Institut für Signalverarbeitung
und Sprachkommunikation
Technische Universität Graz

Bachelorarbeit: DSP Laborübung

Student / Betreuer:

Peter Kapfer / Dipl.Ing. Bernhard Geiger

Inhaltsverzeichnis

1 Theeorieteil

1.1 Einführung zeitabhängige Fouriertransformation

Bei der klassischen Fouriertransformation wird für die Berechnung das gesamte Signal herangezogen. Ist dieses Signal nicht stationär, so ist die Beschreibung durch die klassische Fouriertransformation nur zum Teil repräsentativ. Ändert sich nämlich beispielsweise die Frequenz des Signals in einem Zeitabschnitt, so wirkt sich dies auf das gesamte Spektrum aus. Um zu wissen wann welche Frequenzanteile auftreten, muss eine zeitabhängige Fouriertransformation verwendet werden. Nichtstationäre Signalmodelle werden zur Darstellung von Radar-, Sonar- oder auch Sprachsignalen benötigt. Verfahren, die eine gleichzeitige Analyse des Zeit- und Frequenzverhaltens ermöglichen, werden unter dem Begriff zeitabhängige Fouriertransformation zusammengefasst. Zu diesen Verfahren zählen u.a. die Kurzzeit-Fouriertranformation, die Gábortransformation, die Wignertransformation oder die Wavelettransformation. Es wird in den folgenden Kapiteln nur auf die Kurzzeit-Fouriertranformation und weiterleitend auf die Wavelettransformation eingegangen ([2], Kapitel 4.6).
In Abbildung 1) sind zwei Signale und ihre Spektren dargestellt. In 1.a) sind die Frequenzanteile stationär und in 1.b) nicht stationär. Aus dem Spektrum des nichtstationären Signals ist nicht zu erkennen, wann die unterschiedlichen Frequenzanteile auftreten.

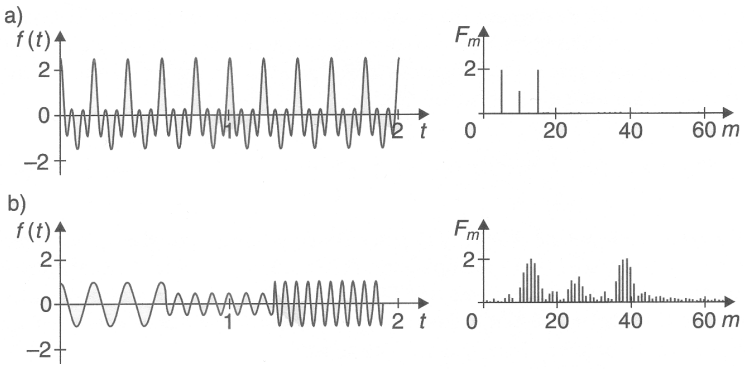

Abbildung 1: Zwei Signale im Zeit- und Frequenzbereich. a) stationäres Signal. b) nicht stationäres Signal (entnommen aus [2], S.193).

Um das Ergebnis einer zeitabhängige Fouriertransformation darzustellen, kann ein Spektrogramm verwendet werden. Hier wird auf der Abszisse die Zeit, auf der Ordinate die Frequenz und die Intensität des Signals, entweder bei einer dreidimensionalen Darstellung auf die dritte Koordinate oder bei einer zweidimensionalen Darstellung als Grau- oder Farbskala aufgetragen. In Abbildung 7) ist solch ein Spektrogramm dargestellt ([2], Kapitel 4.6).

1.2 Kurzzeit-Fouriertransformation

1.2.1 Einführung, Analyse

Mit dem Konzept der Kurzzeit-Fouriertransformation (*engl.* Short-Time Fourier Transform, STFT) ist nun eine gleichzeitige Analyse des Zeit- und Frequenzverhalten eines nichtstationären Signals, wie zum Beispiel Sprache, möglich. Das Signal $x[n]$ wird mit einer Fensterfunktion $w[n]$ (*siehe* Kapitel: Fensterung) in mehrere Signalblöcke

$$x_m[n] = x[n + mH]w[n], \qquad n, m \in \mathbb{Z} \tag{1}$$

unterteilt. H ist hier die Anzahl der Werte zwischen zwei Signalblöcken, auch "hop size" genannt. Formel (2) zeigt die Kurzzeit-Fouriertransformierte $X_m[k]$ des Signals $x[n]$ zu dem Zeitpunkt m und Frequenz ω_k mit $k = 0, 1, ..., N-1$ und $\omega_k = \frac{2\pi k}{N}$, wobei N die Anzahl der Samples eines Signalblocks ist. Für die Transformation der Signalblöcke in den Frequenzbereich wird die DFT (*engl.* discrete Fourier Transformation) verwendet. Man kann sagen, die STFT ist eine zweidimensionale Darstellung (Zeitpunkt m, Frequenz ω_k) des eindimensionalen Signals $x[n]$.

$$X_m[k] = \sum_{n=0}^{N-1} x[n + mH]\ w[n]\ e^{-j\omega_k n} \tag{2}$$

$X_m[k]$ kann nun auf zwei Arten interpretiert werden. Einerseits als Serie von Spektren und andererseits als Ausgang einer Filterbank mit N komplexwertigen Bandpassfiltern ([3], Kapitel 1.3).

1.2.2 Fensterung

Wird wie bei der STFT ein Signal durch Fensterung in Signalblöcke unterteilt, ist die Wahl des Fenstertyps und die Größe des Fensters für das resultierende Spektrum relevant. Die Größe bzw. die Länge des Fensters hängt von zwei Faktoren ab. Einerseits von dem Abtastintervall bei der Diskretisierung des zeitkontinuirlichen Signal $\Delta t = T_A$ und der Anzahl der Samples N, die innerhalb des Fensters liegen sollen. Daraus ergibt sich die Länge des Fensters oder auch Beobachtungsintervall $T_B = N\ T_A$. Es sei zu Beachten, dass bei der Verwendung der DFT zur Berechnung der STFT Spektren, das Beobachtungsintervall T_B und die Frequenzauflösung $\Delta\omega$ in einem fixen Zusammenhang stehen.

$$T_B \cdot \Delta\omega = const. \tag{3}$$

Gleichungen von diesem Typ werden auch als Unschärferelation bezeichnet ([2], Kapitel 4.3.5).
Resultierend aus dieser Unschärferelation, kann bei gleichzeitiger Darstellung der Zeit- und Frequenzeigenschaft eines Signals in einem Spektrogramm, die Darstellung nur als Rechteck und nicht als scharfer Punkt geschehen. Diese "unscharfe" Auflösung ist in Abbildung 2) dargestellt ([2], Kapitel 4.6).
Durch Umformung der Gl. (3) auf

$$\Delta\omega = \frac{1}{T_B} = \frac{1}{N \cdot T_A} \tag{4}$$

erkennt man, dass die Frequenzauflösung $\Delta\omega$ bei bereits abgetasteten Signal $x[n]$ nur mehr von der Anzahl der Samples N abhängt (T_A ist durch die D/A - Wandlung festgelegt) ([2], Kapitel 4.3.5).

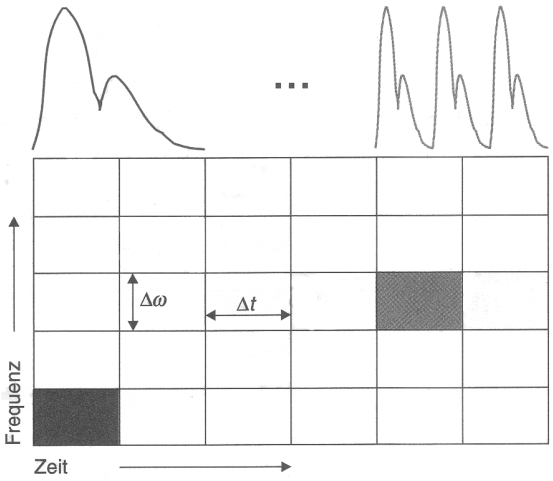

Abbildung 2: Zeit-Frequenz-Ebene der STFT (entnommen aus [2], S.196).

Man möchte natürlich die Fenstergröße gleich der Periode oder einem ganzzahligem Vielfachen eines quasistationären Bereiche des Ausgangssignal $x[n]$ wählen. Praktisch ist dies aber oft nicht möglich. Dadurch entstehen so genannte künstliche Diskontinuitäten (Signalsprünge). Diese Diskontinuitäten bewirken im Spektrum ein "Auslaufen" der Spektrallinien. Dieser Effekt wird als Leckeffekt (*engl.* leakage) bezeichnet und ist in Abbildung 3.b) erkennbar. Es werden im Spektrum Frequenzen dargestellt, die im Signal nicht vorhanden sind. Um dem Leckeffekt entgegen zu wirken, kann der Signalblock periodisch fortgesetzt und somit eine Annäherung (Abb. 3.c)) an das wahre Spektrum (Abb. 3.a)) erzielt werden. Bei Verwendung eines FFT- Algorithmus ist es notwendig die periodische Fortsetzung so zu wählen, dass die Bedingung nach einer Zweierpotenz der Abtastwerte erfüllt ist. Dies ist auch der Grund warum in Abbildung 3.c) bei der Annäherung $N = 1024$ gewählt wurde ([2], Kapitel 4.3.5).
Um die Auswirkungen der Diskontinuitäten noch weiter zu verringen, werden anstatt eines Rechteckfensters (Gl. (5), ([1], Kapitel 7.2.1)) Fensterformen gewählt, die das Signal an den Grenzen des Beobachtungsintervalls dämpfen. Bekannte Fenstertypen sind das Hamming- oder auch das Von-Hann-Fenster. Diese beiden Fenstertypen werden durch Gl. (6) ([1], Kapitel 7.2.1) beschrieben. Sie unterscheiden sich lediglich durch die Wahl des Parameters α. Von-Hann-Fenster: $\alpha = 0,5$. Hammingfenster: $\alpha = 0,54$.

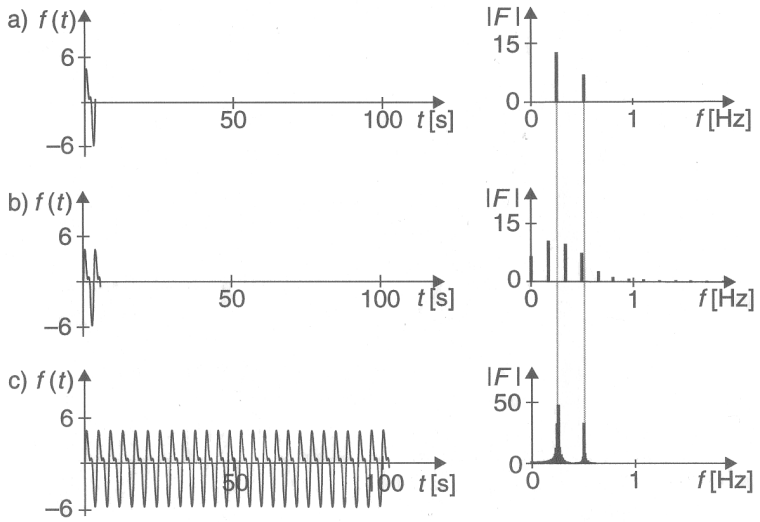

Abbildung 3: Das hier gefensterte Signal hat eine Periodendauer von genau 4s. Abtastintervall $T_A = 100ms$. Fenstertyp: Reckteckfenster.
In a) ist Beobachtungsdauer T_B gleich der Periodendauer. Hier ist das Spektrum korrekt dargestellt. (keine Diskontinuitäten)
In b) ist $T_B = 64 \cdot 100ms = 6,4s$, ungleich der Periodendauer oder einem Vielfachen davon. Dadurch tritt der Leckeffekt im Spektrum auf.
In c) wurde der Signalblock periodisch fortgesetzt um den Leckeffekt zu reduzieren. ($T_B = 1024 \cdot 100ms = 102,4s$) (entnommen aus [2], S.183)

$$w[n] = \begin{cases} 1, & 0 \leq n \leq N \\ 0, & \text{sonst} \end{cases} \tag{5}$$

$$w[n] = \begin{cases} \alpha + (1-\alpha)cos(\frac{2\pi \cdot n}{N}), & 0 \leq n \leq N \\ 0, & \text{sonst} \end{cases} \tag{6}$$

Eine weitere Auswirkung bei der Fensterung eines Signals ist die Reduzierung der Auflösung des Spektrums. So kann es vorkommen, dass mehrere Frequenzen im Spektrum zu einem Maximum verschmelzen und somit die Anzahl der Frequenzlinien reduziert wird. Nun gibt es bei den Fenstertypen zwei Merkmale, die die Auswirkungen auf das Spektrum sehr gut beschreiben. So ist die Reduzierung der Auflösung in erster Linie von der Breite der Hauptkeule der Fouriertransformierten des Fensters abhängig. Die Auswirkungen des Leckeffektes werden hingegen von der relativen Amplitude der Haupt- und Seitenkeulen bestimmt ([1], Kapitel 10.2.1).
In Tabelle 1) ist eine kurze Gegenüberstellung dieser Merkmale und in Abbildung 4) sind Beispiele für ein Rechteck-, Hamming- und Von-Hann- Fenster mit ihren zugehörigen Spektren dargestellt.

Typ des Fensters	ungefähre Breite der Hauptkeule	max. Amplitude der Seitenkeulen im Vergleich zur Hauptkeule (db)
Rechteck	$4\ \pi/(N+1)$	-13
Von-Hann	$8\ \pi/N$	-31
Hamming	$8\ \pi/N$	-41

Tabelle 1: Vergleich von häufig genutzten Fenstertypen. (adaptiert von [1], S.569)

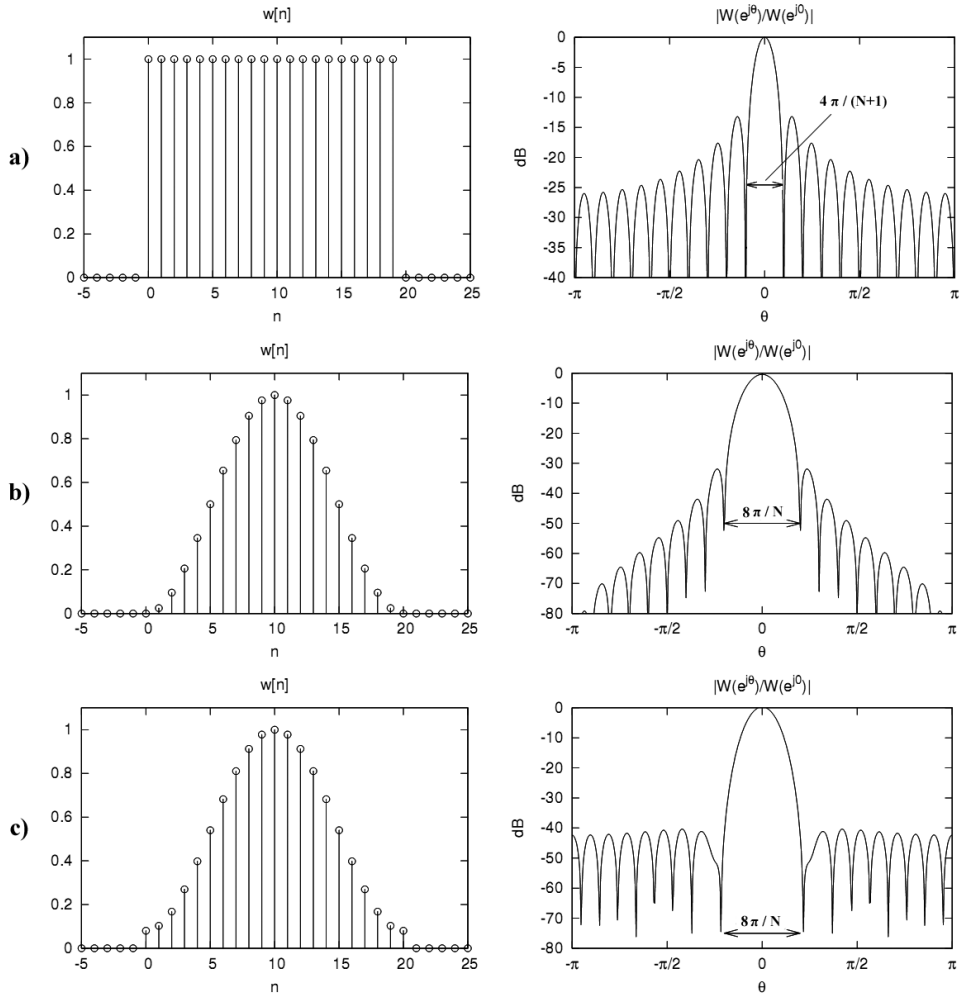

Abbildung 4: a) Rechteckfenster (N = 19) b) Von-Hann-Fenster (N = 20) c) Hamming-Fenster (N = 20) mit dazugehörigem Spektrum (adaptiert von [5], Kapitel 3)

1.2.3 Synthese

Nachdem das Zeitsignal $x[n]$ nun mit Gl. (2) analysiert wurde, kann man mit der STFT-Synthese das Signal wieder reproduzieren. Gl. (7) zeigt die so genannte "Overlap addition" (OLA) Methode zur Rückgewinnung des Zeitsignals. Es werden die einzelnen Signalblöcke mittels der IDFT rücktranformiert und anschließend aufsummiert ([9], Kapitel 6.1.5).

$$y[n] = \sum_{m=-\infty}^{\infty} \left[\frac{1}{N} \sum_{k=0}^{N-1} X_m[k] e^{jw_k n} \right] \delta[n - mH] \tag{7}$$

Abbildung 5) zeigt eine grafische Interpretation der OLA. Hier werden die einzelnen, schon rücktransformierten Signalblöcke zusammenaddiert. Die Summe ergibt das Ausgangssignal $y[n]$.

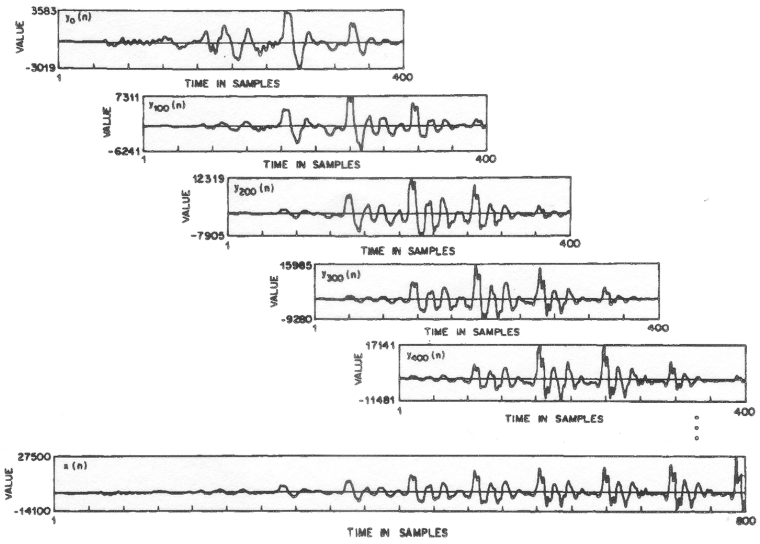

Abbildung 5: Grafische Interpretation der OLA Methode. Fensterfunktion: Hammingfenster der Länge N. Anzahl der Signalblöcke ist gleich 5. (entnommen aus [8])

1.2.4 Anwendungsfälle

Als Anwendungsfall wird die Analyse eines Sprachsignals gezeigt. In Abbildung 6) ist das offensichtlich nichtstationäre Sprachsignal des Satzes *"Two plus seven is less than ten"* dargestellt. Jede Zeile der Abbildung stellt einen 0,17 s langen Ausschnitt mit zugehöriger Lautschrift aus dem Signal dar ([1], Kapitel 10.5.1).

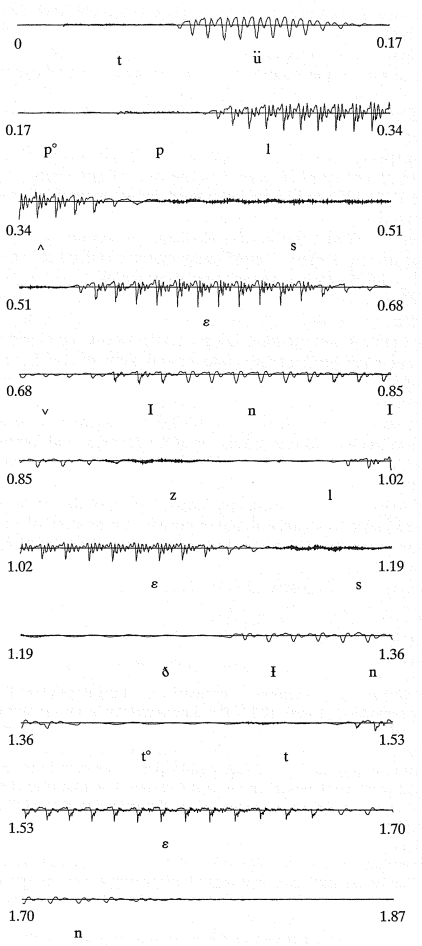

Abbildung 6: Sprachsignal des Satzes *"Two plus seven is less than ten"*(entnommen aus [1], S.861)

Hier ist zu erkennen, dass sich in Zeitintervallen von 30 bis 40 ms keine wesentlichen Unterschiede im Signalverhalten abzeichnen. Diese Bereiche können also als stationär angenommen werden. Sie werden mit einer Fensterfunktion in Signalblöcken unterteilt und auf diese wird dann einzeln die DFT angewendet. In Abbildung 7.a) ist das Breitbandspektrogramm und in 7.b) das Schmalbandspektrogramm des Signals aus Abbildung 6) dargestellt. Ein Breitbandspektrogramm erhält man bei einem relativ kurzen Fenster im Zeitbereich. Wobei dieses kleine Fenster im Zeitbereich eine schlechte Auflösung im Frequenzbereich zur Folge hat. Bei der Darstellung als Schmalbandspektrogramm wird ein größeres Fenster im Zeitbereich verwendet. Dadurch wird eine Erhöhung der Auflösung

im Frequenzbereich erreicht. Dies ist der zwingende Kompromiss zwischen der Auflösung im Zeit- und Frequenzbereich. Wird das Zeitfenster zu groß gewählt, können sich die Signaleigenschaften innerhalb des Fensters zu stark ändern und das Fenster kann nicht mehr als stationär angesehen werden. Ist das Zeitfenster hingegen zu klein, so ist eine schmalbandige Auflösung der Frequenzkomponenten nicht mehr möglich ([1], Kapitel 10.5.1).

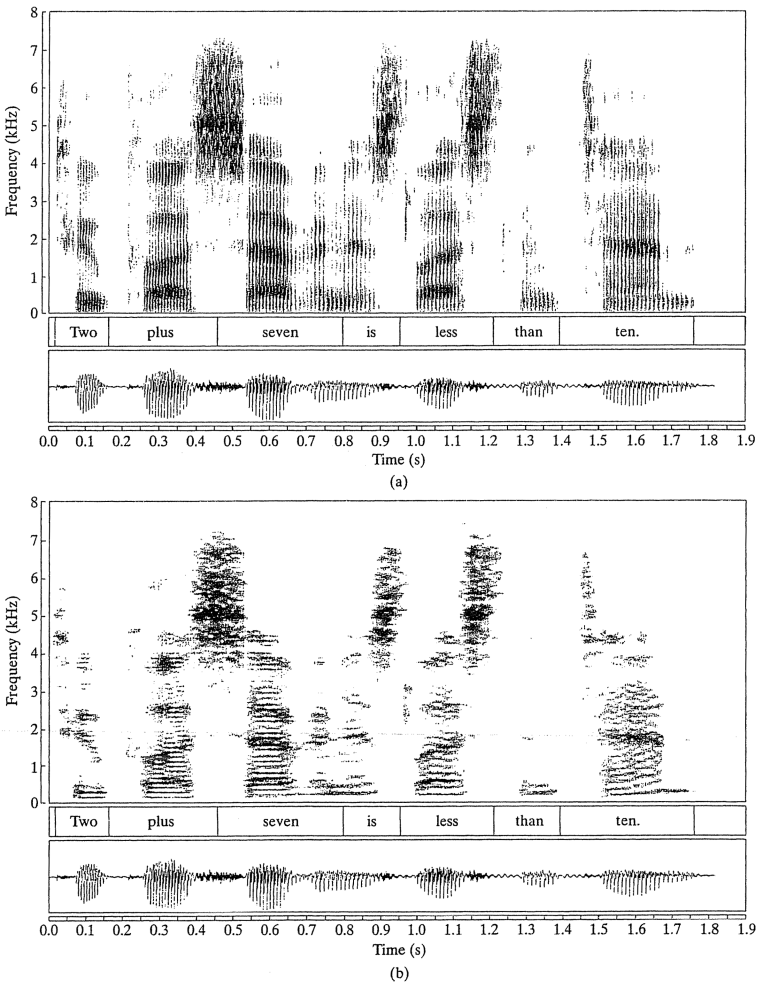

Abbildung 7: a) Breitbandspektrogramm (Hamming- Fenster mit Fensterbreite 6,7 ms (N = 108) und einem Zeitintervall von 1 ms (H = 16).) und b) Schmalbandspektrogramm (Hamming- Fenster mit N = 720 und H = 16.) des Sprachsignal aus Abbildung 6). Die Abtastrate des Signals beträgt 16 000 Abtastwerten/s. (entnommen aus [1], S.863)

Überall dort, wo das Auftreten von bestimmten Frequenzen zeitabhängig und dies zur Klassifizierung entscheidend ist, kann die STFT zur Anwendung kommen. Beispielsweise bei der Analyse von bioakustischen Signalen (*Bsp.* Herzklappenklänge, Vogelstimmen ([7], Kapitel 5.4), etc.).

1.3 Wavelettransformation

Die Wavelettransformation ermöglicht gleich wie die STFT, eine gleichzeitige Analyse im Zeit- und Frequenzbereich. Der Unterschied besteht darin, dass in mehreren Auflösungsstufen gearbeitet werden kann. In der Praxis tritt oft der Fall auf, dass tieffrequente Signalteile lange andauern, während hochfrequente Anteile nur kurz auftreten. Mit mehreren Auflösungsstufen kann man die kurz und hochfrequent auftretenden Signalanteile in der Zeit hoch auflösen, während man die langen und tieffrequent auftretenden Signalanteile in der Frequenz hoch auflöst. In Abbildung 8) ist eine schematische Zeit-Skalen- Ebene der Wavelettransformation dargestellt. Hier wird als Unterschied zur STFT anstatt der Frequenz die Skalierung (umgekehrt proportional zur Frequenz) aufgetragen ([2], Kapitel 4.7).

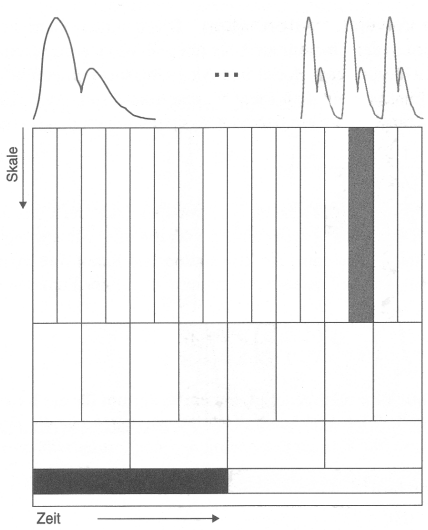

Abbildung 8: Zeit-Skalen-Ebene der Wavelettransformation.(entnommen aus [2], S.197)

Der Begriff Wavelettransformation kommt daher, dass so genannte Wavelet - Funktionen $\psi(t)$ für die Transformation bzw. Analyse eines Signals $x(t)$ verwendet werden. Als Wavelets gelten Funktionen, die folgende Bedingungen erfüllen: Das Wavelet muss endliche Energie besitzen (Gl.(8)) und die Fouriertransformierte $\psi(f)$ (Gl.(9)) von $\psi(t)$ muss die (Gl.(10), Konvergenzbedingung (*engl.* admissibility condition)) erfüllen. C_g in Gl.(10) wird als "admissibility constant" bezeichnet ([2], Kapitel 4.7).

$$E = \int_{-\infty}^{\infty} |\psi(t)|^2 \, dt < \infty \tag{8}$$

$$\psi(f) = \int_{-\infty}^{\infty} \psi(t) e^{\frac{-j(2\pi f)}{t}} dt \tag{9}$$

$$C_g = \int_{0}^{\infty} \frac{|\psi(f)|^2}{f} df < \infty \tag{10}$$

Diese Bedingung sagt im Wesentlichen aus, dass ein Wavelet $\psi(t)$ einen Mittelwert gleich 0 besitzen muss, d.h.:

$$\int_0^\infty \psi(t)dt = 0 \tag{11}$$

Wavelets sind folgernd Funktionen, die im weitesten Sinne schwingen und abklingen (kleine Wellen, *engl.* wavelets). Gl.(12) zeigt das häufig verwendete Wavelet, basierend auf der 2.Ableitung der gaußschen Glockenkurve $e^{\frac{-t^2}{2}}$.

$$\psi(t) = (1 - t^2)e^{\frac{-t^2}{2}} \tag{12}$$

Substituieren wir nun in Gl.(12) t mit $(t - b)/a$, so erhalten wir

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi(\frac{t-b}{a}). \tag{13}$$

Mit der Skalierung a und der Translation b können wir uns nun einen Satz aus Basisfunktionen erstellten, mit der das Signal $x(t)$ analysiert wird. Der Faktor $\frac{1}{\sqrt{a}}$ garantiert uns, dass alle verschieden skalierten Wavelets dieselbe Energie besitzen ([2], Kapitel 4.7).

Für weitere Erklärung sei hier auf die Literatur [10] verwiesen.

2 Arbeiten im DSP Labor

Zur Durchführung des praktischen Teils wurde im Wesentlichen als Hardware das DSP Board TMS320C6713 DSK (DSP Starter Kit) und die Entwicklungsumgebung Code Composer Studio v.3 (CCSv.3) verwendet. Für weitere Informationen über das DSP Board sei auf [6], Kapitel 2 verwiesen.
Das Ziel des Laborteils war ein Code Composer Studio Projekt für den STFT Übungsteil des "Digital Signal Processing Laboratory" zu erstellen. Mit Hilfe dieses DSP Programms sollen Studierende, die dieses Labor absolvieren die Grundzüge und insbesondere den Zusammenhang zwischen Zeit- und Frequenzauflösung der STFT verstehen lernen. In Kapitel 2.4) ist eine mögliche Übungsangabe angeführt. Als Ausgangspunkt wurde ein bereits existierendes Projekt verwendet. Für Informationen über die allgemeinen Bestandteile eines CCSv.3 Projekt sei auf [4], Kapitel 2 verwiesen. Das DSP Programm zur STFT Übung verwendet eine so genannte Block Processing Umgebung zur Bufferung und Verarbeitung des Eingangssignals. Im folgenden Kapitel wird auf dieses Konzept der Block Processing Umgebung weiter eingegangen.

2.1 Block Processing Umgebung, ISR (Interrupt Service Routine)

Es ist oft nicht effizient bzw. möglich den gesamten Eingang zu buffern und erst dann zu verarbeiten. So kommt man zwingend auf ein Abarbeitungsschema auf Block- Basis. Abarbeitungsschema auf Block- Basis bedeutet, dass nach Speichern einer gewissen Anzahl an eingehenden Samples in den DMA Buffer der DMA Controller des DSP Boards mittels Hardware Interrupt die Interrupt Service Routine ISR aufruft. Die ISR ist in `isr_short.c` implementiert. In dieser ISR wird unter anderem ein Working Buffer erzeugt und mit den Samples aus dem DMA Buffer befüllt. Dieser Working Buffer wird anschließend der Funktion `processing( (void *)workbuf)` übergeben. Nach Bearbeitung des Blocks wird der Woking Buffer wieder auf den DMA Buffer geschrieben. Weiters ist die Overlap Add Methode in der ISR schon implementiert und wird durch mehrere Konstanten, definiert in `blocks.h`, gesteuert [4].
Das Intervall indem die ISR aufgerufen wird, wird mit der "hop size" eingestellt. In der Block Processing Umgebung präsentiert durch die Konstante `HOPSZ`. Die Größe des Working Buffers wird durch `BLKSZ = NWBLOCKS*HOPSZ` festgelegt, wobei sich `NWBLOCKS` aus `OVERLAP` mit `NWBLOCKS = OVERLAP+1` ergibt. Mit der Konstanten `OVERLAP` kann die Überlappung der einzelnen Signalblöcke in Prozent eingestellt werden `(Überlappungsfaktor = OVERLAP/(OVERLAP+1))`.
Als weitere Konstante in `blocks.h` sei noch `NWSTFTCHANNEL` erwähnt. Wie im nächsten Kapitel beschrieben, soll eine Modifikation der STFT Spektren durchgeführt werden. Mit der Konstanten `NWSTFTCHANNEL` kann nun die Anzahl angegeben werden, wie viele Spektralwerte eines STFT Spektrums (STFT Channels) bei der Verwendung der Funktion `find_max_koeff_sort()` gesucht und bei der Funktion `pass_channel_by_index()` durchgelassen werden sollen.

2.2 STFT Laborübung

Aus Gleichung (4), Kapitel 1.2.2 zeigt sich, dass bei vorgegebener Abtastfrequenz die Frequenzauflösung nur mehr von der Anzahl der Abtastwerte N abängt.
Wo ist nun dieser Parameter N der STFT in der Implementierung zu finden?
Aus der durch die Block Processing Umgebung bereitgestellten Konstanten `HOPSZ` ergibt sich die Anzahl der Abtastwerte, die im DMA Buffer gespeichert werden, bis der Hardware Interrupt die ISR aufruft. Angenommen wir wählen `HOPSZ = 16` und `OVERLAP = 1`. Daraus ergibt sich die Anzahl der Blöcke `NWBLOCKS = OVERLAP + 1` zu 2. Die Anzahl der Samples im Working Buffer ergibt sich dann mit `BLKSZ = NWBLOCKS * HOPSZ` zu 32. Das heißt, die Anzahl der Abtastwerte N, wird mit `HOPSZ` und `OVERLAP` eingestellt. Der nächste Parameter, die Länge der FFT ergibt sich aus der `BLKSZ` mit `FFTSZ = BLKSZ`. Bei unserem Beispiel ergibt sich die Länge der FFT mit 32 Punkten.
Der letzte Parameter, der Fenstertyp, kann im Userfile `stft.c` in der `init()` und `processing()` Funktion eingestellt werden. In der `init()` Funktion können die Werte für ein Rechteck-, Hamming- oder Von-Hann-Fenster berechnet werden. In der `processing()` Funktion wird dann (ausser bei dem Rechteckfenster) der Working Buffer mit diesen Fensterkoeffizienten multipliziert.
Wie kann nun die Unschärferelation zwischen Zeit- und Frequenzauflösung beobachtet werden?
Als erstes wird ein Audiosignal (CD-Player, MP3 - Player oder sonstige Audioquelle) an den 3,5 mm Klinkeneingang angeschlossen. Anschließend wird in `blocks.h` eine `HOPSZ` und ein `OVERLAP`- Faktor eingestellt. Weiters soll in `blocks.h` die Konstante `NWSTFTCHANNEL` gesetzt werden. Mit dieser Konstante wird die Anzahl der STFT Channels festgelegt, die anschließend im Frequenzbereich durchgelassen werden sollen. In `stft.c` wird nun nach dem Wechsel in den Frequenzbereich mit den Funktionen `find_max_koeff_sort(koeff, max_koeff_sort)` und `pass_channel_by_index()` das Spektrum so modifiziert, dass STFT Channels gemutet werden. Durch dieses Nullsetzen bzw. Durchlassen einzelner Spektralwerte verschlechtert sich natürlich die Qualität des Audiosignals nach der Rücktransformation in den Zeitbereich. Wie stark die Qualität beeinflusst wird, hängt jetzt von der Anzahl und Energie der Spektralwerte ab, die durchgelassen werden.
Lässt man `NWSTFTCHANNEL` gleich und verringert die Zeitauflösung (Vergrößerung von N bzw. `HOPSZ`), so kann man eine Verschlechterung des Audiosignals feststellen. Diese Verschlechterung beruht jetzt darauf, dass durch die Vergrößerung der `HOPSZ` die FFT auch mehr Werte berechnet (feinere Frequenzauflösung), aber gleich viel STFT Channels durchgelassen werden und dadurch mehr Frequenzen unterdrückt werden. Um bei Verringerung der Zeitauflösung die gleiche Qualität des Audiosignals nach der Rücktransformation zu erhalten, muss `NWSTFTCHANNEL` auch erhöht werden, um mehr STFT Channels durchzulassen.

2.3 Dokumentation Userfiles

- **fft.c:** Hier sind die benötigten Funktionen für die Berechnung der FFT und zusätzliche Funktionen wie z.B. die Berechnung der Fensterkoeffizienten implementiert.

Die folgenden Funktionen

```
void fft_init( fft16Params*,unsigned fft_size)

void fft( fft16Params*, short buf[])

void ifft( fft16Params*, short buf[])

unsigned short log_dual_size( unsigned fft_size)

void bitrev_index(short ind[], unsigned nbits)

void calc_sine_coeffs(short ptr1[], unsigned fft_size)
```

und das Strukt `fft16Params`, definiert in `fft.h`, sind aus dem Projekt welches als Basis verwendet wurde übernommen worden und werden daher nicht extra dokumentiert. Es werden ausschließlich die neu hinzugekommenen Funktionen beschrieben.

```
void hamming_window( float buf[], unsigned short size)
{
  unsigned short i=0;
  for(i=0;i<size;i++)
    buf[i] = 0.54 - 0.46*cosf(2*PI*i/size);
}
```

Die Funktion `hamming_window()` berechnet die Fensterkoeffizienten laut Gl. (6) mit $\alpha = 0,54$. Die Anzahl der benötigten Koeffizienten wird mit `size` übergeben und in `buf[]` gespeichert.

```
void von_hann_window( float buf[], unsigned short size)
{
  unsigned short i=0;
  for(i=0;i<size;i++)
    buf[i] = 0.5- 0.5*cosf(2*PI*i/size);
}
```

In der Funktion `von_hann_window_window()` werden wie in der Funktion `hamming_window()` die Fensterkoeffizienten laut Gl. (6) berechnet. Der Unterschied besteht durch die Wahl von α mit $0,5$.

```
void find_max_koeff_sort(float koeff[], int max_koeff_sort[])
{
  int i = 0;
  int j = 0;
  int max_koeff_index = 0;
  float max_koeff = 0;
  float tmp[FFTSZ];

  // calc the maxima from the coeff and save them temporary
  for(i=0;i<FFTSZ;i++)
  {
    tmp[i] = (koeff[2*i]*koeff[2*i]) + (koeff[2*i+1]*koeff[2*i+1]);
  }

  for(j=0; j < NWSTFTCHANNEL; j++)
  {
    for(i=0;i<FFTSZ;i++)
    {
      if(tmp[i] > max_koeff)
      {
        max_koeff = tmp[i];
        max_koeff_index = i;
      }
    }
    max_koeff_sort[j] = max_koeff_index; // save max coeff
    tmp[max_koeff_index] = 0; // delete max coeff out of search
    max_koeff = 0;
  }
}
```

`find_max_koeff_sort()` berechnet in der ersten `for`-Schleife aus den übergebenen Fourier - Koeffizienten in `fftbuf[]` die Energie der Koeffizienten und speichert sie in `tmp[]` zwischen. In der zweiten `for`-Schleife werden nun die Indexwerte der Koeffizienten mit der maximalen Energie in max_koeff_sort[] gespeichert. Wie viele Koeffizienten gesucht und deren Indexwerte gespeichert werden sollen, wird mit der Konstanten `NWSTFTCHANNEL` festgelegt. Es ist keine Sortierung nötig, da nur eine Menge an maximalen Koeffizienten gesucht ist. Hier sei noch anzumerken, dass von dem übergebenen Array `fftbuf[]` der Betrag gebildet und dafür kein zusätzliches Array verwendet wird. Das heist das Array wird manipuliert. Dies muss beim Funktionsaufruf berücksichtigt werden, indem eine Kopie des wahren `fftbuf[]` übergeben werden muss.

```
void pass_channel_by_index(int max_koeff_sort[], short fftbuf[])
{
  int i = 0;
```

```
  int j = 0;
  int pass_channel_flag = 0;
  for( i=0; i<2*FFTSZ; i++)
  {
   // check if index i is equal to koeff index
   for(j=0; j < NWSTFTCHANNEL; j++)
   {
      if(max_koeff_sort[j] == i)
      {
        pass_channel_flag = 1;
        break;
      }
      else
        pass_channel_flag = 0;
   }

   // mute STFT Channel if it is not in max_koeff_sort[]
    if(!pass_channel_flag)
    {
      fftbuf[2*i]   = 0;
      fftbuf[2*i+1] = 0;
    }
  }
}
```

Die Funktion `pass_channel_by_index()` setzt alle Fourier - Koeff. bzw. STFT Channels ausser jene auf Null, die mit den Indizes in `max_koeff_sort[]` übergeben werden.

stft.c: Implementierung der `init()` und `processing()` Funktion.

```
#define _TI_ENHANCED_MATH_H 1
#include <math.h>

#include "fft.h"
#include "blocks.h"

/*   global variables */
fft16Params fft16; // fft parameters
float window[FFTSZ]; //Hamming window coefficients
float koeff[2*FFTSZ*DATSZ];
int max_koeff_sort[NWSTFTCHANNEL];
```

Inlcudes und Defintion der benötigten globalen Variablen. Dokumentation ist in Kommentaren ausgeführt.

```
void init( void)
{
  fft_init( &fft16, FFTSZ);
  /* choose window
  commet all function calls out to get rectangle window
  (automaticlly choosen by block processing environmentl)
  */
  //hamming_window( window, FFTSZ);
  //von_hann_window( window, FFTSZ);
}
```

In der `init()` Funktion wird unter anderen das Strukt `fft16Params` für die FFT Berechnung der einzelnen Signalblöcke angelegt. Weiters kann hier durch Ein-/Auskommentieren der jeweiligen Programmzeile entschieden werden, welcher Fenstertyp verwendet werden soll. Implementiert sind das Hamming- und das Von-Hann- Fenster. Wenn beide Fenstertypen auskommentiert werden, wird automatisch ein Rechteckfenster, bedingt durch das Abarbeitungsschema der Block Processing Umgebung, verwendet.

```
void processing( void *ptr )
{

  int i = 0;
  int j = 0;
  short *sigbuf = ptr;
  short *fftbuf;
```

```
  fftbuf=(short*)malloc(2*FFTSZ);
  if (fftbuf )
  {
    /* analysis window */
//    for(i=0; i<fft16.fft_size; i++)
//    {
//      sigbuf[i]= (short)((float)sigbuf[i]*window[i]);
//    }
```

Nach Definition von diverses Hilfsvariablen wird mit `malloc` ein Array mit zweifacher `FFTSZ` für die Berechnung der FFT angelegt. Sind Fensterkoeffizienten in der `init()` Funktion angelegt worden, so kann hier durch Ein-/Auskommentieren der `for`- Schleife eine Fensterung mit einem der Fenstertypen durchgeführt werden.

```
    /* add imaginary part to signal,
     clear sigbuf to check that is filled with new values after processing
     */
   for(i=0; i<fft16.fft_size; i++)
   {
      fftbuf[2*i]=sigbuf[i];
      fftbuf[2*i+1]=0;
      sigbuf[i] = 0;
    }

   fft(&fft16,fftbuf);
```

Für die DFT Analyse mittels FFT Algorithmus wird zu dem realen Eingangssignal ein Imaginärteil hinzugefügt und Null gesetzt. Zu jedem reell wertigen Eingangssample ist nun ein Imaginärteil gleich null im Array `fftbuf[]` gespeichert. Anschließend wird mittels Funktionsaufruf von `fft(&fft16,fftbuf)` die FFT berechnet und das Ergebnis im `fftbuf[]` gespeichert. Um sicher zu gehen, dass nach der `processing()` Funktion nur Samples auf den DMA buffer geschrieben werden, die tatsächlich bearbeitet wurden (und nicht ohne Bearbeitung durchgeschleift wurden), wird in der `for`- Schleife das Array `sigbuf[i]` mit den Originalsamples gleich Null gesetzt.

```
   for(i=0; i<2*fft16.fft_size; i++)
   {
     koeff[i] = fftbuf[i];
   }
```

```
   /* take set of max koeff or one fixed value
    for fixed value, NWSTFTCHANNEL in blocks.h have to be 1
    */
    //max_koeff_index[0] = 4;
    find_max_koeff_sort(koeff, max_koeff_sort);

    // only pass channels, indexed by max_koeff_sort array
    pass_channel_by_index(max_koeff_sort, fftbuf);

   ifft(&fft16,fftbuf);

    // only real part is given back to DMA buffer
   for(i=0; i<fft16.fft_size; i++)
   {
     sigbuf[i] = fftbuf[2*i];
   }

   free(fftbuf);
  }

}
```

In der ersten `for`- Schleife wird das berechnete Spektrum in den Zwischenspeicher `koeff[]` kopiert. Dieses Array wird anschließend beim Aufruf von `find_max_koeff_sort(koeff, max_koeff_sort)` übergeben. Es muss eine Kopie von `fftbuf[]` verwendet werden, da die Funktion das Array manipuliert. Nun kann man durch Ein-/Auskommentieren der beiden Codezeilen max_koeff_index[0] = 4 und `find_max_koeff_sort(koeff, max_koeff_sort)` entschieden werden, ob nur ein oder eine Menge an den stärksten STFT Channels durchgelassen bzw. nicht gemutet werden soll. Natürlich kann auch nur der stärkste STFT Channel durchgelassen werden. Dazu muss die Konstante `NWSTFTCHANNEL` in `blocks.h` auf den Wert 1 gesetzt werden und die Codezeile `find_max_koeff_sort(koeff, max_koeff_sort)` einkommentiert sein. Anschließend wird mit dem Aufruf von `ifft(fft16,fftbuf)` die IDFT / IFFT berechnet und das Ergebnis in `fftbuf[]` gespeichert. Da nun `fftbuf[]` komplexe Werte beinhaltet und wir nur den reell wertigen Teil benötigen, werden in der letzten `for`- Schleife nur die geraden Samples in den `sigbuf[]` bzw. dem Working Buffer, welcher dem DMA Buffer zur Ausgabe entspricht, gespeichert.

2.4 Übungsangabe

Experiment: **Short-Time Fourier Transform** Equipment: PC + DSK, soundcard or CD/MP3 player, headphones, Software: CCS

1. Plug the output of the PC soundcard (or a CD/MP3 player) to the DSK input and headphones to the DSK output.

2. In CCS, load the project file `stft.pjt`. Look for the constants `HOPSZ`, `OVERLAP`, `OLA`, `FFTSZ` and `NWSTFTCHANNEL` in `blocks.h`. Observe how these constants change the properties of the block processing environment.

3. Make sure that your code uses $N = 32$ Samples and 50% overlap. How do you achieve this? What is the right value for the constant `OLA`? Look for this constant in `isr_short.c`! Do not run the program yet.

4. Look at the function `processing()` in `stft.c`. You can see a loop for muting all STFT filter bank channels but one. Build and run the program and listen to the output signal.

5. Modify the program to let several channels pass: Use the given functions (see `fft.c`) to find the coefficients with the maximum energy and let these channels pass. Choose different values for the constant `NWSTFTCHANNEL` and listen to the output signal.

6. Adjust a finer time resolution, but don´t change the other constants of the block processing environment. Is the quality of the output signal getting better or worse? Why is the quality changing? Hint: Think about the relationship between time and frequency resolution of the STFT (uncertainty relation).

Literatur

[1] OPPENHEIM A.V., SCHAFER R.W., and BUCK J.R. Zeitdiskrete Signalverarbeitung. *Prentice Hall, Pearson Studium*, 2004. 4, 5, 6, 8, 9

[2] MEFFERT B. and HOCHMUTH O. Werkzeuge der Signalverarbeitung. *Pearson Studium*, 2004. 2, 3, 4, 5, 10, 11

[3] FELDBAUER C., KÉPESI M., WITRISAL K., and RANK E. Multirate Signal Processing. *TU Graz, SPSC Laboratory Handout*, 2005. 3

[4] FELDBAUER Ch. Real-Time Block Processing environment. *TU Graz, SPSC Laboratory Handout*, 2005. 12

[5] SHUTIN D., WITRISAL K., RANK E., KÉPESI M., and MEISSNER P. Discrete Fourier transform. *TU Graz, SPSC Laboratory Handout,*, 2009. 6

[6] PERNKOPF F., MEISSNER P., ROMSDORFER H., and GEIGER B. Introduction. *TU Graz, SPSC Laboratory Handout*, 2011. 12

[7] TEMBROCK G. Akustische Kommunikation bei Säugetieren. *Darmstadt: Wissenschaftliche Buchgesellschaft*, 1996. 9

[8] ALLEN J.B. and RABINER L.R. A unified approach to short-time Fourier analysis and synthesis. *Proc. IEEE, vol. 65*, 1977. 7

[9] RABINER L.R. and SCHAFER R.W. Digital Processing in Speech Signals. *Prentice-Hall, Inc., Englewood Cliffs*, 1978. 7

[10] ADDISON P.S. The Illustrated Wavelet Transform Handbook. *Institute of Physics Publishing*, 2002. 11