

Schlussbericht der Studienarbeit WS 2001/2002

Time Domain Harmonic Scaling

Marti Adrian

1. März 2002

Inhaltsverzeichnis

	Aufgabenstellung	ii
1	Einleitung	1
2	Eingangssignal	2
3	Autokorrelation	3
	3.1 Vorgehen	3
	3.2 Resultate der Autokorrelation	4
4	Autokorrelation mit Filterung	5
	4.1 Resultate der Autokorrelation mit Filterung	6
5	Autokorrelation mit Center Clipping	7
	5.1 Die Berechnung der Envelope	7
	5.2 Die Durchführung des Center Clipping	7
	5.3 Resultate	8
	5.4 Resultate der Autokorrelation mit Center Clipping	9
6	Die schnellere Wiedergabe	10
	6.1 Vorgehen	10
	6.1.1 Grundperiodenbestimmung	10
	6.2 Verkürzung des Signals	11
	6.3 Resultate der Wiedergabe	12
	6.4 Resultate der Autokorrelation	14
7	Auswertung	15
	7.1 Verwendete Signale	15
8	Schlusswort	16
9	Literaturverzeichnis	17
A	Listings Matlab	18
B	Listings C++	24
C	File-Struktur	31

Aufgabenstellung

Studienarbeit für Herrn Adrian Marti

Time Domain Harmonic Scaling

Unter Time Domain Harmonic Scaling versteht man ein Verfahren, das die Wiedergabegeschwindigkeit eines Sprachsignals bei gleich bleibender Stimmlage ändert. Für eine schnellere Wiedergabe werden dazu aufeinander folgende Segmente eines digitalen Sprachsignals geeignet gewichtet und additiv überlappt. Für eine langsamere Wiedergabe werden einzelne Segmente wiederholt, wobei auch in diesem Fall eine geeignete Gewichtung und Überlappung angewendet wird, damit sich zwischen den Segmenten sanft Übergänge ergeben.

Die schnellere Wiedergabe von Sprachsignalen ist in vielen Fällen nützlich; z.B. zum Auffinden einer bestimmten Aussage in einem längeren Interview oder beim Abhören der Aufzeichnungen eines Anrufbeantworters.

Im einfachsten Fall werden Segmente fester Länge in regelmässigen Abständen überlappt. Dadurch lassen sich je nach Häufigkeit der Überlappungen verschiedene Wiedergabegeschwindigkeiten erreichen. Die Wiedergabequalität wird in diesem Fall jedoch durch zwei Effekte beeinträchtigt. Einerseits stören die festen Segmentlängen die Periodizität in stimmhaften Signalabschnitten, was sich in einem rauhen Klang bemerkbar macht. Andererseits wirkt auch der natürliche Eindruck des Sprachsignals beeinträchtigt, wenn alle Phoneme im gleichen Ausmass verkürzt werden.

Zum Erreichen einer guten Wiedergabequalität ist es deshalb unerlässlich, die Periode des Sprachsignals in stimmhaften Intervallen zu ermitteln und die Segmentlänge adaptiv darauf anzupassen. In einem weiteren Schritt kann die Überlappung schliesslich nur in den längeren, stimmhaften Signalabschnitten angewendet werden, um damit die Natürlichkeit des Signals zu bewahren.

Aufgabe

Die Studienarbeit lässt sich in folgende Teilaufgaben gliedern:

- Kennenlernen des Verfahrens aus [1].
- Entwickeln des Verfahrens mit festen Segmentlängen; Reduktion der Wiedergabezeit auf $3/4$, $2/3$ und $1/2$
- Kennenlernen der Methoden zur Schätzung der Pitchperiode aus [1].
- Entwickeln des Verfahrens mit adaptiver Segmentlänge; Reduktion der Wiedergabezeit auf $3/4$, $2/3$ und $1/2$
- Entwickeln des Verfahrens mit ausschliesslicher Anwendung auf die längeren, stimmhaften Signalabschnitte.
- Entwickeln der Algorithmen zunächst in Matlab, anschliessend in einer höheren Programmiersprache (C/C++ oder Java) mit Festkomma-Arithmetik.
- Erproben der verschiedenen Verfahren mit umfangreichen Signalbeispielen
- Ausführliche Dokumentation mit Schwergewicht auf den selbst geleisteten Arbeiten

Literatur

[1]Panos E. Papamichalis: Practical Approaches to Speech Coding, Prentice Hall, Englewood Cliffs, New Jersey, 1987.

Bericht

Über die Arbeit ist ein Bericht zu verfassen. Dabei sind die Richtlinien der Abteilung für Elektrotechnik für das Erstellen von Berichten einzuhalten. Alle verwendeten Quellen sind im Literaturverzeichnis des Berichts anzugeben (Hinweis im Text). Der Bericht ist in doppelter Ausführung abzugeben; ein Exemplar verbleibt am Labor für digitale Signalverarbeitung der HSR, das Doppel erhalten die Verfasser nach der Korrektur und der Bewertung zurück. Die erstellten Programme und der Text des Berichtes sind auf Diskette(n) (mit Nummer der Studienarbeit) beizulegen.

Termine, Bedingungen

Gemäss Vorgaben und Terminplan des Vorstandes der Abteilung für Elektrotechnik

Ausgabe der Aufgabenstellungen:

Präsentation der Studienarbeiten:

Abgabe des Berichts, Ende der Arbeit:

Assistent:

Kontaktadresse

Bernafon AG

A. Schaub

Eichtalstrasse 55

8634 Hombrechtikon

Tel. 055-264 13 51

E-Mail: as@bernafon.ch

1 Einleitung

In dieser Semesterarbeit werden zwei Varianten der schnelleren Wiedergabe von Sprachsignalen behandelt. Um dieses Ziel zu erreichen, ohne dass die Stimmlage und die Klangqualität ändert, muss die Grundfrequenz der sprechenden Person erkannt werden. Verschiedene Verfahren sind in der vorgegebenen Literatur *Pitch Detection And Its Importance In Speech Coding* erklärt. Nach dem Studium der Unterlagen wurde die Autokorrelationsfunktion ausgewählt.

Um die Ergebnisse der Autokorrelation zu verbessern, kann das Signal gefiltert oder mit Center Clipping vorverarbeitet werden.

Bei der Variante mit fester Grundperiode ist diese Autokorrelationsfunktion nicht notwendig, doch bei der Version mit adaptiver Grundperiode wird dieses Verfahren benötigt. Zur Verschnellerung der Sprache werden nun zwei nacheinander folgende Grundfrequenzen überlagert. Das Resultat ist, dass eine Periode im Sprachsignal verschwindet. Dadurch wird das Sprachsignal schneller abgespielt. Diese Semesterarbeit gibt Aufschluss über die Vorteile oder Nachteile der beiden Varianten und zeigt das Vorgehen zur Realisierung der Vorgabe.

2 Eingangssignal

Das verwendete Sprachsignal ist in Abbildung 2-1 dargestellt. Es wurde mit 16kHz abgetastet und weist eine Länge von 2.4 Sekunden auf. Es handelt sich um die Aufzeichnung einer Männerstimme, welche sagt: „*Ich bin Rudolf Ranick hier vom FCZ*“.

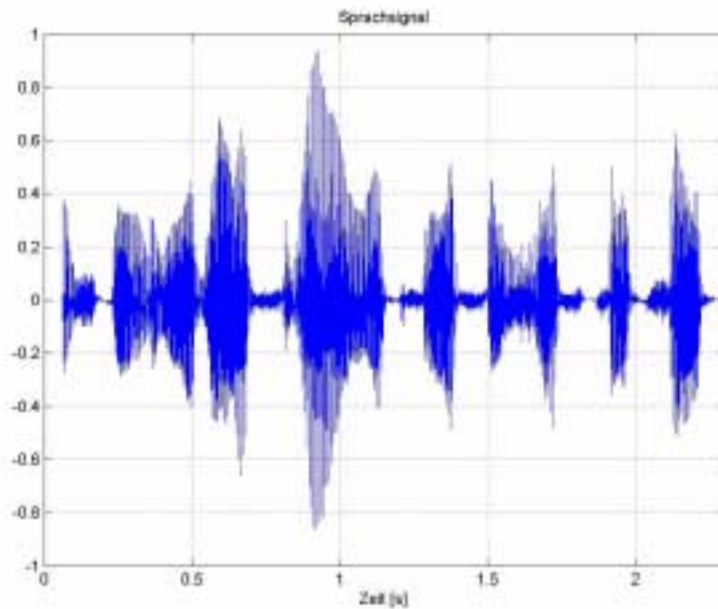


Abbildung 2-1: Das Sprachsignal

Die Abbildung 2-2 zeigt das verwendete Musiksignal aus dem Film „Titanic“. Die Abtastrate ist wiederum 16kHz, und die Länge beträgt 1.6 Sekunden.

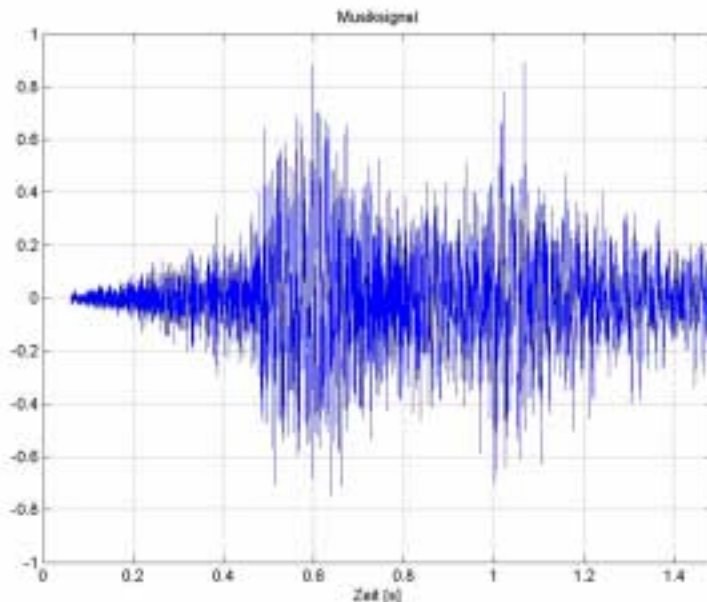


Abbildung 2-2: Das Musiksignal

3 Autokorrelation

Die Idee der Autokorrelation gemäss Abbildung 3-1 ist, einen Signalausschnitt mit einem dazu leicht verschobenen zu vergleichen. Mit diesem Verfahren kann die Grundperiode des Sprachsignals erkannt werden.

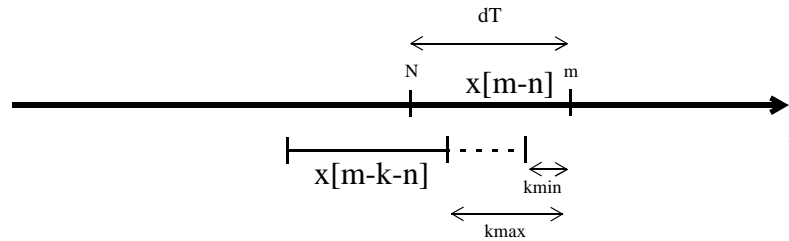


Abbildung 3-1: Autokorrelation

3.1 Vorgehen

Gemäss Gleichung 3-1 wird die Autokorrelation $\rho(m, k)$ des Eingangssignals $x[m]$ berechnet.

$$\rho(m, k) = \frac{\sum_{n=0}^{N-1} x[m-n]x[m-k-n]}{\sqrt{\sum_{n=0}^{N-1} x[m-n]^2 \sum_{n=0}^{N-1} x[m-k-n]^2}}$$

Gleichung 3-1: Berechnung der Autokorrelation

Zur Bestimmung der Grundperiode ist der Maximalwert $\varphi(m)$ der Autokorrelation $\rho(m, k)$ gemäss Gleichung 3-2 entscheidend, wobei das Intervall $[kmin, kmax]$ den Bereich der Grundfrequenz eines Sprachsignals abdeckt.

$$\varphi(m) = \max_{k \in [kmin, kmax]} (\rho(m, k))$$

Gleichung 3-2: Maximum des Autokorrelations Frames

Die Grundperiode entspricht dem Wert von k , welcher $\rho(m, k)$ maximiert.

3.2 Resultate der Autokorrelation

Die folgenden Abbildungen zeigen die Resultate, welche die Autokorrelation für die gegebenen Eingangssignale liefert.

Sprachsignal

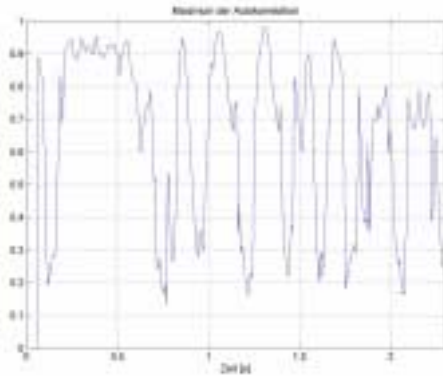


Abbildung 3-1: Das Maximum der Autokorrelationen

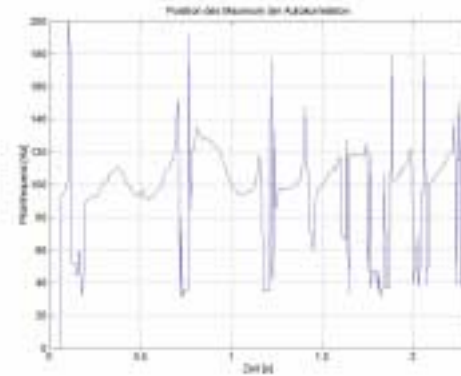


Abbildung 3-2: Position des Maximums

Musiksignal

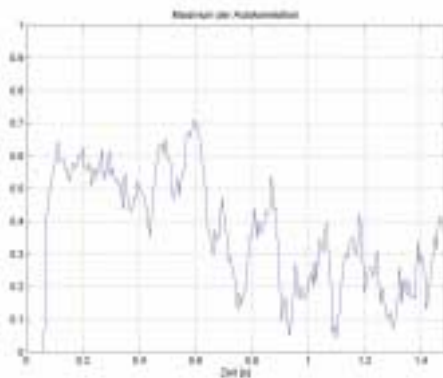


Abbildung 3-3: Das Maximum der Autokorrelationen

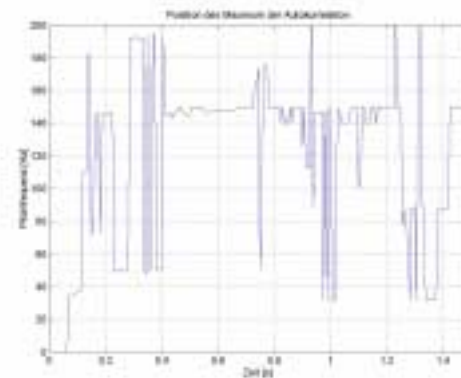


Abbildung 3-4: Position des Maximums

Es ist gut ersichtlich, dass bei einem Sprachsignal mit der Autokorrelation die besseren Werte erzielt werden als bei einem Musiksignal.

4 Autokorrelation mit Filterung

Die maximale Grundfrequenz eines Sprachsignals ist ungefähr 800Hz. Daher werden die höheren Frequenzen mit Hilfe eines Butterworth Filters unterdrückt. Dazu wurde ein Filter 4. Ordnung eingesetzt. Die folgenden Abbildungen zeigen die gefilterten Eingangssignale.

Sprachsignal

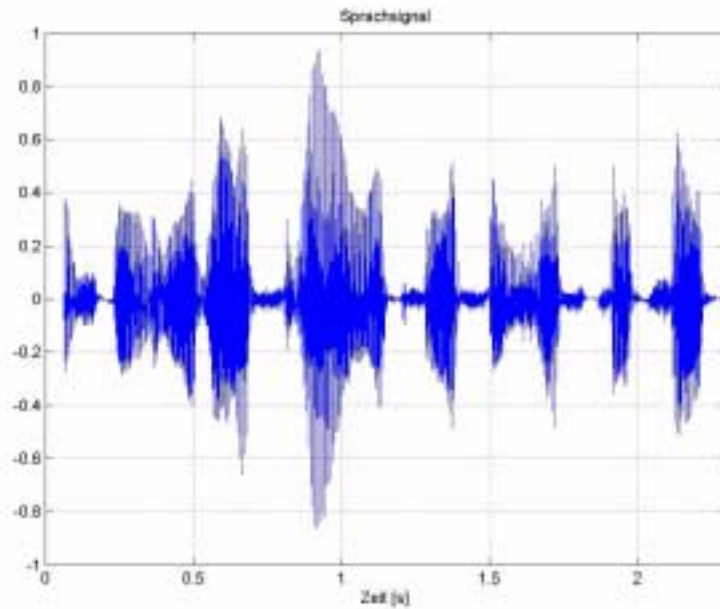


Abbildung 4-1: Gefiltertes Sprachsignal

Musiksignal

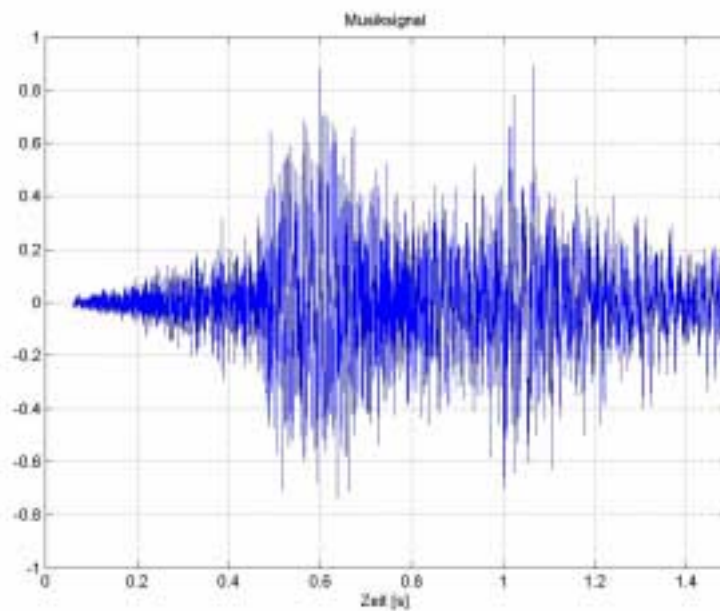


Abbildung 4-2: Gefiltertes Musiksignal

4.1 Resultate der Autokorrelation mit Filterung

Die nachfolgenden Abbildungen zeigen die maximale Autokorrelation und die Positionen, an denen das Maximum gefunden wurde.

Sprachsignal

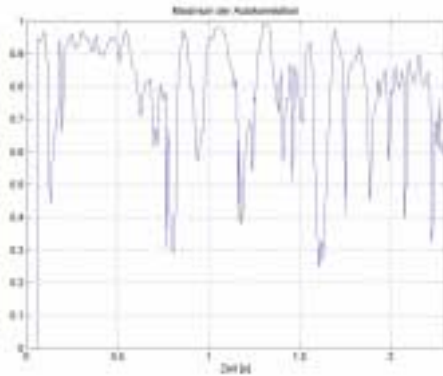


Abbildung 4-3: Das Maximum der Autokorrelationen

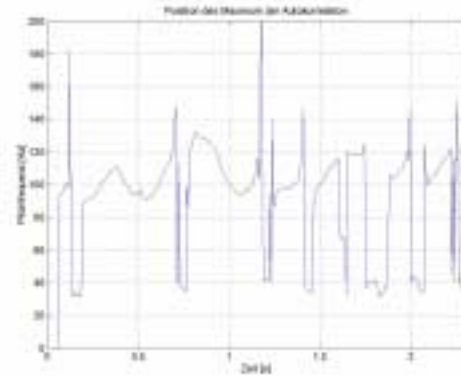


Abbildung 4-4: Position des Maximums

Musiksignal

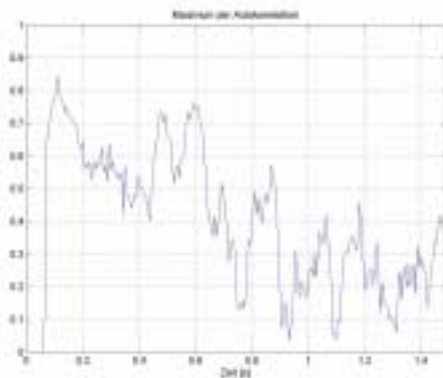


Abbildung 4-5: Das Maximum der Autokorrelationen

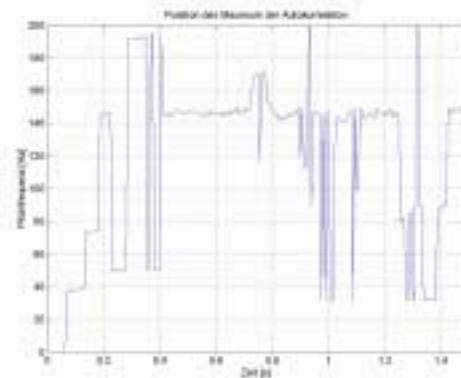


Abbildung 4-6: Position des Maximums

5 Autokorrelation mit Center Clipping

Das Center Clipping setzt alle Amplitudenwerte, welche dem Betrag nach kleiner als ein bestimmter Schwellwert sind, auf Null.

5.1 Die Berechnung der Envelope

Als erster Schritt wird die Umhüllende $u[n]$ des Signals $x[n]$ gemäss Gleichung 5-1 berechnet. Dies wird mit einer Art exponentiell abfallender Funktion gelöst.

$$u[n] = \max(|x[n]|, (1 - \varepsilon)u[n - 1]) \\ \text{mit } (0 < \varepsilon \ll 1)$$

Gleichung 5-1: Absolutwert des Samples

Bei der Festlegung des Schwellwertes $s[n]$ in der Gleichung 5-2 wurden mit $C=0.3$ die besten Resultate erreicht.

$$s[n] = C \cdot u[n]$$

Gleichung 5-2: Schwellwert

5.2 Die Durchführung des Center Clipping

Mit dem Center Clipping wird aus dem Eingangssignal $x[n]$ das Signal $y[n]$ gemäss Gleichung 5-3 berechnet.

$$y[n] = \begin{cases} x[n] & \text{fuer } (|x[n]| > s[n]) \\ 0 & \text{sonst} \end{cases}$$

Gleichung 5-3: Center Clipping

5.3 Resultate

Die Abbildungen 5-1 und 5-3 zeigen das Eingangssignal $x[n]$, die Umhüllende $u[n]$ und den Schwellwert $s[n]$. Weiter zeigen die Abbildungen 5-2 und 5-4 die Signale nach dem Center Clipping.

Sprachsignal

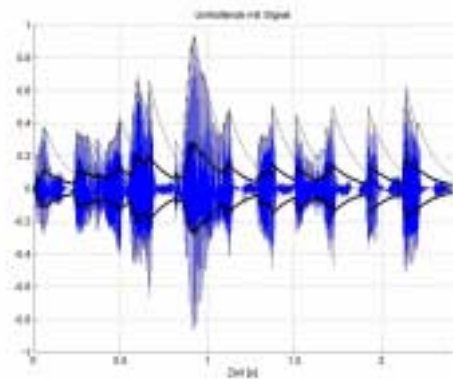


Abbildung 5-1: Envelope des Sprachsignals

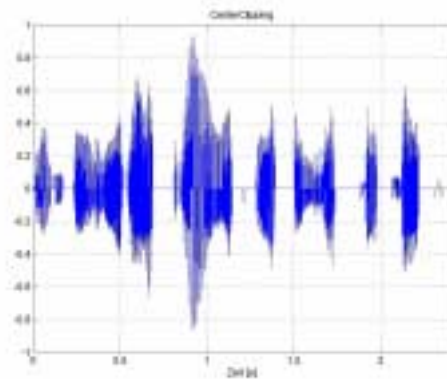


Abbildung 5-2: Das Sprachsignal mit Center Clipping

Musiksignal

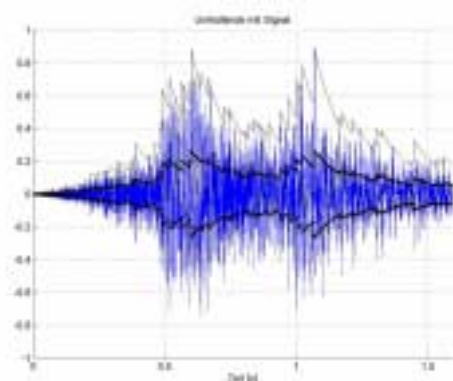


Abbildung 5-3: Envelope des Musiksignals

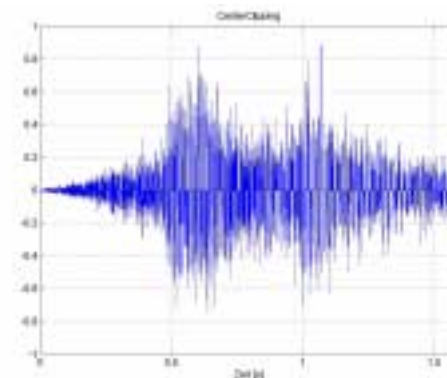


Abbildung 5-4: Das Musiksignal mit Center Clipping

Bei diesen Abbildungen wird sehr gut ersichtlich, dass die Amplituden, welche kleiner als der Schwellwert waren, auf Null gesetzt wurden.

5.4 Resultate der Autokorrelation mit Center Clipping

Die folgenden Abbildungen zeigen die Resultate, welche die Autokorrelation mit den beschnittenen Signalen liefert.

Sprachsignal

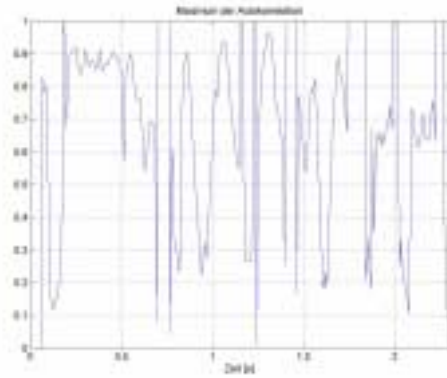


Abbildung 5-5: Das Maximum der Autokorrelationen

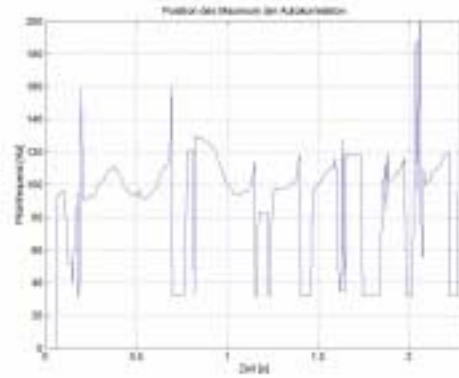


Abbildung 5-6: Position des Maximums

Musiksignal

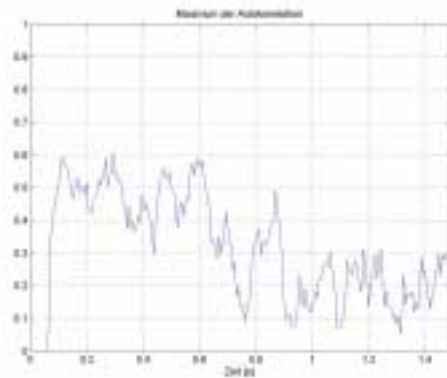


Abbildung 5-7: Das Maximum der Autokorrelationen

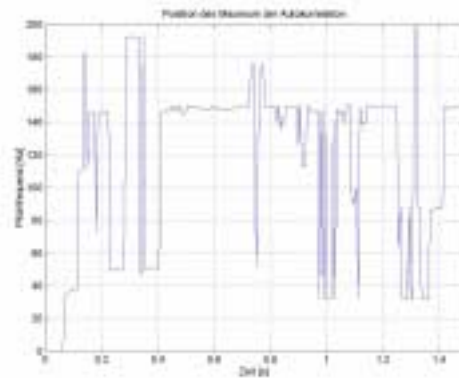


Abbildung 5-8: Position des Maximums

6 Die schnellere Wiedergabe

Für die schnellere Wiedergabe der Sprachsignale wurden in dieser Arbeit, zwei verschiedene Verfahren ausgewertet. Auf den folgenden Seiten werden die Vor- und Nachteile der beiden Varianten erläutert.

6.1 Vorgehen

Das Prinzip zur schnelleren Wiedergabe ist bei beiden Varianten gleich. Der Unterschied liegt bei der Bestimmung der Grundperiode. Im nächsten Kapitel wird die unterschiedliche Berechnung der Grundperiode gezeigt.

Filterung:

Es wurden keine Filter eingesetzt. Dieser Entscheidung wurde gefällt, nachdem verschiedene getestete Signale keine Unterschiede ergaben. Die Ausführung der Filter bräuchte nur Rechenzeit, welche man hier sehr gut sparen kann.

6.1.1 Grundperiodenbestimmung

TDHS mit fester Segmentlänge

Dieses Verfahren benutzt eine konstante Grundperiode. Dies bedeutet, dass sich die Grundperiode der Sprachperiode nicht anpasst. Folglich wird zur Verkürzung der Sprache über das ganze Signal die gleiche Periodenlänge verwendet. Diese Verkürzung erfolgt in festen Abständen auf das ganze Signal verteilt. Der Abstand zwischen den Segmenten, welche überlagert werden, definiert den Faktor für die schnellere Wiedergabe. Je mehr Grundperioden überlagert werden, desto schneller wird das Signal abgespielt. Die Verkürzung des Signals wird auf der nächsten Seite erläutert.

TDHS mit adaptiver Segmentlänge

Die Grundperiode wird mittels der Autokorrelationsfunktion bestimmt. Die Autokorrelation wurde, bis auf einen Punkt, von Kapitel 3 übernommen. Als kleine Änderung wird dT , in der Abbildung 3-1 ersichtlich, nicht konstant gehalten wird, sondern immer um die berechnete Grundperiode weiter geschoben wird. Dies hat zur Folge, dass sich die Grundperiode besser die Sprache anpassen kann. Zur Verkürzung wird somit immer die Grundperiode verwendet, welche mit dem Sprachsignal übereinstimmt. Wenn zwei nachfolgende Perioden gleich sind, oder in einem vordefinierten Bereich liegen, werden diese, wie in der Abbildung 6.1 gezeigt, überlagert.

6.2 Verkürzung des Signals

Die Verkürzung des Signals wurde bei beiden Verfahren gleich durchgeführt. Bei der Version mit fester Segmentlänge ist die Grundperiode P konstant. Bei der zweiten Variante wird diese adaptiv geändert.

Die Überlagerung von zwei aufeinanderfolgenden Signalen ergibt die gewünschte Verkürzung des Signals. Dies ist in der nachfolgenden Abbildung 6-1 sehr gut ersichtlich.

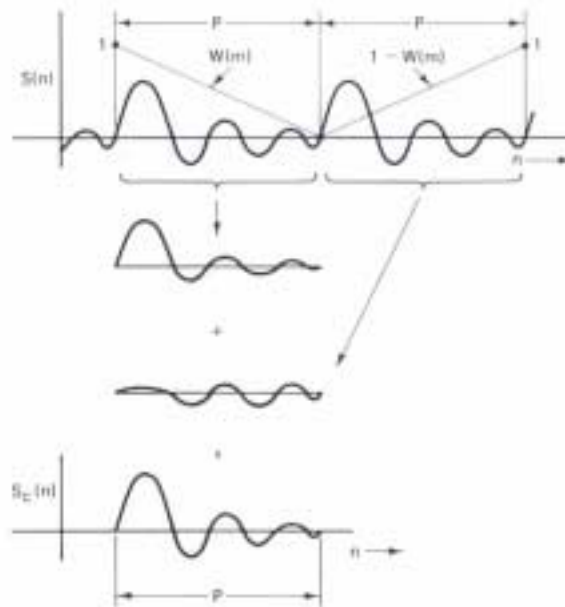


Abbildung 6-1: Überlagerung der Perioden

Die Funktion $W(m)$ in der Gleichung 6.1 ist der Faktor, mit der die erste Periode multipliziert wird. Die zweite Periode wird mit der Funktion $1 - W(m)$ multipliziert. Die Addition der beiden Grundperioden ergibt eine neue Periode.

$$W(m) = 1 - \frac{n}{P}$$

Gleichung 6-1: Funktion $W(m)$

6.3 Resultate der Wiedergabe

Für die nachfolgenden Auswertungen wurden die gleichen Sprachsignale wie bei der Autokorrelation verwendet. Die Abbildungen sind alle mit den Matlabfunktion erstellt worden.

TDHS mit festen Segmentlänge

Das Resultat dieses Verfahrens ist in den Abbildung 6-2 dargestellt. Das Ausgangssignal wird schneller abgespielt, ohne dass sich die Höhe der Sprache ändert. Doch die Hörprobe ergibt, dass das Ausgangssignal nicht mehr so rein klingt wie das Original, denn durch das willkürliche Überlagern der Perioden, geht der harmonische Klang verloren.

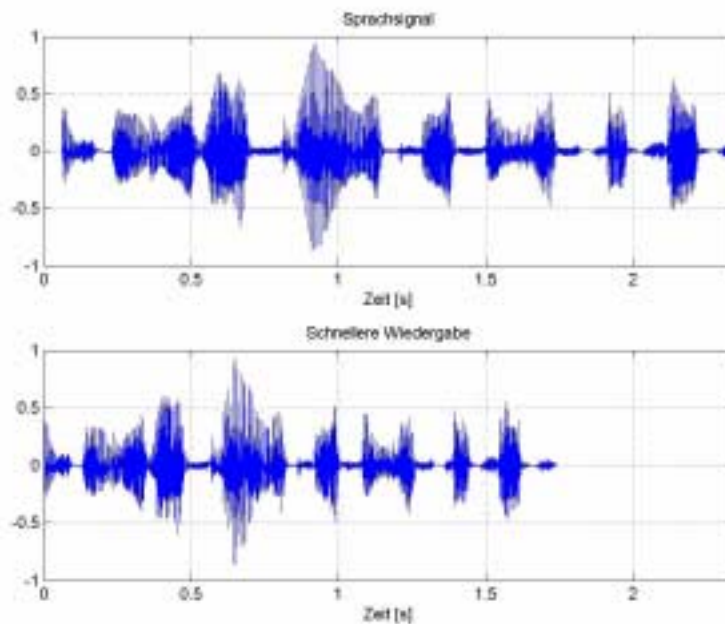


Abbildung 6-2: Sprachsignal um den Faktor 0.7 schneller (mit festen Segmentlänge)

Die weiblichen Stimmen werden erstaunlicherweise besser verarbeitet als die männlichen. Dies liegt an der höheren Grundperiode der Frauenstimmen.

Der Vorteil dieses Verfahrens ist, dass die Implementierung sehr schnell vor sich geht. Im Vergleich zur zweiten Variante, ist der Programm Code um einen beträchtlichen Teil kürzer. Die Rechenzeit für die Berechnung des Ausgangssignals wird auch um einen Faktor gekürzt. Sonst ist dieses Verfahren nicht einsatzfähig.

TDHS mit adaptiver Segmentlänge

Das Ergebnis dieses Verfahrens ist, dass bei Lauten, welche mehrmals die gleiche Grundperiode beinhalten, die Wiederholungen entfernt werden, das heisst der Laut wird bei gleichbleibender Frequenz schneller wiedergegeben.

Dies wird aus Abbildung 6-2 ersichtlich. Das Ausgangssignal hat keine der wichtigen Informationen des Originals verloren.

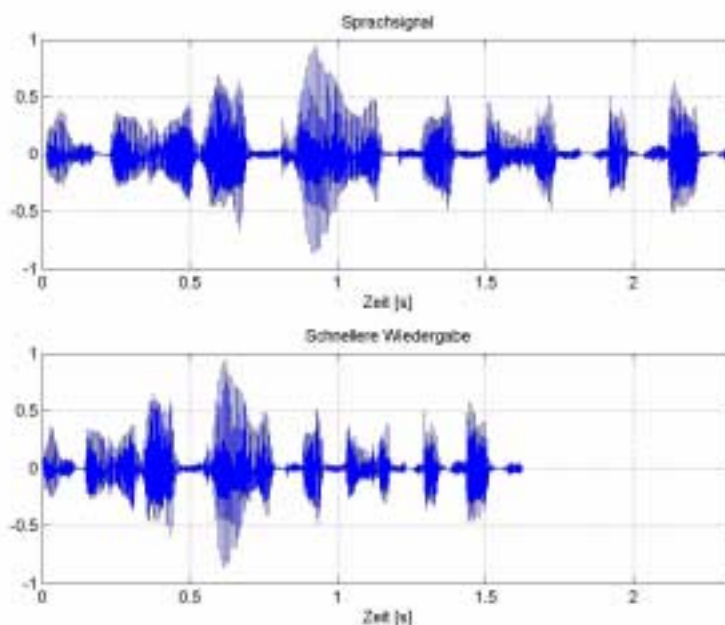


Abbildung 6-3: Sprachsignal um den Faktor 0.7 schneller (mit adaptiver Segmentlänge)

Die Hörproben von verschiedenen Sprachsignalen sind alle sehr gut ausgefallen. Alle weiblichen und männlichen Stimmen wurden ohne Verluste an Klangqualität schneller abgespielt. Das Problem mit der Reinheit des Signals wie bei der Variante mit fester Segmentlänge, wurde hier bei keinem der getesteten Signalen festgestellt.

Geschwindigkeitsanpassung:

Um die Geschwindigkeit mehr zu variieren, kann ein Frequenzbereich angegeben werden, in dem sich die beiden Perioden befinden müssen. Dadurch werden mehr (oder weniger) Perioden überlagert. Die Qualität des Ausgangssignals wird dadurch nicht beeinträchtigt.

Um das Verfahren noch zu verfeinern, wird die Anzahl Durchläufe gezählt. Wenn ein angegebener Wert erreicht wird, werden die Perioden nicht mehr überlagert. Bei verschiedenen getesteten Sprachsignalen wurde dadurch aber der Rhythmus der Wiedergabe etwas gestört. Das will heissen, dass am Beginn die Wiedergabe schnell von statten ging, und als der Faktor erreicht wurde, war die originale Wiedergabegeschwindigkeit wieder eingestellt. Doch dieses Problem könnte damit gelöst werden, dass nur jede zweite Übereinstimmung überlagert wird. Dies wurde aber bei dieser Arbeit nicht implementiert.

6.4 Resultate der Autokorrelation

Die Resultate der Autokorrelation des neuen Sprachsignals sind in den folgenden Abbildungen dargestellt.

TDHS mit festen Segmentlänge

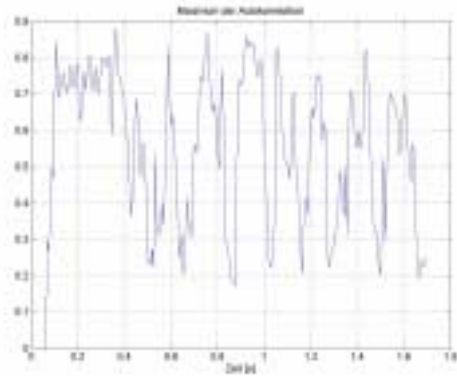


Abbildung 6-4: Das Maximum der Autokorrelationen

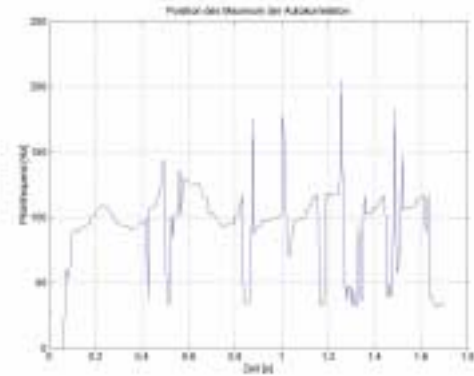


Abbildung 6-5: Position des Maximums

TDHS mit adaptiver Segmentlänge

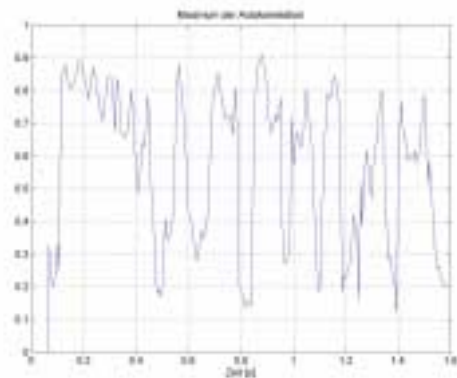


Abbildung 6-6: Das Maximum der Autokorrelationen

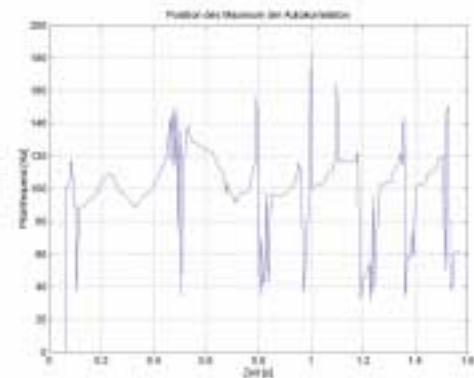


Abbildung 6-7: Position des Maximums

7 Auswertung

Dieser Bericht zeigt, dass die Variante mit adaptiver Segmentlänge das gestellte Problem sehr gut löst. Das grösste Problem bei beiden Varianten ist die beanspruchte Rechenzeit, welche bei der adaptiven Variante sogar noch etwas länger ist. Mit einer ressourcensparender Programmierung könnte vielleicht noch etwas Zeit gespart werden. Es ist sicher zu lange, wenn für ein 30 Sekunden Sprache eine ebenso lange Rechenzeit benötigt wird. Wenn das ganze Problem in einem DSP implementiert würde, dann könnte sicher noch Zeit eingespart werden, doch auf einem normalen Prozessor ist die benötigte Zeit klar zu lange.

Noch ein Wort zur Filterung des Eingangssignals. Dies wurde nicht angewendet, weil die Rechenzeit für eine gute Filterung zu lange wäre und das Resultat keine Verbesserung zeigt würde. Deshalb wurde dies nicht in C/C++ implementiert.

Alle getesteten Sprachsignale lieferten sehr gute Resultate. Die Wiedergabe eines Signals ist möglich bis zu einem Faktor von 0.5. Eine schnellere Wiedergabe ist mit diesen beiden Varianten nicht möglich. Dies ist aber auch nicht sinnvoll, denn eine Reduktion um 70% liefert schon Resultate, bei denen man Mühe hat, noch alle Information des Ausgangssignals richtig aufzunehmen.

7.1 Verwendete Signale

Alle verwendeten Signale sind im Verzeichnis ../Signale abgelegt. Durch das öffnen der Datei startme.html (oder startme.txt) wird die File-Struktur, welche im Anhang C beigelegt wurde, ersichtlich.

Musiksignal

Das verwendete Musiksignal für die Autokorrelation ist die musik.wav Datei.

Sprachsignal

Die folgenden Signalen wurden als Testsignale verwendet. Die längeren Signalen wurden nur mit der C++ Version getestet, dies weil die Berechnung im Matlab viel zu lange dauern würde.

Signal	Signaleigenschaft	Länge [min]
• moephone.wav	Telefon gespräch (Kind/Mann)	0:16
• katzen.wav	Telefon gespräch (Mann/Frau)	3:13
• h_bier.wav	hohe Männerstimme	0:04
• man.wav	Männerstimme	0:04
• aufforderung.wav	Männerstimme	0:12
• ichbin.wav	Männerstimme	0:02
• totemuegeli.wav	Toten Mürgerli (Franz Hohler)	5:56
• totemuegeli1.wav	Ausschnitt des Toten Mürgerli	0:20
• 2_frauen.wav	zwei Frauenstimmen	0:28
• woman.wav	verschiedene Frauenstimmen	0:31
• message.wav	Frauenstimmen	0:02
• zerror.wav	Errormeldung	0:01

8 Schlusswort

Ziel erreicht?

Die Aufgabe lautete, dass ein Verfahren mit fester und adaptiver Segmentlänge zur Reduktion der Wiedergabezeit zu entwickeln sei. Dieses Verfahren sollte in Matlab und in C/C++ implementiert werden.

Dieses Ziel wurde in der vorgegebenen Zeit erreicht. Das Verfahren, welches sich ausschliesslich auf die längeren, stimmhaften Signalabschnitte befassen soll, wurde in dieser Arbeit nicht behandelt. Dies wurde aber nach Absprache mit dem Betreuer so entschieden, weil die Implementierung der beiden ersten Varianten in C/C++ spannender und lehrreicher eingestuft wurden. Somit wurde das vorgegebene Ziel dieser Semesterarbeit erreicht.

Persönlicher Eindruck

Ich habe diese Semesterarbeit als interessant und abwechslungsreich empfunden. Ich konnte mein Wissen über die Entwicklungsumgebungen auffrischen und erweitern. Auch die ganze Prozedur, welche so ein Projekt mit sich bringt, wie Projektplanung, Bericht schreiben und das ganze Vorgehen bei der Programmierung der Algorithmen, brachten mir neue Impulse und neues Wissen.

Dies will heissen, dass ich an dieser Arbeit meistens Spass hatte zu arbeiten. Die Tatsache, dass ich die Arbeit alleine durchführte, war aus meiner Sicht kein Nachteil. Denn die gesetzten Ziele waren sehr gut auf eine Person abgestimmt.

An dieser Stelle möchte ich noch Herrn Schaub, meinem Betreuer, danken.

Rapperswil den 1. März 2002

Marti Adrian

9 Literaturverzeichnis

- 1 Panos E. Papamichalis: Practical Approaches to Speech Coding, Prentice Hall, Englewood Cliffs, New Jersey, 1987

A Listings Matlab

Zur Programmierung wurde Matlab R.12 Ver.6.0 von Mathworks angewendet. Die nachfolgenden Funktionen berechnen die Autokorrelation des Eingangssignals und geben die Abbildungen aus, welche in dieser Arbeit benötigt wurden.

A.1 Funktion für die Autokorrelation

File ../Matlab/Auto_Kor.m

```
%Rechnet die Autokorrelation von x (einer wav Datei)
function [kor_out,pos_out,startSample,stopSample,signal,signalFil,auto_out,t,sampleFrq]=AutoKor(x)

%Angaben ueber die Wav-Datei
sampleFrq=16000; %ein Sample = 62.5e-6 sec
Fmin=74; %minimal Frequenz einer Sprache
Fmax=500; %maximal Frequenz einer Sprache
Frame=0.05; %Fensterlaenge 100ms

%Grundeinstellung
dSample=100; %Schritte zwischen den Autokorrelations Samples
startSample=1000; %Startposition der Autokorrelation
startFrame=startSample;

%Einlesen der Datei
signal=wavread(x); %einlesen der wavdatei in eine Matrix 23000x1
signal=signal'; %Transponiert der Matrix 1x23000

%Butterworth-Filter mit der Grenzfrequenz 800Hz
[B,A]=butter(4,0.1); %Butterworth 4-Ordnung Grenzfreq: 800Hz/0.5*sampleFrq
signalFil=filter(B,A,signal);
signalFil=signal;
%Ruft das CenterClipping auf
signalFil=CenterClip(signal,sampleFrq);

%Initialisierung
pos_out=0;
kor_out=0;
auto_out=0;

%Berechnungen
Tmax=(1/Fmin)*sampleFrq; %maximale Samples fuer 74Hz
Tmax=10*fix(Tmax/10);
Tmin=(1/Fmax)*sampleFrq; %minimale Samples fuer 500Hz
Tmin=10*fix(Tmin/10);
N=dSample*fix(length(signalFil)/dSample); %laenge des Signals gerundet
stopSample=N-700; %Stoppoosition der Autokorrelation
dT=sampleFrq*Frame; %FrameLaenge fuer den Ausschnitt (100ms)

%schiebt den Sampelausschnitt um dSample nach rechts
for startFrame=startSample+1:dSample:stopSample
    endFrame=startFrame+dT-1; %setzt das Framefenster x
    x=signalFil(startFrame:endFrame); %schneidet das Frame aus dem Signal
    xx=sum(x.^2); %rechnet die Summe der Quadrate vom Frame
    Rx=0;
%schiebt das kodierte Signal um dTmin nach link und rechnet von jedem die Autokorrelation aus
    for dTmin=Tmin:Tmax
        y=signalFil(startFrame-dTmin:endFrame-dTmin); %verschiebt das Signal um dTmin nach links
        R=sum(x*y'); %rechnet die Summe von x*y' (y'=Transponiert)
        yy=sum(y.^2); %rechnet die Summe der Quadrate von y
        n=sqrt(xx*yy); %nimmt die Wurzel von den beiden Quadraten
        Auto_kor=R/n; %rechnet die Autokorrelation aus
        Rx=[Rx,Auto_kor]; %schreibt die Resultate in einen Array
    end
end
%
auto_out=[auto_out,Rx]; %schreibt die Autokorrelation in einen Array
[max_out,pos]=max(Rx); %max_out= maximum des Array, pos=position des Maximum
pos_out=[pos_out,pos+Tmin]; %schreibt die position in einen Array + Tmin
kor_out=[kor_out,max_out]; %schreibt das maximum in einen Array
end
%
t=(startSample:dSample:stopSample)/sampleFrq; %Zeitvektor fuer die Ausgabe der Signale
```

File ../Matlab/CenterClip.m

```
function CClipRes=CenterClip(signal,sampleFrq)
%Diese Fkt setzt die Amplituden welche kleiner als eine Schwelle (envelopeRes) auf Null

%Initialiesierung
CClip=0;
envelopeRes=0;
CClipRes=0;

%Variablen
envelopeFakt=0.9995; %Envelope Faktor, gibt an wieviel die Umhuellende abfllt
CClipFakt=0.3; %CenterClipping Faktor
dSample=100; %Schritte zwischen den Autokorrelations Samples

N=dSample*fix(length(signal)/dSample); %laenge des Signals gerundet
x=envelopeFakt*abs(signal(1)); %Erster Wert der Umhuellende

%Berechnung der Umhuellende und loescht die Werte welcher unter der Schwelle liegen
for i=1:N
    y=abs(signal(i+1)); %Zweiter Wert der Umhuellende

%Ist der erste Wert groesser oder gleich dem Zweiten Wert
    if x >= y
        x=x*envelopeFakt; %Wenn Ja, dann wird die Umhuellende um einen Faktor reduziert
    else
        x=envelopeFakt*abs(signal(i+1)); %Wenn Nein, wird die Umhuellende auf diesen Wert gelegt
    end

    envelopeRes=[envelopeRes,x]; %Macht aus den Resultaten einen Array

%Berechnet das CenterClipping
    z=CClipFakt*x; %Setzt die Schwelle mit dem Faktor CClipFakt
    v=abs(signal(i)); %Nimmt den Absolutwert des Signals

%Ist die Schwelle groesser oder gleich dem Signal-Wert
    if z >= v
        CClip=0.001; %Wenn Ja, dann wird das Signal auf Null gesetzt
    else
        CClip=signal(i); %Wenn Nein, wird der Signal Wert ,bernommen
    end
    CClipRes=[CClipRes,CClip]; %Macht aus den Resultaten einen Array
end

%Angaben fuer den Plot
t=0:N; %Zeitvariable fuer die Ausgabe
t=t/sampleFrq; %Zeitvaribale in sec
NLength=length(t)/sampleFrq; %Laenge des Array
envelopeResPos=CClipFakt*envelopeRes; %Positive Schwelle
envelopeResNeg=-envelopeResPos; %Positive Schwelle

%Gibt die Signale aus
hold on
plot(t,signal(1:N+1));
plot(t,envelopeRes,'k');
plot(t,envelopeResPos,'k. ');
plot(t,envelopeResNeg,'k. ');
hold off
axis([0 NLength -1 1]);
title ('Umh,llende mit Signal');
xlabel ('Zeit [s]');
grid on;

figure;
plot(t,CClipRes)
axis([0 NLength -1 1]);
title ('CenterClipping');
xlabel ('Zeit [s]');
grid on;

figure;
hold on;
plot(t,envelopeRes,'k');
plot(t,envelopeResPos,'k. ');
plot(t,envelopeResNeg,'k. ');
hold off
axis([0 NLength -1 1]);
title ('Umh,llende');
xlabel ('Zeit [s]');
grid on;
```

A.1.1 Aufruf und Ausgabe

Die Berechnungen wurden über die folgende Funktion Sprache.m, wenn es ein Sprachsignal ist, oder über die Funktion Musik.m gestartet. Diese Funktionen rufen die Programme zur Berechnung der Autokorrelation auf.

Die Funktion Musik unterscheidet sich nur die Beschriftung der Ausgabe von der Funktion Sprache.

File ../Matlab/Sprache.m und ../Matlab/Musik.m

```
function Sprache(x)

[kor_out, pos_out, startSample, stopSample, signal, signalFil, auto_out, t, sampleFrq]=AutoKor(x);

%Angaben fuer die Ausgabe der Signalen
TSignal=(startSample:stopSample)/sampleFrq;
lengthSignal=length(TSignal)/sampleFrq;
n=(stopSample-startSample)/sampleFrq;

figure;
%Eingangssignal
plot(TSignal, signal(startSample:stopSample));
axis([0 lengthSignal -1 1]);
title (,Sprachsignal');
xlabel (,Zeit [s]');
grid on;

figure;
%Sprachsignal
plot(TSignal, signal(startSample:stopSample));
axis([0 lengthSignal -1 1]);
title (,Sprachsignal');
xlabel (,Zeit [s]');
grid on;

figure;
%Sprachsignal gefiltert
plot(TSignal, signalFil(startSample:stopSample));
axis([0 lengthSignal -1 1]);
title (,Sprachsignal gefiltert');
xlabel (,Zeit [s]');
grid on;

figure;
%Maximum der Autokorrelation
plot(t, kor_out);
axis([0 n 0 1]);
title (,Maximum der Autokorrelation');
xlabel (,Zeit [s]');
grid on;

figure;
%Position des Maximum der Autokorrelation
plot(t, pos_out);
axis([0 n 0 200]);
title (,Position des Maximum der Autokorrelation');
xlabel (,Zeit [s]');
ylabel (,Pitchfrequenz [Hz]');
grid on;

figure
plot(TSignal, signalFil(startSample:stopSample));
hold on;
plot(t, kor_out);
axis([0 n -1 1]);
title (,Signal + Autokorrelation');
xlabel (,Zeit [s]');
grid on;
hold off;
```

A.2 Funktionen für die schnellere Wiedergabe

Für die schneller Wiedergabe wurden die nachfolgenden Matlab-Programme geschrieben. Der Source-Code der beiden Versionen und den benützten Funktionen ist auf den nachfolgenden Seiten dargestellt.

A.2.1 Funktion für die Version mit fester Segmentlänge

File ../Matlab/faster.m

```
function Sprache(x)
% Funktionsaufruf: faster('wav');
% schnelle Wiedergabe mit fester Segmentlaenge

%Einlesen der Datei
[signal,sampleFrq]=wavread(x);           %einlesen der wavdatei in eine Matrix + einlesen der sampleFrq
signal=signal';                          %Transponiert der Matrix

%Initialisierung
N=10*fix(length(signal)/10);             %laenge des Signals gerundet
frame=600;                               %der Abstand zwischen den Segmenten
s=1000;                                  %Setzt die Startposition
cut=150;                                  %definiert die Grundperiode welche ueberlagert wird
pos=s+frame;                             %Setzt die erste Segmentposition
OutOut=0;

%while schleife mit s++
while s<N-500
    %wenn die s gleich Segmentperiode
    if (s==pos)
        for J=1:cut
            Neg=signal(s+J)*(1-(J/cut));   %erste Periode welche mit 1-W(m) multipliziert wird
            Positive=signal(s+cut+J)*(J/cut); %zweite Periode welche mit W(m) multipliziert wird
            Out=Neg+Positive;              %ueberlagerung der beiden Perioden
            OutOut=[OutOut,Out];          %schreibt das Resultat in einen Array
        end
        s=pos+2*cut;                       %setzt die neue Startposition
        pos=frame+pos;                     %setzt die neue Position der naechsten Ueberlappung
    else
        Out=signal(s);                     %wenn nichts ausgeschnitten werden soll, wird die das Signal kopiert
        OutOut=[OutOut,Out];
    end
    s=s+1;
end

WAVWRITE(OutOut,sampleFrq,16,'test.wav') %das Out-Signal wird in ein test.wav file gschrieben

%Ausgabe der Resultate
t=0:(length(OutOut)-1);
t=t/sampleFrq;
TSignal=(1000:N-10)/sampleFrq;
lengthSignal=length(TSignal)/sampleFrq;

figure;
%Eingangssignal
subplot (2,1,1)
plot(TSignal,signal(1000:N-10));
axis ([0 lengthSignal -1 1]);
title (,'Sprachsignal');
xlabel (,'Zeit [s]');
grid on;

%Resultat
subplot (2,1,2)
plot(t,OutOut);
axis([0 lengthSignal -1 1]);
title (,'Schnellere Wiedergabe');
xlabel (,'Zeit [s]');
ylabel (,'Pitchfrequenz [Hz]');
grid on;
```

A.2.2 Funktion für die Version mit adaptiver Segmentlänge

File ../Matlab/FastPlay.m

```

function FastPlay(x,FakTDHS)
%Funktionsaufruf: FastPlay('wav',Faktor);
%Rechnet die Autokorrelation von x (einer wav Datei) und ruft FakBe.m und TDHS.m auf

%Angaben ueber Sprache
Fmin=74; %minimal Frequenz einer Sprache
Fmax=500; %maximal Frequenz einer Sprache
Frame=0.05; %Fensterlaenge 50ms

%Grundeinstellung
dSample=100; %Schritte zwischen den Autokorrelations Samples

%Einlesen der Datei
[signal,sampleFrq,NBITS]=wavread(x); %einlesen der wavdatei in eine Matrix + sample Frequenz + Bits
signalFil=signal'; %Transponiert der Matrix

%Berechnungen
Tmax=(1/Fmin)*sampleFrq; %maximale Samples fuer 74Hz
Tmax=10*fix(Tmax/10)
Tmin=(1/Fmax)*sampleFrq; %minimale Samples fuer 500Hz
Tmin=10*fix(Tmin/10)
N=dSample*fix(length(signalFil)/dSample) %laenge des Signals gerundet
startSample=Tmax; %Startposition der Autokorrelation
startFrame=startSample+1;
dT=sampleFrq*Frame; %Framelaenge fuer den Ausschnitt (50ms)
dT=10*fix(dT/10);
stopSample=N-dT-1; %Stopposition der Autokorrelation

%Ruft die Funktion zur Berechnung des Faktors auf
faktor=FakBe(FakTDHS)
laenge=length(signalFil)/sampleFrq %rechnet die laenge des Originalsignals
laengeFak=laenge*(1-FakTDHS) %rechnet die gewuenschte laenge des Ausgangsignals

%Initialisierung
pos0=100;
laengeRes=0;
Cut=0;
Resultat=0;
i=0;
%Schiebt den Sampelausschnitt um dSample nach rechts
while startFrame<stopSample
    laengeRes=laengeRes+(Cut/sampleFrq);
    endFrame=startFrame+dT-1; %setzt das Framefenster x
    x=signalFil(startFrame:endFrame); %schneidet das Frame aus dem Signal
    xx=sum(x.^2); %rechnet die Summe der Quadrate vom Frame
    Rx=0;
%Schiebt das kopierte Signal um dTmin nach link und rechnet von jedem die Autokorrelation aus
    for dTmin=Tmin:Tmax
        y=signalFil(startFrame-dTmin:endFrame-dTmin); %verschiebt das Signal um dTmin nach links
        R=sum(x*y'); %rechnet die Summe von x*y' (y'=Transponiert)
        yy=sum(y.^2); %rechnet die Summe der Quadrate von y
        n=sqrt(xx*yy); %nimmt die Wurzel von den beiden Quadraten
        Auto_kor=R/n; %rechnet die Autokorrelation aus
        Rx=[Rx,Auto_kor]; %schreibt die Resultate in einen Array
    end
    [max_out,pos]=max(Rx); %max_out= maximum des Array, pos=position des Maximum
    pos=pos+Tmin; %Grundperiode des Signals
%ruft die tdhs Funktion auf
    [posNew,Out,Cut]=tdhs(pos,pos0,startFrame,signalFil,laengeRes,laengeFak,faktor);
    Resultat=[Resultat,Out]; %schreibt das Ergebnis der tdhs in einen Array
    startFrame=posNew+pos; %setzt das startFrame auf die neue Position
    pos0=pos; %schreib die pitch wieder ins original pitch
end
laengeRes=length(Resultat)/sampleFrq %schreibt zur Kontrolle die laenge der Resultat aus
laengeRes/laenge %schreibt den Erzielten Faktor aus

WAVWRITE(Resultat,sampleFrq,NBITS,'test.wav') %schreibt das Resultat in eine Wavdatei

%Ausgabe der Resultate
t=0:(length(Resultat)-1);
t=t/sampleFrq;
TSignal=(startSample:stopSample)/sampleFrq;
lengthSignal=length(TSignal)/sampleFrq;

figure;
%Eingangsignal
subplot(2,1,1)
plot(TSignal,signal(startSample:stopSample));
axis([0 lengthSignal -1 1]);
title('Sprachsignal');
xlabel('Zeit [s]');

```

```
grid on;

%Resultat
subplot (2,1,2)
plot(t,Resultat);
axis([0 lengthSignal -1 1]);
title (,'Schnellere Wiedergabe');
xlabel (,'Zeit [s]');
ylabel (,'Pitchfrequenz [Hz]');
grid on;
```

File ../Matlab/TDHS.m

```
%Funktion TDHS
%Ueberlagert zwei Pitchperioden, welche die fast gleiche Grundperiode haben
%gibt die neue Grundperiode, das neue Ausgangssignal und den Teil welcher ausgeschnitten wird zurueck

function [posNew,OutOut,Cut]=thds(pos,pos0,startFrame,signal,laengeRes,laengeFak,faktor)

%Initialisierung
Neg=0;
Positive=0;
Out=0;
Out2=0;
OutOut=0;
res3=0;
Cut=0;
%Ist der Unterschied der beiden Perioden kleiner als einen Faktor und ist die laenge des Resultat kleiner
als die gewünschte Laenge
if ((pos-pos0)^2<=faktor)&(laengeRes<laengeFak))
%wenn ja
for J=1:pos0
Neg=signal(startFrame-pos0+J)*(1-(J/pos0)); %berechnet die erste Periode
Positive=signal(startFrame+J)*(J/pos0); %berechnet die zweite Periode
Out=Neg+Positive; %addiert die beiden Perioden
Out2=[Out2,Out]; %schreibt das Ergebnis in einen Array
end
Cut=pos0; %speichert die Grundperiode
OutOut=Out2(2:pos0+1); %kopiert das Frame in den OutOut Array
posNew=startFrame+pos0; %die neue Position fuer die naechste Berechnung
else
Out=signal(startFrame+1-pos0:startFrame); %schneidet die Grundperiode raus
Out2=[Out2,Out]; %speichert diese im Array
OutOut=Out2(2:pos0+1); %kopiert das Frame in den OutOut Array
posNew=startFrame; %die neue Position fuer die naechste Berechnung
end
end
```

File ../Matlab/FakBe.m

```
%Funktion fuer die Berechnung des Faktors, welcher den Grundperiodenbereich angiebt

function fak=FakBe(FakTDHS)

if ((FakTDHS>=0.8)&(FakTDHS<=1))
fak=0;
end
if ((FakTDHS>=0.68)&(FakTDHS<0.8))
fak=1;
end
if ((FakTDHS>=0.60)&(FakTDHS<0.68))
fak=4;
end
if ((FakTDHS>=0.57)&(FakTDHS<0.60))
fak=9;
end
if ((FakTDHS>=0.55)&(FakTDHS<0.57))
fak=16;
end
if ((FakTDHS>=0.3)&(FakTDHS<0.55))
fak=25;
end
if ((FakTDHS<0.3))
fak=100;
end
end
```

B Listings C++

B.1 Main Programm

Dieses Programm öffnet das gewünschte wav-File und speichert dieses um den angegebenen Faktor schneller, wieder in ein wav-File. Das Main-Programm heisst TDHS.exe.

File ../Visual/TDHS.cpp

```
#include "fastplay.h"
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <dsound.h>
#include <mmreg.h>
#include <string.h>
#include <math.h>

void main () {
//char * name= {"d:/TDHS/Visual/zerror.wav"};
//char * outname= {"d:/TDHS/Visual/test.wav"};

fastplay beisp;
cout << "Mit diesem Programm wird die Wiedergabe des Input-Sprachsignals schneller"<<endl;
cout << "*****"<<endl;
cout << endl;
cout << "Geben sie das Input-Sprachsignal ein (mit dem Pfad z.B: c:test.wav) :" <<endl;
char * name;
    name = new char[1000];
    cin.getline(name, 1000, '\n');
cout << endl;
cout << "Geben sie das Output-Sprachsignal ein (mit dem Pfad z.B: c:test.wav) :" <<endl;
char * outname;
    outname = new char[1000];
    cin.getline(outname, 1000, '\n');
cout << endl;
cout << "Geben sie den gewuenschten Faktor ein :"<<endl;;
cin >> beisp.fFakIn;
cout << endl;

beisp.OpenWaveFile (name);
beisp.CharToFloat();
beisp.PitchFaktor();
beisp.PitchDetection(beisp.fSignallinks);
beisp.FloatToChar();
beisp.WriteWaveFile (outname);
beisp.WriteTxtFile("d:/TDHS/Visual/text.txt");

cout << endl;
cout << "Das Output-Signal wurde erfolgreich erststellt"<<endl;
cout << "*****"<<endl;
cout <<endl;
cout << "Semesterarbeit der HSR von Marti Adrian"<<endl;
cout << "copyright by Marti Adrian"<<endl;
cout << "*****"<<endl;

}
}
```

B.2 Header Datei

File ../Visual/fastplay.h

```
#include <iostream.h>
#include <dsound.h>
#include <mmreg.h>
#include <string.h>

class fastplay {

public:
float fFakOut,
    fFakIn,
    fFak;
int  ibit,
    z;

char * rID,
    * wID,
    * fID,
    * dID,
    * cSignal,
    * cSignalOut;

long  lFileSize,
    lWaveFormatSize,
    lSignalSize,
    lSignal,
    lOutSize;

float * fSignal,
    * fSignalLinks,
    * fSignalOut,
    * fMaxArray;

int * iPosArray;

WAVEFORMATEX InfoWave;
fastplay();

void OpenWaveFile(char *);
void CharToFloat();
void PitchFaktor();
void FloatToChar();
void PitchDetection(float *);
void WriteWaveFile(char *);
void WriteTxtFile(char *);

};
```

B.3 Operation

File ../Visual/fastplay.cpp

```
#include "fastplay.h"
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <dsound.h>
#include <mmreg.h>
#include <string.h>
#include <math.h>
```

B.3.1 fastplay()

```
// Konstruktor
//*****
fastplay::fastplay(){

    lSignalSize = 0;

    InfoWave.wFormatTag = 0;
    InfoWave.nChannels = 0;
    InfoWave.nSamplesPerSec = 0;
    InfoWave.nAvgBytesPerSec = 0;
    InfoWave.nBlockAlign = 0;
    InfoWave.wBitsPerSample = 0;

}
}
```

B.3.2 OpenWaveFile

```
// OpenWaveFile
// öffnet das wav-File und speichert die Daten in einem
// char array. Die restlichen Angaben über das wav-File
// werden für die Ausgabe übernommen
// (Grundstruktur wurde von A.Schlöpfer und F.Meier
// übernommen)
//*****
void fastplay::OpenWaveFile(char * FileName){

    ifstream hFile;
    rID = new char [5];
    rID[4] = '\0';
    dID = new char[5];
    dID[4] = '\0';
    fID = new char[5];
    fID[4] = '\0';
    wID = new char[5];
    wID[4] = '\0';
    hFile.open(FileName, ios::binary);
    if (hFile==NULL){
        cout << "Kann File nicht öffnen";
    }
    hFile.read(rID, 4);
    if (strcmp(rID,"RIFF")){
        cout << "Hat keine RIFF ID";
    }
    //cout <<"rID " <<rID << endl;
    hFile.read((char*)&lFileSize, 4);
    //cout << "FileSize " << lFileSize << endl;
    hFile.read(wID, 4);
    if (strcmp(wID,"WAVE")){
        cout << "Ist nicht vom Typ WAVE";
    }
    //cout <<"wID " <<wID << endl;
    hFile.read(fID, 4);
    if (strcmp(fID,"fmt ")){
        cout << "Hat nicht die fmt chunk ID";
    }
    //cout <<"fID " << fID << endl;
    hFile.read((char*)&lWaveFormatSize, 4);
    //cout << "WaveFormatSize " <<lWaveFormatSize << endl;
    hFile.read((char*) &InfoWave, lWaveFormatSize);
    //cout << "wFormatTag " <<InfoWave.wFormatTag << endl;
    //cout << "nChannels " <<InfoWave.nChannels << endl;
    //cout << "nSamplesPerSec " <<InfoWave.nSamplesPerSec << endl;
    //cout << "nAvgBytesPerSec " <<InfoWave.nAvgBytesPerSec << endl;
    //cout << "nBlockAlign " <<InfoWave.nBlockAlign << endl;
    //cout << "wBitsPerSample " << InfoWave.wBitsPerSample << endl;
    if (InfoWave.wFormatTag != WAVE_FORMAT_PCM){
```

```
    cout << "Wave Format ist nicht vom Typ PCM";
}

hFile.read(dID, 4);
if (strcmp(dID,"data")){
    cout << "Die zweite chunk ID ist nicht data";
}
//cout <<"dID "<< dID << endl;
hFile.read((char*)&lSignalSize, 8);
cSignal = new char [lSignalSize+1];
cSignal[lSignalSize] = '\0';

hFile.read(cSignal, lSignalSize);

float * fDaten;
fDaten = new float[lSignalSize];
}
```

B.3.3 CharToFloat

```
//          CharToFloat
// wandelt den Input-char Array in einen float Array
// und Normiert diesen noch
//*****
void fastplay::CharToFloat(){

switch( InfoWave.wBitsPerSample )
{
    case 8:
        ibit=128;
        break;
    case 16:
        ibit=32768;
        break;
    default :
        break;
}

int k = 0;
fSignal = new float[lSignalSize];
while (k<lSignalSize) {

    fSignal[k] = cSignal[k];
    //Normierung
    fSignal[k]=fSignal[k]/ibit;

    if (fSignal[k]<0)
        fSignal[k]=1+fSignal[k];
    else
        fSignal[k]=-1+fSignal[k];

    k++;
}
int q=0,
    i=0;

if (InfoWave.nChannels==2) {
    lSignal=lSignalSize/2;
    fSignallinks = new float[lSignalSize/2];
    while (q<lSignalSize) {
        fSignallinks[i]=fSignal[q];
        q=q+2;
        i++;
    }
}
else {
    fSignallinks = new float[lSignalSize];
    lSignal=lSignalSize;
    while (i<lSignalSize) {
        fSignallinks[i]=fSignal[i];
        i++;
    }
}
fFak=(float)(1-fFakIn);
}
```

B.3.4 PitchFaktor

```
//          Faktor für die Pitchperiode löschen
// berechnet den Faktor, welcher angibt um wieviel
```

```
// das wav-File gelöscht werden soll
//*****

void fastplay::PitchFaktor(){

if (fFakIn>=0.69)
    fFakOut=0;
else
{
    if (fFakIn>=0.56)
        fFakOut=1;
    else
    {
        if (fFakIn>=0.55)
            fFakOut=4;
        else
        {
            if (fFakIn>=0.52)
                fFakOut=16;
            else
            {
                if (fFakIn>=0.49)
                    fFakOut=49;
                else
                {
                    cout << "Zu kleiner Faktor"<<endl;
                }
            }
        }
    }
}
}
```

B.3.5 PitchDetection

```
// PitchDetection
// durch die Übergabe des Files, wird die ganze Berechnung
// für die schnellere Wiedergabe gestartet.
// Resultat: ein neuer float des Ausgangsignals
//*****

void fastplay::PitchDetection(float * fSignalIn){
//Angaben über die Sprache
float flangeSig=(float)lSignalSize/(InfoWave.nChannels*InfoWave.nSamplesPerSec),
    flaengeFak=flangeSig*(float)fFak;

float fFmin=74.0,
    fFmax=500.0;

double dFrame=0.05;

int idSample=100;
int iTmax=(float)(1/fFmin)*InfoWave.nSamplesPerSec,
    iTmin=(float)(1/fFmax)*InfoWave.nSamplesPerSec,
    idT=dFrame*(int)InfoWave.nSamplesPerSec;

float * fx = new float[idT],
    * fy = new float[idT],
    * fKor = new float [lSignalSize],
    * fmaxKor = new float [lSignalSize],
    * fResultat = new float [lSignalSize],
    * fNeg = new float [lSignalSize],
    * fPositive = new float [lSignalSize];

fMaxArray= new float [lSignalSize];
fSignalOut = new float [lSignalSize];

float fxSum=0,
    fySum=0,
    flaengeRes=0;

int istartSample=iTmax,
    istartFrame=istartSample+1,
    istopSample=lSignal-idT-1,
    ipos=0,
    ipos0=100,
    inewStart=0,
    iCut=0,
    iposNew=0;
z=0;
while (istartFrame<istopSample){

    flaengeRes=flaengeRes+(float)(iCut/(float)(InfoWave.nChannels*InfoWave.nSamplesPerSec));
    //cout <<flaengeRes<<endl;
    if (flaengeRes<flaengeFak){
```

```

float fxSum=0;
float fySum=0;
int iendFrame = istartFrame+idT-1;

for (int i=0; i<(idT-1);i++){
    fx[i] = fSignalIn[iendFrame-idT+i];
    fxSum=fxSum+(fx[i]*fx[i]);
}

int idTmin=idTmin;
int q=0;
float fMax=0;

while (idTmin<=iTmax){
    float fySum=0;
    float fAuto=0;
    float fAutoKor=0;
    for (int l=0; l<(idT-1);l++){
        fy[l] = fSignalIn[iendFrame-idTmin-idT+l];
        fySum=fySum + (fy[l]*fy[l]);
    }
    for (int j=0; j<(idT-1);j++){
        fAuto = fAuto + fx[j]*fy[j];
    }

    fAutoKor=(float)(fAuto/(sqrt(fxSum*fySum)));
    fKor[q] = fAutoKor;
    if (fKor[q]>fMax) {

        fMax=fKor[q];
        ipos=q;
    }
    idTmin=idTmin+1;
    q++;
}

ipos=ipos+iTmin;
fMaxArray[z]=ipos;

if ((ipos-ipos0)*(ipos-ipos0)<=fFakOut){
    istartFrame=istartFrame+ipos0;

    for (int i=0;i<=(ipos0-1);i++){

        fNeg[i]=fSignalIn[istartFrame-ipos0+i]*(1-(i/(float)ipos0));
        fPositive[i]=fSignalIn[istartFrame+i]*(i/(float)ipos0);
        fSignalOut[inewStart+i]=fNeg[i]+fPositive[i];
    }
    iCut=ipos0;
}
else {
    iCut=0;
    for (int j=0;j<=(ipos0-1);j++){
        fSignalOut[inewStart+j]=fSignalIn[istartFrame+j];
    }
}
}
else {
    iCut=0;
    for (int j=0;j<=(ipos0-1);j++){
        fSignalOut[inewStart+j]=fSignalIn[istartFrame+j];
    }
}

istartFrame=istartFrame+ipos0;
inewStart=inewStart+ipos0;
ipos0=ipos;

z++;
}

lOutSize=inewStart;
cout <<"Faktor: " <<(float)lOutSize/(float)lSignal <<endl;
cout <<"laenge des Output-Signals: " << (float)inewStart/(float)InfoWave.nSamplesPerSec <<endl;
}

```

B.3.6 FloatToChar

```

// FloatToChar
// Berechnet des Ausgangs-Float wieder den Char für

```

```
// das Wav-File
//*****
void fastplay::FloatToChar(){

int k = 0;
cSignalOut = new char[lOutSize+1];
cSignalOut[lOutSize] = '\0';
while (k<lOutSize) {

    //Normierung
    //Big to Little Indian
    if (fSignalOut[k]<0)
        fSignalOut[k]=1+fSignalOut[k];
    else
        fSignalOut[k]=-1+fSignalOut[k];

    fSignalOut[k]=fSignalOut[k]*ibit;
    cSignalOut[k] = (char)fSignalOut[k];
    k++;

}
}
```

B.3.7 WriteWaveFile

```
//          WriteWaveFile
// schreibt das neue Wav-File, mit der neuen Information
// über das Ausgangsignal (Länge und Daten)
//*****
void fastplay::WriteWaveFile(char * FileName){

if (InfoWave.nChannels==2){
    InfoWave.nChannels=InfoWave.nChannels*(float)0.5;
    InfoWave.nAvgBytesPerSec=InfoWave.nAvgBytesPerSec*(float)0.5;
    InfoWave.nBlockAlign=InfoWave.nBlockAlign*(float)0.5;
}

ofstream hFile;

hFile.open(FileName, ios::binary);
if (hFile==NULL){
    cout << "Kann File nicht öffnen";
}
hFile.write(rID, 4);
hFile.write ((char*)&lFileSize, 4);
hFile.write(wID, 4);
hFile.write(fID, 4);
hFile.write((char*)&lWaveFormatSize, 4);
hFile.write((char*) &InfoWave, lWaveFormatSize);
hFile.write(dID, 4);
hFile.write((char*)&lOutSize, 4);
hFile.write(cSignalOut, lOutSize);
}
}
```

B.3.8 WriteTxtFile

```
//          WriteTxtFile
// zur Kontrolle der Resultate kann ein txt-File mit
// den gewünschten Infos geschrieben werden, welches
// in Matlab dargestellt werden kann
//*****
void fastplay::WriteTxtFile(char * filename) {

    ofstream TxtFile;

    TxtFile.open(filename, ios::out);

    if(!TxtFile)
        cerr << "Fehler beim Öffnen der *.txt Datei";
    for(int i = 0; i < z; i++) {

        TxtFile << fMaxArray[i] << endl;

    }

    TxtFile.close();
}
}
```

C File-Struktur

Die nachfolgenden Sprachsignalen wurden zur Auswertung der Semesterarbeit benötigt. Alle Verwendeten Signale sind im Verzeichnis ../Signale abgelegt.

Resultate der Matlab-Funktionen

Orginal	Funktion faster	Funktion FastPlay
../Signale/Orginal	../Signale/Matlab/Faster	../Signale/Matlab/FastPlay
h_bier.wav	h_bier_faster.wav	h_bier 0.7.wav
ichbin.wav	ichbin_faster.wav	ichbin 0.7.wav
man.wav	man_faster.wav	man 0.7.wav
message.wav	message_faster.wav	message 0.7.wav
zerror.wav	zerror_faster.wav	zerror 0.7.wav
moephone.wav		moephone 0.7.wav
aufforderung.wav		aufforderung 0.7.wav

Resultate der C++-Funktion

Orginal	TDHS
../Signale/Orginal	../Signale/Visual
h_bier.wav	h_bier 0.7.wav
ichbin.wav	ichbin 0.7.wav
man.wav	man 0.7.wav
message.wav	message 0.7.wav
zerror.wav	zerror 0.7.wav
moephone.wav	moephone 0.7.wav
aufforderung.wav	aufforderung 0.7.wav
2_frauen.wav	2_frauen 0.7.wav
katzen.wav	katzen 0.7.wav
woman.wav	woman 0.7.wav
totemuegerli1.wav	totemuegerli1 0.7.wav
totemuegerli.wav	totemuegerli 0.7.wav