei ist p die Ordnung des Filters und a sind die Filterkoeffizienten. Die Differenz aus Originalsignal x(n) und geschätztem Signal x'(n) liefert den Vorhersagefehler e(n).

$$e(n)=x(n)-x'(n)$$

Wird der Prediction Filter z transformiert ergibt sich:

$$P(z)=\sum_{k=1}^{p} a_k * z^{-k}$$

Die Gleichung 2 kann in der z Ebene angegeben werden als

$$E(z)=X(z)-X'(z)=X(z)*(1-P(z))$$

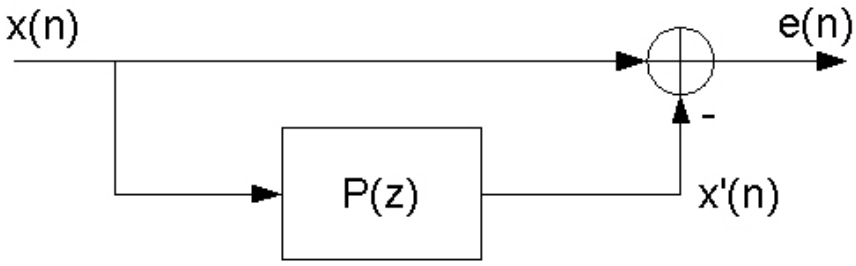


Bild 4: feed-forward prediction filter

Das Strukturbild zeigt die feed-forward Berechnung der Vorhersagefehlers e(n) mit dem Filter P(z).

Man geht nun von der Darstellung des Vorhersagefehlers in der z Ebene aus:

$$E(z)=X(z)*(1-P(z))$$

Es kann gezeigt werden, dass das Eingangssignal X(z) aus dem Vorhersagefehler E(z) und der invertierten Filterstruktur $1-P(z)$ wiederhergestellt werden kann.

$$X(z)=E(z)*\frac{1}{1-P(z)}$$

mit $\frac{1}{1-P(z)}=H(z)$ folgt: $X(z)=E(z)*H(z)$

Das IIR Filter H(z) wird als Synthesefilter bezeichnet und repräsentiert das spektrale Modell des Eingangssignals. Ein derartiges Filter kann nun genutzt werden, um mit einem geeignet nachgebildeten Fehlersignal e(n) das Originalsignal, dass der Analyse zu Grunde lag, zu synthetisieren.

Koeffizientenbestimmung des Synthesefilters

Um die Koeffizienten a_k des Filters zu bestimmen haben wir die Autokorrelationsmethode verwendet. Die Idee diese Methode ist es die Energie des Vorhersagefehlers zu minimieren. Das entspricht einer bestmöglichen Nachbildung der spektralen Einhüllenden des Signals durch den Filter.
Hierfür soll die Fehlerfunktion $J(a_k)$ des quadratischen Fehlers minimiert werden.

$$J(a_k)=E\{e(n)^2\}\rightarrow min$$

$$J(a_k)=E\{(x(n)-a_k^T*x(n))^2\}$$

Nach dem Lösen des Binoms und dem Ersetzen von

$x(n)*x(n)=r_{xx}$ (Autokorrelationsvektor)

$x(n)*x^T(n)=R_{xx}$ (Autokorrelationsmatrix)

ergibt sich für $J(a_k)$:

$$J(a_k)=E\{x(n)^2\}-2*a_k*r_{xx}+a_k^T(n)*R_{xx}*a_k$$

Um das Minimum zu bestimmen werden nun die partiellen Ableitungen nach a_k bestimmt und gleich Null gesetzt.

$$\nabla_{a_k} J(a_k)\Rightarrow 0$$

es ergibt sich:

$R_{xx}*a_k=r_{xx}$ bzw. $a_k=R_{xx}^{-1}*r_{xx}$

Um die Filterkoeffizienten a_k zu erhalten muss nun das so erhaltene Gleichungssystem, das einen Rang gleich der Anzahl der Filterkoeffizienten besitzt, gelöst werden. Dazu kann der Levinson-Durbin Algorithmus verwendet werden der die Koeffizienten rekursiv berechnet. In Matlab existiert der Befehl LPC auf Basis der Levinson-Durbin Rekursion. Berechnet man über die LPC die spektrale Einhüllende eines Signals und vergleicht diese mit dessen Spektrum zeigt sich, dass der Gain noch angepasst werden muss. Bei der Autokorrelationsmethode ist der Gainfaktor für die spektrale Einhüllende definiert als:

$$g=\sqrt{r_{xx}(0)-E\{a_k^T*r_{xx}\}}$$

Aus dem mit g gewichteten Synthesefilters H(z) kann nun die spektrale Einhüllende und der Vorhersagefehler bestimmt werden.
Die folgenden Bilder zeigen die ermittelte spektrale Einhüllende des Signals x(n). Diese wurden für verschiedene Filterordnungen p ermittelt und mit dem tatsächliche Spektrum überlagert.

Bild 5: Spektrale Einhüllende mit Filterordnung p=20, p=50, p=100

Die folgenden Bilder zeigen den Zeitverlauf des zu analysierenden Signals x(n) und den Vorhersagefehler e(n).

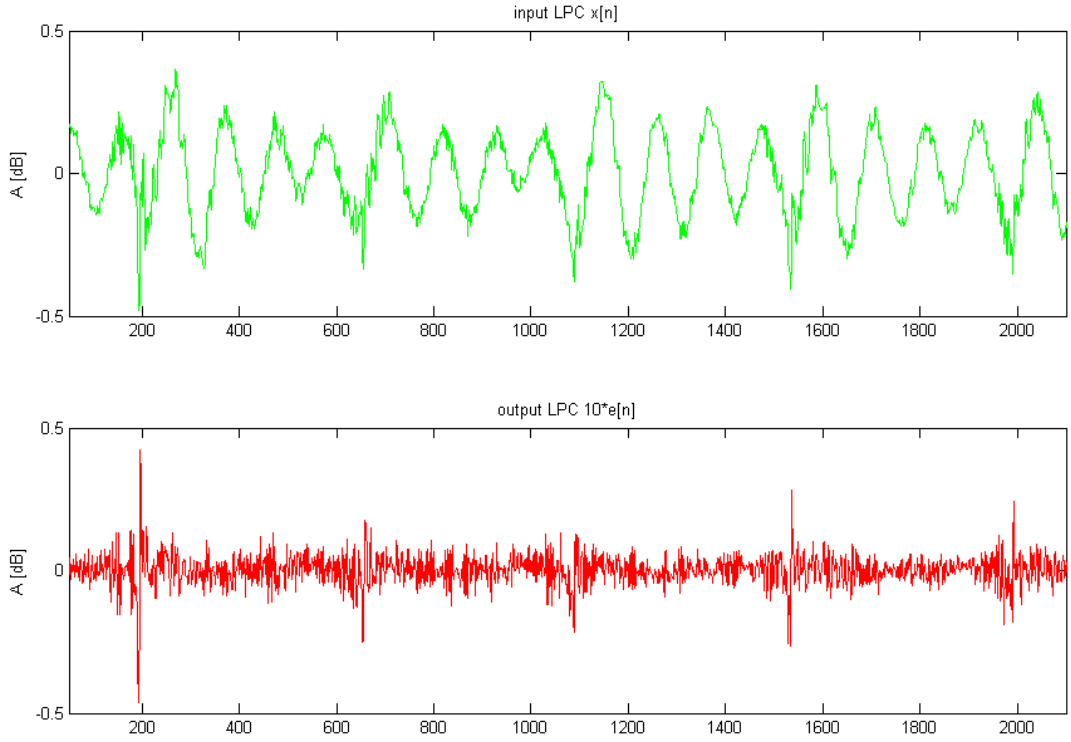


Bild 7: Signal x(n), Vorhersagefehler e(n)

Es zeigt sich, dass das Zeitsignal des Vorhersagefehlers e(n) ausgeprägte Peaks mit der zur Grundfrequenz des analysierten Signals gehörigen Periode aufweist.

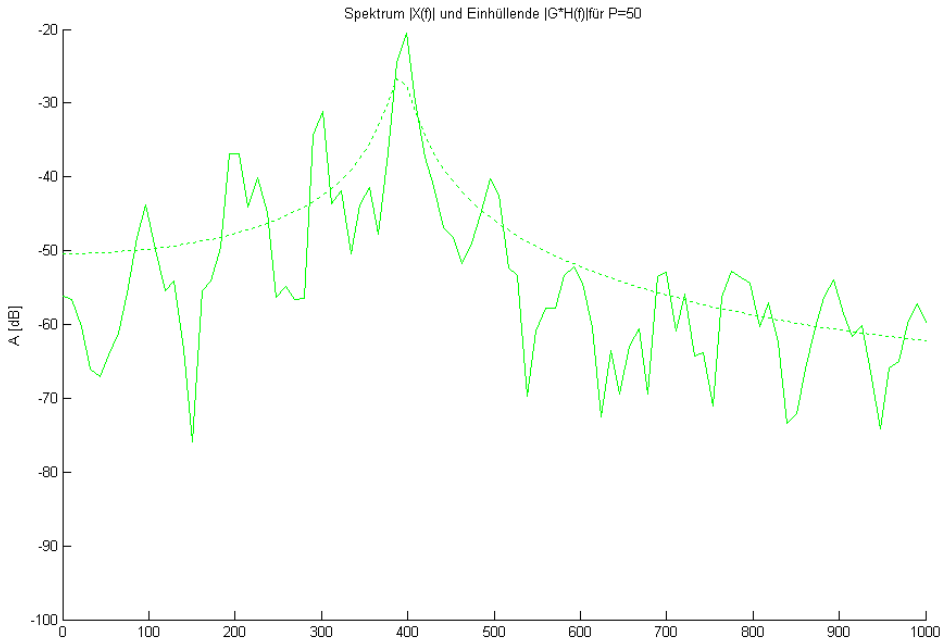


Bild 8: Orginalspektrum und Einhüllende p=50

Interessant ist, dass die Grundfrequenz also der Pitch nicht dem absolute Maximum der FFT entsprichen muss. Während das absolute Maximum bei ca. 400 Hz, dem Grundton des Signals liegt, befindet sich das lokale Maximum mit der niedrigsten Frequenz bei 100 Hz. Der PSOLA - Algorithmus benötigt neben dem Eingangssignal die Pitchmarks. Anhand eines Pitchmarkvektors M, in dem die Zeitwerte aller Pitchmarks in Samples enthalten sind, wird das Signal später segmentiert und gedehnt. Es gilt nun aus dem Vorhersagefehler e[n] die Pitchmarks zu berechnen und sie in den oben beschriebenen Vektor M zu schreiben.

Aus dem vorliegendem Fehlersignal e[n] der LPC sollen die lokalen Maxima detektiert und ihre Positionen gespeichert werden. Es gilt jene Maxima zu erkennen, die sich mit einer nur leicht veränderlichen Grundfrequenz periodisch wiederholen. Um dies zu erreichen wird der Absolutbetrag des Fehlersignal gebildet und dieser auf Eins normiert. Um nicht relevante Signalanteile auszublenden wird eine Threshold definiert, die sich aus der Signalenergie bzw. dem Durchschnittswert des Fehlersignals e[n] berechnet. Alle Werte unter der Threshold werden Null gesetzt. Der Ergebnisvektor enthält die gewünschte Maxima, die jedoch noch nicht eindeutig identifizierbar sind. Wie aus der Graphik ersichtlich, besteht ein Maxima aus einer unbestimmten Anzahl von Werten. Somit lässt sich vorerst kein konkreter Zeitpunkt festhalten.

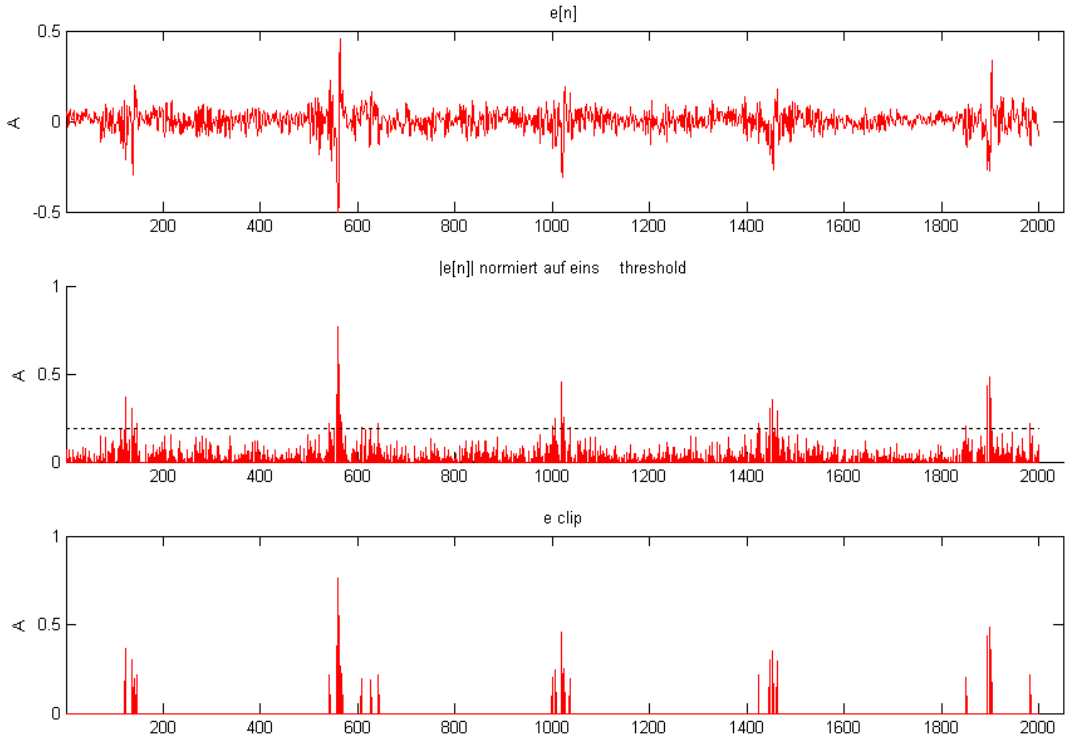


Bild 9: e(n), |e(n)|, e_clip

Abbildung eins zeigt das unbearbeitete Fehlersignal. Die zweite Abbildung zeigt den Betrag, des auf eins normierten Fehlersignals und die Threshold th. In der dritten Abbildung sieht man den Vektor e_clip, bei dem alle Werte unter der Threshold durch Null ersetzt wurden.

Um einen eindeutigen Zeitpunkt zu fixieren wird vorerst die Periodendauer bestimmt, indem die Nullen zwischen den bestehenden Werten gezählt werden. Der längste Abstand wird als vorläufige Periodendauer angenommen. Ausgehend vom absolutem Maximum des Signalabschnitts kann man nun um die Periodendauer verschoben eindeutig identifizierbare Punkte festlegen. Diese entsprechen jedoch nicht exakt den tatsächlichen Pitchmarks. Es ist möglich sie zu verfeinern, indem man in einem definiertem Bereich um den vorläufigen Pitchmarks nach den tatsächlichen Maxima sucht. Man teilt das Signal also in Segmente , die der Periodendauer entsprechen und tastet es nach lokalen Maxima des Signals e_clip ab. Dieses Signal e_clip (siehe Abbildung 3) beinhaltetet bereits die Maxima mit ihren umliegenden Peaks, der Rest wurde Null gesetzt. Befinden sich im jeweiligen Fenster nur Nullen, bleibt der ursprüngliche Pitchmark erhalten. Wird ein Maxima gefunden wird der alte Pitchmark überschrieben. Die nun entstehenden Pitchmarks sollten den tatsächlichen entsprechen.

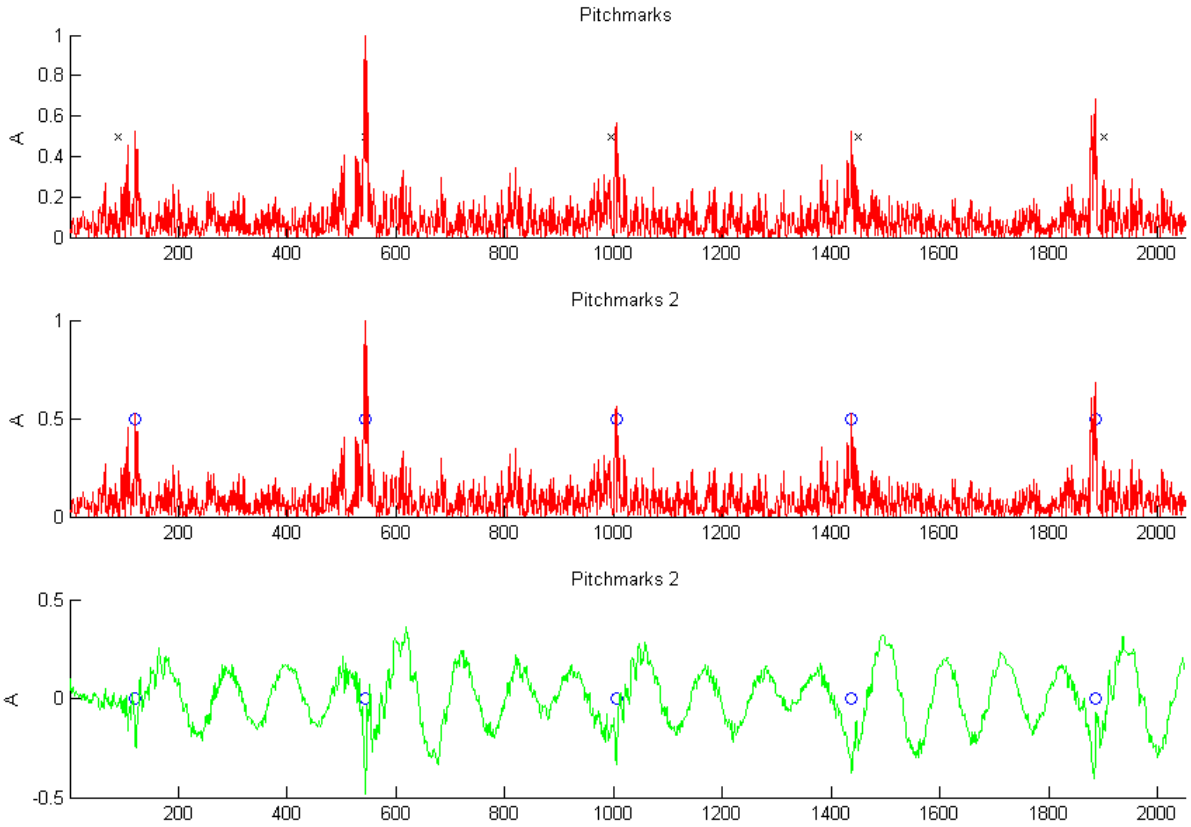


Bild 9: e(n) mit vorläufigen Pitchmarks, e(n) mit verbesserten Pitchm., x(n) mit verbesserten Pitchm.

Die erste Abbildung zeigt die vorläufigen Pitchmarks. Da sie sich aus der Periodendauer und dem absoluten Maximum errechnen, stimmen tatsächliches Maxima und Pitchmark nur an der Stelle des absoluten Maximas überein. Die zweite und dritte Abbildung zeigen die verfeinerten Pitchmarks mit dem Fehlersignal e[n] und dem Eingangssignal x[n].
Problematisch wird dieser Verfeinerungsschritt, wenn die errechnete Periodendauer sich stärker von den tatsächlichen Pitchmarks unterscheidet. Ein Beispiel: Ist die errechnete Periodendauer 400 Samples und sind die lokalen Maxima ca. 330 Samples voneinander

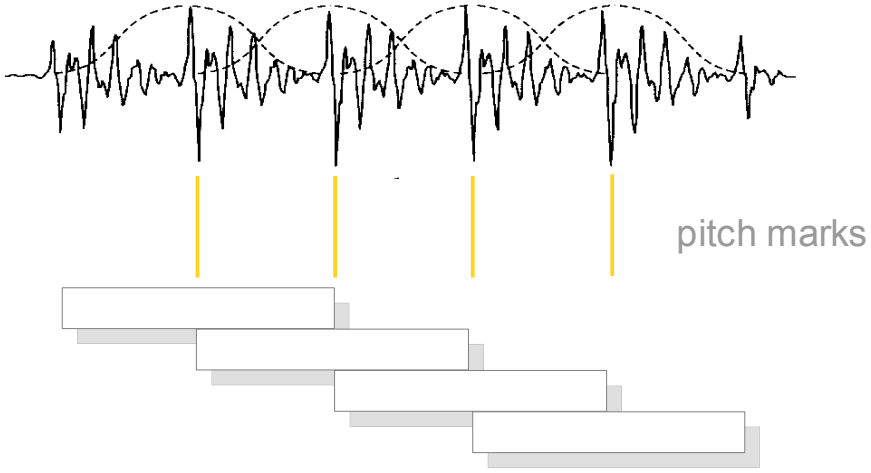
entfernt, so wird der Algorithmus ausgehend vom höchsten Maxima schon zwei Pitchmarks entfernt einen Fehler produzieren. Die Fensterbreite soll der halben Periodendauer, also 200 Samples entsprechen. Gesucht werden Pitchmarks um die Punkte 400, 800, 1200 ... , tatsächlich befinden sie sich bei 330, 660, 990 ... . Beim zweiten Pitchmark, an der Stelle 800 würde der falschen Pitchmark bestehen bleiben, da im Bereich des Fensters kein Maximum sondern nur Nullen vorzufinden sind. Tritt dies ein, so wird der Pitchmark als potentieller Fehler markiert. Für diesen Pitchmark wird nun erneut nach einem Maximum gesucht, jedoch nicht vom absoluten Maximum ausgehend sondern vom letzten korrekten Pitchmark. In unserem Fall ist an der Stelle 800 ein Fehler markiert. Der letzte korrekte Pitchmark befindet sich an der Stelle 330. Mit der berechneten Periodendauer von 400 Samples wird der Pitchmark nun um den Punkt 730 gesucht und der richtige Pitchmark kann gefunden werden. Für alle weiteren Fehler wird der Vorgang wiederholt. Ein weiterer häufiger Fehler sind zwei kurz aufeinander folgende Pitchmarks. In einem Postprocessing werde alle Pitchmarks die weniger als eine bestimmte Anzahl Samples vom nächsten entfernt sind gelöscht. Weitere Verbesserungen wie z.B. das Festlegen von einem maximalem Abstand wären möglich.

In der Praxis zeigt sich, dass der oben beschriebener Algorithmus nur bedingt funktioniert. Prinzipiell ist eine Pitchmarkdetection nur unter bestimmten Voraussetzungen sinnvoll. Bei der Analyse von Sprache unterscheidet man zwischen stimmhaften und stimmlosen Signalanteilen. Bei einem stimmlosen Signal ist der Rauschanteil hoch und eine Pitchmarkdetection nicht möglich. Die Pitchmarkdetection solcher Signale gibt keine brauchbaren Ergebnisse. Ein komplexes Sprachsignal besteht sowohl aus stimmlosen als auch stimmhaften Signalabschnitten. Es wäre also sinnvoll dem Algorithmus nur stimmhafte Signalabschnitte zu übergeben. Für die stimmlosen Anteile könnten Pitchmarks mit einer vorher festgelegten, konstanten Periodendauer gesetzt werden. Voraussetzungen dafür ist die Trennung in stimmhafte und stimmlose Signalanteile, dessen praktikable Implementierung ein nicht triviales Problem darstellt.

Die größte Schwäche des Algorithmus ist die teilweise auftretende, sprunghafte Änderung der Periodendauer. Grund dafür ist die notwendige Segmentierung des Eingangssignals. Der Algorithmus wird für jedes Segment durchgeführt und die resultierenden Pitchmarks später für das ganze Signal in einem Vektor gespeichert. Es kommt vor, dass die Periodendauer, eines Segments der doppelten oder halben Periodendauer des vorherigen Segments entspricht. Zum einen entsprechen die Pitchmarks nicht mehr der Grundfrequenz, zum zweiten ist die sprunghafte Änderung der Periodendauer beim später implementierten PSOLA-Algorithmus deutlich als Artefakt hörbar.

4.2 PSOLA Pitch Synchronous Overlap and Add

Die Grundlage des Algorithmus bilden die Informationen aus der vorangegangenen Pitchmarkdedection. Es wird das zu bearbeitende Signal mit seinen entsprechenden Pitchmarks eingelesen. Durch Differenzieren des Pitchmarkvektors M erhält man den Vektor P, der die zeitabhängige Pitchperiode enthält. Im Optimalfall sollte die Periodendauer nur leicht schwanken und über mehrere Pitchmarks konstant sein. Das Eingangssignal wird in N Segmente zerlegt, wobei N der Anzahl der Pitchmarks entspricht. Jedes Pitchmark bildet den zeitlichen Mittelpunkt eines Segments, wobei es mit einem hanning window der Länge 2*P gefenstert wird.



Das Verlängern/Verkürzen des Signals wird realisiert, indem man einzelne Segmente mehrfach abspielt oder verwirft.

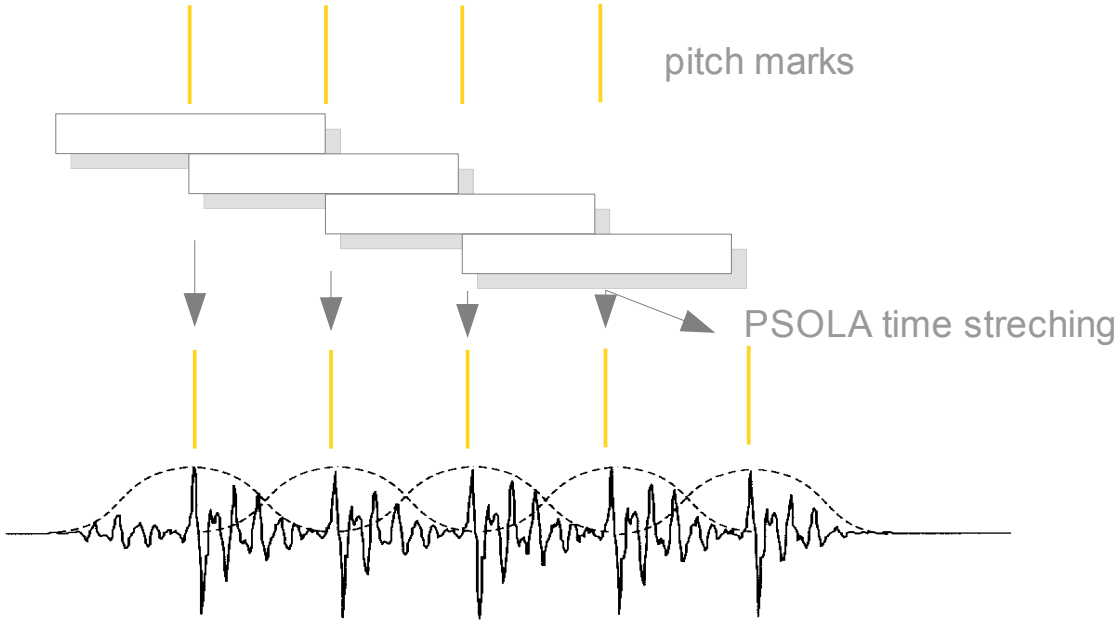


Bild 10: Funktionsweise PSOLA

Ist der Strechfaktor a größer als eins, werden bestimmte Segmente mehrfach wiederholt. Bei a<1 werden Segmente verworfen um auf die gewünschte Ausgangslänge zu kommen. Im Folgendem gehen wir davon aus, dass a>1 ist, das Signal also gedehnt wird. Die Auswahl der zu wiederholenden Segmente hängt ausschließlich vom Faktor a und den Längen der einzelnen Segmente ab. Der Psola Algorithmus entscheidet für jedes Segment neu, ob dieses wiederholt oder das nächste Segment angefügt wird. Dies geschieht mittels der Funktion $|a*M-t_k|$, die den zeitlichen Fehler ermittelt. M ist der Pitchmarkvektor in dem die Positionen aller Pitchmarks stehen. a*M entspricht den idealen Positionen der

zeitgedehnten Pitchmarks. t_k entspricht dem Wert, um den das nächste Segment gesetzt wird. Dieser Wert wird beim Hinzufügen eines jeden Segments neu bestimmt. Er berechnet sich aus der Addition des vorangegangenem t_k und der halben Fensterbreite des vorher gesetzten Segments. Durch Bestimmung des Minimums von $|a*M-t_k|$ kann ermittelt werden, mit welchem Segment die nächsten ideale Position eines Pitchmarks mit kleinst möglichem Fehler erreicht werden kann. Beim Timestretchen ergeben sich in der Regel nur zwei Möglichkeiten. Entweder wird das letzten Segments wiederholt oder das Nächste hinzugefügt.
Der PSOLA-Algorithmus wird maßgeblich durch die Qualität der Pitchmarkanalyse bestimmt. Wie oben bereits erwähnt, muss zwischen stimmlosen und stimmhaften Signalanteilen unterschieden werden. Die Abstände der einzelnen Pitchmarks dürfen lediglich leicht schwanken, keinesfalls jedoch große Sprünge aufweisen. Die Artefakte die durch ein Springen der Periodendauer hervorgerufen werden, sind deutlich hörbar und werden als unnatürlich und störend empfunden.

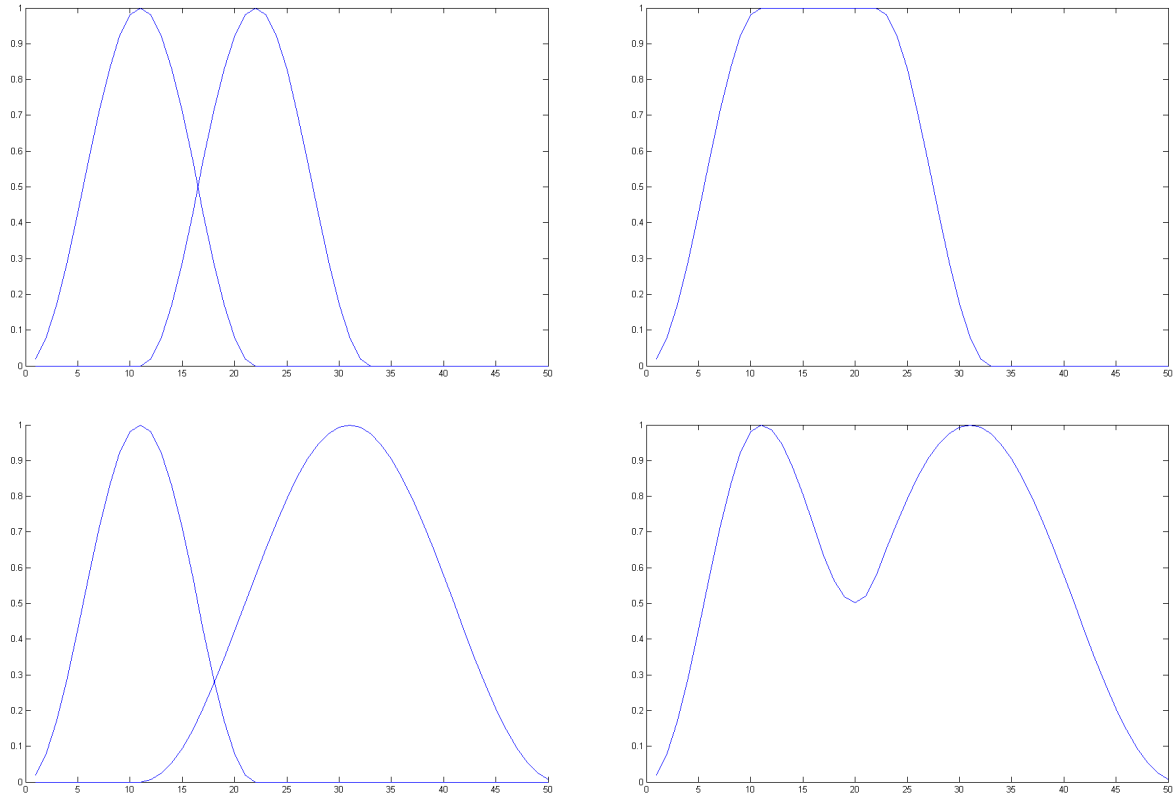


Bild 11: Auswirkung unterschiedlicher Fensterbreiten bei PSOLA

Die Ursache liegt in der unterschiedlichen Fensterbreite. Die Summe von zwei nebeneinander liegenden Fenstern soll eins ergeben. Der PSOLA-Algorithmus errechnet die Fensterbreite aus dem Abstand vom vorherigen zum aktuellen Pitchmark. Vergrößert sich wie in Bild 3 gezeigt der Abstand, vergrößert sich auch die Fensterbreite. Die Summe der beiden Fenster ist dem zu Folge kleiner eins. Verkleinert sich der Abstand der Pitchmarks ist die Summe der Fenster größer eins. Dies ist auch der Grund für eine Trennung des Signals in stimmhafte und stimmlose Anteile. Für stimmlose Anteile wird eine konstante Periodendauer festgelegt, um die oben beschriebenen Probleme zu

umgehen.

4.3 Linear Prediction PSOLA (LP-PSOLA)

Ein grundlegendes Problem aller bisher vorgestellten Methoden ist, dass es bei einer Änderung der Tonhöhe auch zu einer Verschiebung der spektralen Einhüllenden kommt. Eine Auswirkung hiervon ist der bekannte "Mickey-Mouse" Effekt. Wird ein Sprachsignal schneller abgespielt kommt es durch die Verschiebung der Formanten zu einer sehr unnatürlichen Klangverfärbung. Um dies zu vermeiden wird beim LP-PSOLA das Eingangssignal durch eine LPC-Analyse in das Fehlersignal e(n) und die spektrale Einhüllende H(z) aufgespalten. Die Tonhöhenänderung wird nun mit dem bereits vorgestellten PSOLA - Algorithmus am Fehlersignal e(n) durchgeführt, das die Tonhöheninformation enthält. Wird dieses nun mit dem ermittelten Synthesefilter H(z) gefiltert erhält man ein Ausgangssignal, dass zwar in der Tonhöhe verändert ist aber die selbe Formantstruktur wie das Eingangssignal aufweist.

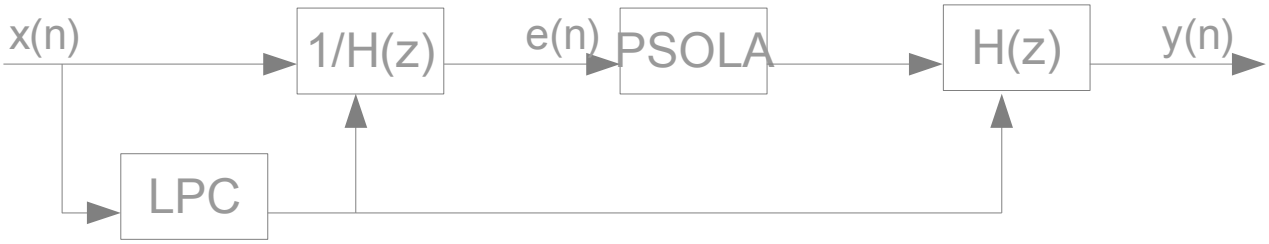


Bild 12: Funktionsprinzip LP-PSOLA

Implementierung LP-PSOLA

Da es nicht ausreicht eine einzige spektrale Einhüllende für das Eingangssignal zu bestimmen, da sonst die zeitliche Veränderungen des Spektrums nicht abgebildet werden, muss die LPC Analyse in Blöcken durchgeführt werden. Das Eingangssignal x(n) wird in Blöcke mit der Länge N segmentiert. Jeder dieser Blöcke wird mit LPC analysiert und die jeweiligen Impulsantworten des LPC-Filters in eine Matrix zwischengespeichert. Die Fehlersignale der einzelnen Blöcke werden aneinander gefügt und dienen als Eingangssignal für den bereits beschriebenen PSOLA - Algorithmus. Dieser führt die Tonhöhenveränderung durch Time Streching mit anschließendem Resampling durch. Das Ausgangssignal wird nun mit der hop size N in überlappende Segmente geschnitten. Diese werden nun mit der Inverse ihres entsprechenden Analysefilters gefiltert, das der spektralen Einhüllenden entspricht. Mit Overlap and Add werden die Segmente nun zusammengefügt wobei mittels Crossfades zwischen den unterschiedlich gefilterten Segmenten überblendet wird.

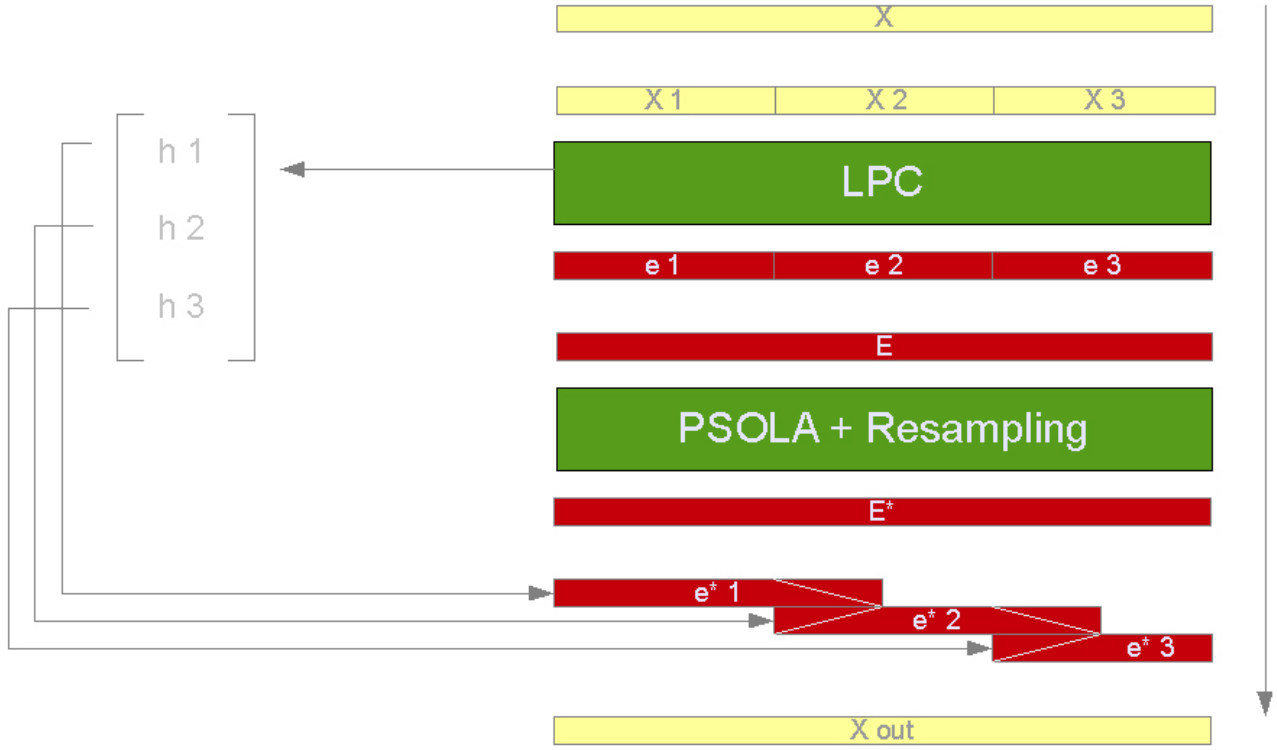


Bild 13: Implementierung LP - PSOLA

Formanterhaltung durch LP-PSOLA

Das erste Bild zeigt das Spektrum eines monophonen Sprachsignals. Das zweite Bild zeigt das Spektrum nach der Tonhöhenänderung durch den PSOLA - Algorithmus. Man sieht deutlich, dass sowohl die Grundfrequenz als auch die Formantenstruktur nach oben verschoben wird. Das dritte Bild zeigt das Spektrum des mittels LP-PSOLA bearbeiteten Signals. Hierbei wird lediglich der Grundtonbereich verschoben, das Formantspektrum entspricht hingegen dem des Orginalsignals. Der Unterschied ist deutlich hörbar und die unnatürliche Klangfärbung durch Formantverschiebung kann so weitgehend vermieden werden.

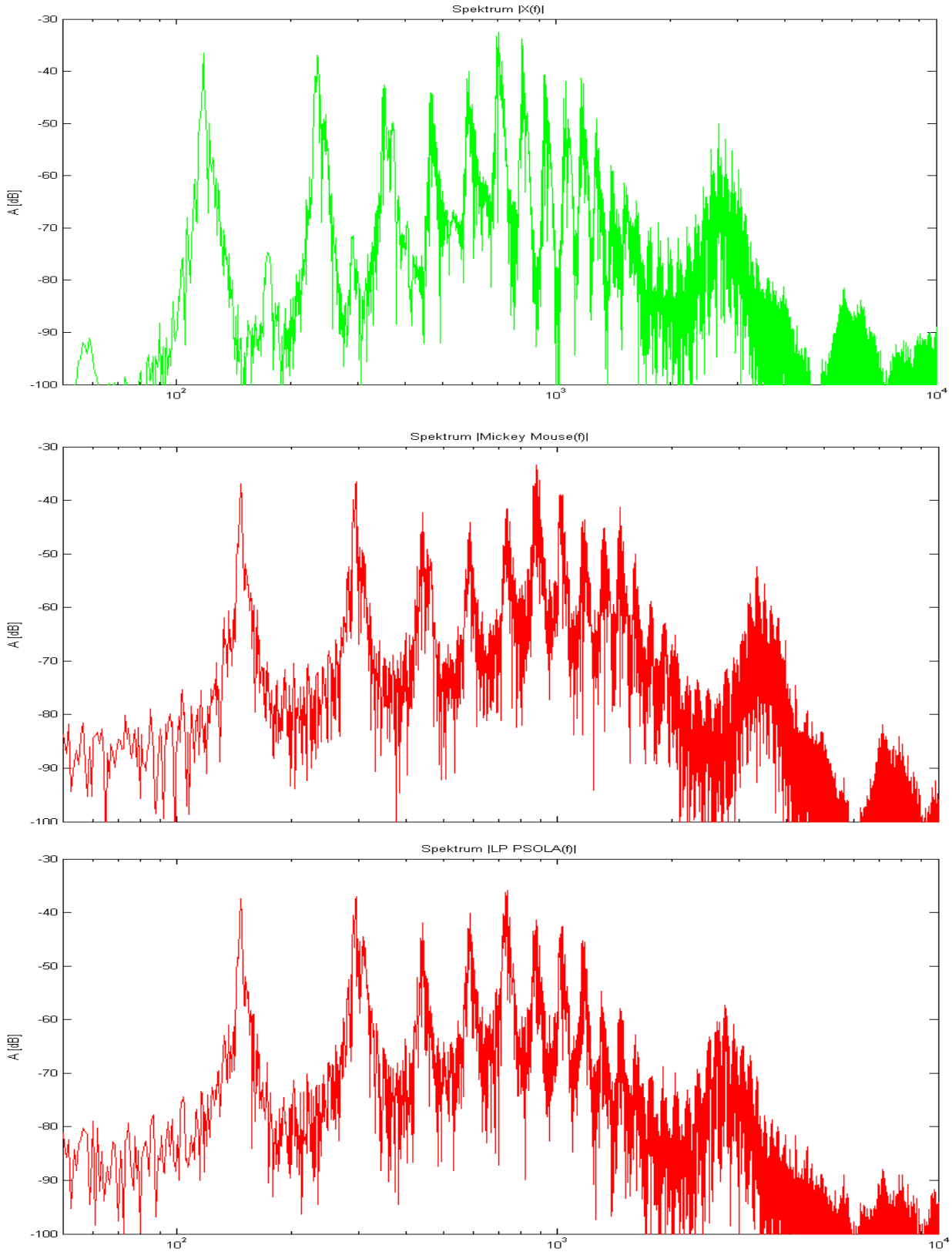


Bild 14: Formanterhaltung durch LP-PSOLA

5. Eingabe-/Ausgabeparameter und Zusammenhänge der Matlab - Funktionen

ola — Die Funktion ola benötigt das Eingangssignal, die hop size und den Strechfaktor a. Ausgegeben wird das zeitgedehnte Signal.

sola — Benötigt die selben Eingabeparameter wie ola, berechnet das Ausgangssignal jedoch mittels Synchronous Overlap and Add.

lpc_out — Benötigt ein Eingangssignal welches in x_pre und x_in unterteilt ist, wobei x_pre so lang wie die Filterordnung sein muss. Weitere Eingabeparameter sind die Filterordnung der LPC, die Länge des Eingangsignals und die Samplingfrequenz. Der Algorithmus führt die LPC aus und berechnet die Pitchmarks für den Signalabschnitt. Ausgegeben werden unter anderem die spektrale Einhüllende, sowie die Impulsantwort der LPC, das Fehlersignal und der Pitchmarkvektor.

catch_M — Da die LPC nur für einen kürzere Signalabschnitt sinnvoll ist, muss sie mehrfach ausgeführt werden. Zu übergeben sind das komplette Eingangssignal, die Filterordnung p , die Blocklänge N und die Samplefrequenz. Ausgegeben werden der Pitchmarkvektor, das Fehlersignal, und eine Matrix, die alle Impulsantworten enthällt.

Psola — Der Eingangsvektor wird mittels Pitchmarkvektor um den Faktor a gedehnt. Ausgegeben wird das gedehnte Signal.

lp_psola — Neben dem Eingangsvektor, den Pitchshiftfaktoren und der Filterordnung sind x_hop, die Segmentlänge der Pitchmarkanalyse und block_N, die Blocklänge der LPC einzugeben. Ausgegeben wird das pitchgeshiftete Signal mit und ohne formantsaving und die Matrix mit allen Impulsantworten der LPC.

algo_2 → OLA
→ SOLA

lpc_test → psola → catch_M → lpc_out
→ lpc_out

test_lp_psola → lp_psola → psola → catch_M → lpc_out

algo_2 testet ola oder sola durch einfachen Aufruf dieser. Lpc_test führt entweder eine einmalig lpc_out aus oder ruft catch_M auf, die ihrerseits das Signal zerteilt und die LPC für alle Signalabschnitte berechnet. Die komplette Pitchmarkdetection, dessen Implementierung auf dem Fehlersignal der LPC beruht wird ebenfalls in Catch_M ausgeführt. Psola greift zwar nicht direkt auf catch_M zu, benötigt jedoch die Pitchmarks aus catch_M. Der letzte Schritt ist die lp_psola, die alle Funktionen benötigt.

6: Quellenangabe:

[1] Ray Brunelle: The Art of Sound Effects. Experimental Musical Instruments Volume 12 , issue 1 and 2 Sept , Dec 1996

[2] Hans Ulrich Humpert: Elektronische Musik, Geschichte - Technik - Komposition. Schott 1987

[3] Udo Zölzer (Hrsg.): DAFX - Digital Audio Effects. John Wiley & Sons 2002

[4] F.M. Giménez de los Galanes, M.H. Savoji, J.M. Prado: Speech synthesis system based on a variable decimation/interpolation factor. IEEE 1995

[5] Alan V. Oppenheim, Roland W. Schafer, John R. Buck: Zeitdiskrete Signalverarbeitung. Pearson Studium 2004

[6] G. Moschytz, M. Hofbauer: Adaptive Filter. Springer 2000