Berner Fachhochschule

Hochschule für Technik und Informatik HTI
Fachbereich Elektro- und Kommunikations-technik

Beat Detection - akustisch und optisch

Diplomarbeit Bildverarbeitung

Betreuer: Dr. Franz Bachmann
Peter Aeschimann

Experte: Dr. Hansjörg Klock

Verfasser: Michael Bernhard
Adrian Straubhaar

Burgdorf, im Januar 2004

Zusammenfassung

Was geschieht, wenn wir in einem Raum sitzen, nichts weiter zu tun haben, und von irgendwo her erklingt Musik? In den meisten Fällen beginnt unser Fuss zu zucken, der Kopf zu wippen oder die Hand klopft rhythmisch auf die Stuhllehne. Ganz automatisch. Da wir schon von jüngstem Alter an mit Musik in Kontakt kommen, entwickeln wir ein Gefühl für sie, besonders aber für Rhythmus und Takt. So lässt sich erklären, warum die meisten Leute in der Lage sind zu einem Musikstück den Takt zu klopfen.

Der musikalische Puls ist aber auch eine wichtige Kenngrösse bei der digitalen Analyse von Musik. Die Bestimmung des Takts bildet die Grundlage für weitere musikalische Elemente. Was aber für Menschen selbstverständlich und einfach ist, muss dem Computer durch geeignete Algorithmen implementiert werden. Denn er hat kein musikalisches Verständnis und auch keine langjährige Erfahrung im Musik hören, auf die er zurückblicken könnte.

Unsere Aufgabe in dieser Diplomarbeit war die Entwicklung und Implementation einer optischen und einer akustischen Takterkennung, sowie die Vereinigung beider Teile in einem Demosystem. Als Grundlage diente die vorangegangene Semesterarbeit, in der wir eine akustische Beat Detection in Matlab implementiert hatten. Diese galt es zu verbessern und um die oben genannten Elemente zu ergänzen. Ausserdem sollte das entstandene System echtzeitfähig sein.

Die Entwicklung der Algorithmen erfolgte weiterhin mit der Mathematiksoftware Matlab, während die endgültige Version in der Programmiersprache C++ realisiert wurde. Für die Echtzeit-Implementation verwendeten wir den objektorientierten Ansatz. Um nicht an eine bestimmte Hardware gebunden zu sein, sprechen wir die verschiedenen Komponenten mit Hilfe von DirectX an.

Entstanden ist nun ein Demoprogramm, welches beide Takterkennungen enthält. Die akustische Beat Detection erkennt den Takt verschiedenster Musikstücke von Klassik bis Pop und gibt ihn auf dem Bildschirm aus. Das Augenmerk richtete sich dabei auf eine möglichst grosse Vielfalt verschiedener Stilrichtungen. Die Musikstücke können von vorhandenen Audio- oder Video-Dateien eingelesen werden, aber auch ab CD oder live über ein Mikrofon.

Der Videoteil ist in der Lage die Bewegungen einer Dirigentin auszuwerten und den Takt ebenfalls auf dem Bildschirm auszugeben. Dabei spielt es keine Rolle, ob eine vorbereitete Video-Datei verwendet wird, oder ob ein Dirigent live von einer Digitalkamera gefilmt wird.

Burgdorf, im Januar 2004

Michael Bernhard Adrian Straubhaar

Übersichtsverzeichnis

Inhaltsverzeichnis

1 Einleitung

1.1 Einführung

Der musikalische Puls ist eine wichtige Kenngrösse von Musikstücken. Bei der digitalen Analyse von Musik bildet er eine Grundlage, auf welcher weitere musikalische Elemente aufbauen. Die meisten Menschen sind in der Lage den Puls zu erkennen und zum Beispiel als Dirigierbewegung wiederzugeben oder im richtigen Takt zu klatschen. Einem Computer fehlt dieses musikalische Verständnis. Es ist durch geeignete Algorithmen zu ersetzen.

1.2 Weiterführung der Semesterarbeit

In unserer Semesterarbeit im sechsten Semester beschäftigten wir uns erstmals mit der Problematik der akustischen Beat Detection. Dabei stellten wir fest, dass die computerunterstützte Analyse von Musik im Allgemeinen und die Beat Detection im Speziellen aktuelle Thematiken und viel erforschte Gebiete sind. So konnten wir uns mit unserer Arbeit auf bestehende Studien verschiedener Forscher stützen.

Entstanden ist ein Matlab-Programm, welches die Onbeats verschiedener Musikstücke von Klassik bis Pop bestimmen kann. Dieses lief allerdings nicht in Echtzeit. Mit der Realisierung in Echtzeit wurde begonnen, die Zeit reichte aber nicht für die Fertigstellung.

Viele Teile des Algorithmus wurden nun vereinfacht, abgeändert oder durch andere Methoden ersetzt. Das soll die Semesterarbeit nicht abwerten. Während der Diplomarbeit tauchten neue Ansätze und Ideen auf, Erkenntnisse der Semesterarbeit wurden bestätigt oder relativiert und Probleme konnten dank der intensiveren Beschäftigung mit dem Thema gelöst werden. So liefen Teile der Arbeit, wie zum Beispiel die Bestimmung des Tempos, in eine völlig neue Richtung, während andere, das Ermitteln der Peaks etwa, ziemlich unverändert übernommen wurden.

Wegen der Erweiterung des Algorithmus und dem Hinzukommen des Video-Teils konnten die schon geschriebenen Echtzeit-Codes leider nicht weiter verwendet werden. Eine Änderung und Anpassung wäre aufwändiger gewesen als ein Neubeginn.

Die Dokumentation der Semesterarbeit befindet sich im Anhang, die Programme auf der beigelegten DVD.

1.3 Ziel der Diplomarbeit

Das Ziel dieser Diplomarbeit war dann die Programmierung einer optischen und akustischen Takterkennung in Echtzeit. Die von der Semesterarbeit her bestehende Software musste verbessert und als echtzeitfähiges System realisiert werden. Weiter galt es einen Algorithmus zur optischen Erkennung des musikalischen Pulses zu entwickeln und zu implementieren. Schliesslich sollten wir ein Demosystem entwickeln, welches beide Teile miteinander vereint und die Funktionsweisen des optischen und des akustischen Teils veranschaulicht.

2 Beschrieb der Algorithmen

Die Algorithmen wurden vor ihrer Implementation in C++ mit Hilfe von Matlab entwickelt. Die Matlab-Dateien sind auf der beigefügten DVD enthalten. Über deren Benützung gibt der als Kommentar enthaltene Hilfetext Auskunft. Geben Sie dazu `help FileName` in das Matlab Command Window ein.

2.1 Audio Algorithmus

2.1.1 Übersicht

Der Audioteil besteht, grob eingeteilt, aus den Bereichen Time Induction und Tempo Induction, welche parallel laufen. Es folgt ein Beschrieb der beiden Teile, je nach Art des Algorithmus mit Hilfe mathematischer Formeln oder als Pseudocode. Für einen groben Überblick empfiehlt sich das Studium von Abbildung 1.

2.1.2 Time Induction

Um den Takt eines Musikstücks erkennen zu können, müssen zuerst genügend aussagekräftige Daten zur Verfügung stehen. Diese Daten bestehen aus den Notenanfängen, den so genannten Onsets, der Musik. Das Ziel der Time Induction ist einerseits, möglichst viele aussagekräftige Onsets herauszufinden, andererseits sollen aus den unregelmässig auftretenden Onsets die regelmässig vorkommenden Beats bestimmt werden. Dies wird folgendermassen ausgeführt:

1. Enveloppe berechnen und Samplingrate reduzieren
2. Steigung der Enveloppe berechnen
3. Peaks in der Steigung suchen
4. Beats bestimmen

In den folgenden Unterkapiteln werden die einzelnen Arbeitschritte nun etwas detaillierter beschrieben. Die wichtigsten der verwendeten Funktionen sind auf Abbildung 2 schematisch dargestellt.

Enveloppe berechnen (RMSFloatEnv) Die Samplingrate von 44.1 kHz liefert recht viele Daten, die alle verarbeitet werden müssen. In Anbetracht dessen, dass das System echtzeitfähig sein soll, muss eine Datenreduktion vorgenommen werden. Wie man in verschiedener Literatur nachlesen kann ([1], [2]), reicht eigentlich die umhüllende Kurve, die Enveloppe, der Musik. Die Bewegungen, die durch die Onsets verursacht werden, sind auch in der Enveloppe ersichtlich. Wie schon in der Semesterarbeit, entschieden wir uns auch hier zur Berechnung der Enveloppe mit einem RMS-Wert-Filter.

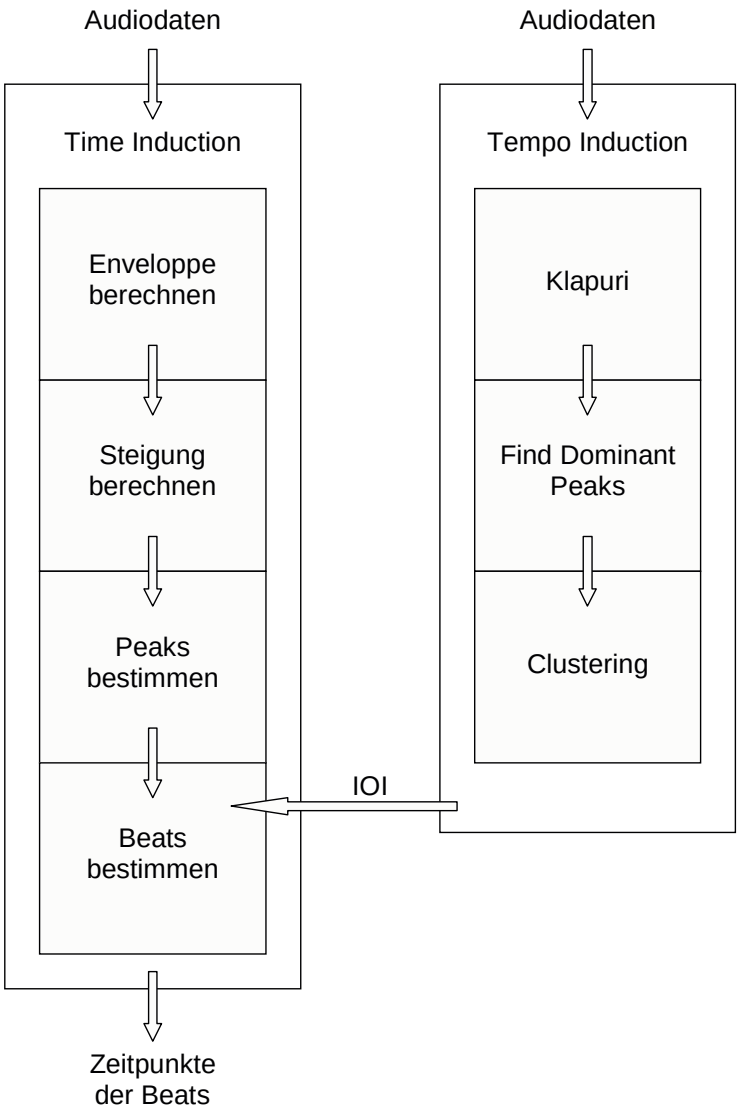


Abbildung 1: Grober Überblick über die Teile Time Induction und Tempo Induction.

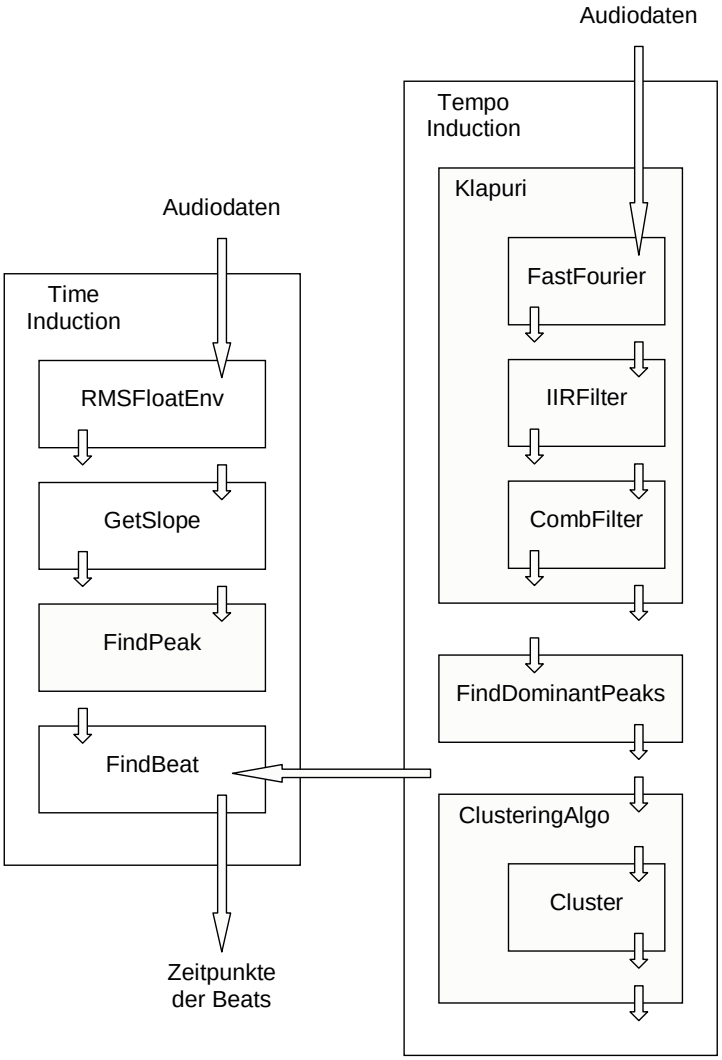


Abbildung 2: Grobüberblick über die wichtigsten der verwendeten Funktionen.

Das Musiksignal sei im Vektor $x[i]$ mit der Samplingrate R. Wir berechnen die Amplitude $A[k]$, mit k als einen Integer, der ein Vielfaches der Blockgrösse b ausdrückt. Die Samplingrate von $A[k]$ ist um den Faktor b/R kleiner. Die RMS-Amplitude wird über h Blöcke berechnet.

$$A[k] = \sqrt{\frac{1}{hb} \sum_{i=kb}^{(k+h)b-1} x[i]^2}$$

Steigung berechnen (GetSlope) Grundsätzlich kann man sagen, dass ein starker Impuls eine schnelle und starke Amplitudenänderung zur Folge hat. Das heisst, die Information für mögliche Onsets liegt in der Steigung. Schloss beschreibt in seiner Arbeit [1] eine Möglichkeit die Steigung zu berechnen. Er legt eine Regressionsgerade durch m Punkte der Amplitude $A[k]$ und berechnet die Steigung $S[k]$. Da für einen Steigungswert m Punkte verwendet werden, wird eine Glättung erreicht, welche die Aussagekraft der Steigungskurve verbessert.

$$S[k] = \frac{m \sum_{i=1}^{m} iA[k+i-m] - (\sum_{i=1}^{m} i)(\sum_{i=1}^{m} A[k+i-m])}{m \sum_{i=1}^{m} A[k+i-m]^2 - \sum_{i=1}^{m} i^2}$$

Peaks suchen (FindPeak) Nun müssen aus der Steigungskurve die Peaks bestimmt werden. Liegen Peaks höher als eine bestimmte Schwelle, wird ein Flag gesetzt. Der nächste Nulldurchgang, die Enveloppe hat dann einen Peak, wird als Onset betrachtet. Dies sieht als Code folgendermassen aus (Die Funktion wird für jeden Wert einzeln durchlaufen):

Input: Wert von GetSlope

if Counter $>$ 0 **then**
  Counter dekrementieren
end if
if Wert $>$ Schwelle **AND** Counter $=$ 0 **then**
  Flag $=$ true
else
  Peak $=$ 0
end if
if Wert $\leq$ 0 **AND** Flag $=$ true **then**
  Peak $=$ 1
  Flag $=$ false
  Counter auf Anfangswert
else
  Peak $=$ 0
end if

Output: Bei jedem Sample wird ausgegeben:
Peak $=$ 0 (kein Peak) oder
Peak $=$ 1 (Peak aufgetreten)

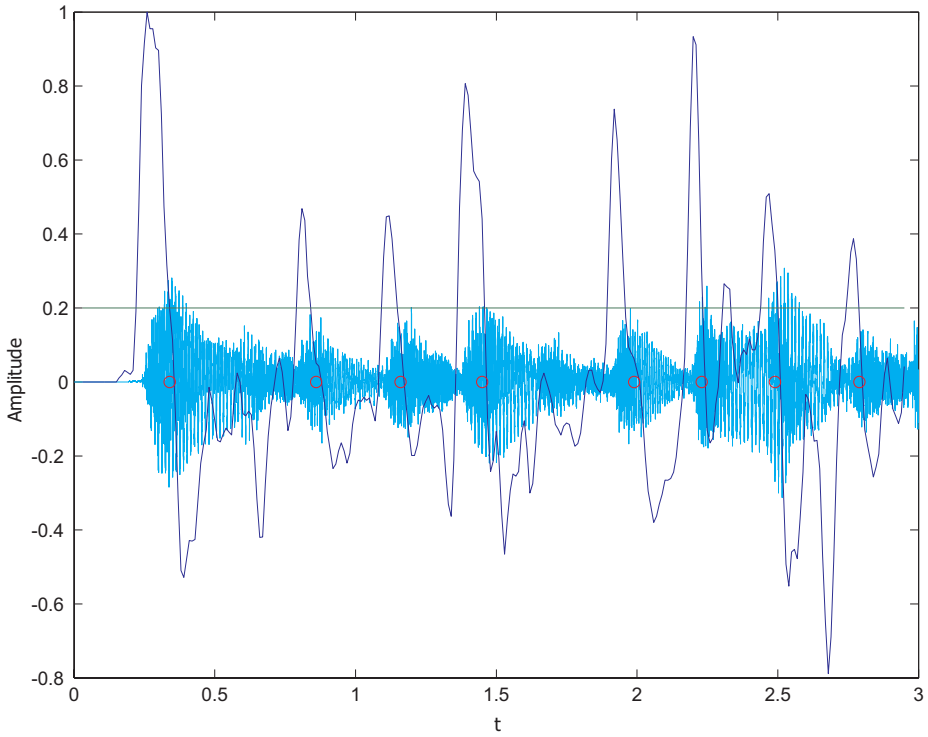


Abbildung 3: Die ersten drei Sekunden eines Musikstücks. Der Graph zeigt die rohen Audiodaten, die Steigungs-Kurve von GetSlope, die Schwelle bei y = 0.2 sowie die detektierten Peaks als Kringel bei y = 0.

Der Counter dient dazu, dass nicht zwei Peak sehr kurz nacheinander auftreten. Sein Anfangswert kann mit einem Parameter verändert werden. Auf Abbildung 3 sieht man ein von FindPeak bearbeitetes Stück einer Audiodatei.

Beats bestimmen (FindBeat) Die gefundenen Peaks könnten Beats darstellen. Nun müssen die Peaks aber noch auf Regelmässigkeiten überprüft, überzählige gelöscht und fehlende eingefügt werden. So entstehen dann die regelmässigen Beats.

Am Anfang eines Musikstücks wird ein kurzer Moment gewartet, damit die Time Induction genügend Zeit hat, das Tempo des Stücks, respektive das Inter Onset Interval (Zeit zwischen zwei Onsets), zu bestimmen. Nach dieser Zeit wird beim ersten auftretenden Peak geprüft, ob dieser verwandte Peaks in der Vergangenheit hat, d.h. ob vergangene Peaks ein Vielfaches des Inter Onset Intervalls zurück liegen. Ein als gut befundener Peak wird zum Beat und somit Ausgangspunkt der weiteren Berechnungen. Eine andere Startmöglichkeit bietet ein Mausklick, bei dem direkt ein Beat gesetzt wird.

Von einem gesetzten Beat aus wird mit Hilfe des Inter Onset Intervalls der nächste Zeitpunkt bestimmt, an welchem wieder ein Beat auftreten soll. Dieser Zeitpunkt liegt noch in der Zukunft, wird aber trotzdem schon gesetzt, da wir so die Durchlaufzeit des Systems kompensieren. Die Bestimmung des tatsächlichen Beats geschieht jedoch, wenn die Peaks schon in der Vergangenheit liegen. Dabei wird bei mehreren Peaks, welche innerhalb einer bestimmten Toleranz um den zuvor berechneten Zeitpunkt liegen, der nächste gewählt. Ist

jedoch zum gewünschten Zeitpunkt gar kein Peak vorhanden, setzt das System selbstständig einen.

Wenn nun aber zu viele Beats selber gesetzt werden, kann dies ein Indiz dafür sein, dass sich die gesetzten von den tatsächlichen Beats verschoben haben. In diesem Fall wird die Startmethode beim nächsten auftretenden Peak wieder durchgeführt und somit gelangt der Algorithmus zurück in den tatsächlichen Takt.

Nachfolgend der Pseudocode der Funktion FindBeat:

```
Input: Siehe Output von FindPeak

aktuelle Zeit bestimmen
if Peak = 1 then
  if Musikstück ist nicht mehr in Anfangsphase then
    if erster Beat ist zu setzten OR zu viele Beats selbst gesetzt then
      Startprozedur ausführen
      if Startprozedur war erfolgreich (Peak als gut befunden) then
        Peak als Beat übernehmen
      end if
    end if
  end if
  Peak für zukünftige Berechnungen speichern
end if
if erster Beat schon gesetzt then
  if Zeit > letzter Beat + 1.2 * IOI then
    for alle gespeicherten Peaks > letzter Beat do
      Suche Peak, der am nächsten liegt zu letztem Beat + IOI
    end for
    if kein genügend guter Peak gefunden then
      Beat selber setzten
    else if guter Peak gefunden then
      diesen Peak als Beat übernehmen
    end if
  end if
end if

Startprozedur:
for alle 20 letzten Peaks do
  if vergangener Peak + x * IOI ≈ aktueller Peak then
    (für x ∈ {1 2 3 4})
    ein guter Peak mehr
  end if
end for
if genügend gute Peaks then
  aktuellen Peak als Beat übernehmen
end if
```

Auf diese Weise kann unser Algorithmus auf kleinere Temposchwankungen reagieren und auch leichte Ritardandi [1] und Accelerandi [2] mitmachen.

2.1.3 Tempo Induction

Die Aufgabe der Tempo Induction ist das Herausfinden des richtigen Tempos eines Musikstücks. Genau genommen wird aber für die meisten Berechnungen das Inter Onset Interval (IOI), also die Zeit zwischen zwei Beats, benötigt. Die Tempo Induction teilt sich in folgende Funktionen ein:

1. Klapuri (nach [4])
2. Find Dominant Peaks
3. Clustering

Diese Teile werden in den folgenden Abschnitten näher beschrieben.

Klapuri Das Kernstück der Tempo Induction bildet ein Algorithmus, welcher nach einem Paper [4] des Finnen Anssi Klapuri entwickelt wurde. Nach einer 50% überlappenden Fourier Transformation wird jedes Frame in 36 Bänder gleicher Bandbreite aufgeteilt. Die Leistungen der einzelnen Bänder werden ausgerechnet und als $x_b[k]$ gespeichert, wo k den Frameindex darstellt und b den Index des jeweiligen Bandes. Die Normierung

$$y_b[k] = \frac{ln(1 + \gamma x_b[k])}{ln(1 + \gamma)}$$

bewirkt, dass die Werte zwischen Null und Eins liegen. In unserem Fall gilt $\gamma = 100$.

Um eine bessere zeitliche Auflösung zu erreichen erfolgt ein Upsampling der Werte $y_b[k]$ um Faktor Zwei (durch Einfügen von Nullen), womit das Signal $y'_b[n]$ mit einer Samplingrate von $f_s = 172$ Hz erzeugt wird. Ein Butterworth-Tiefpassfilter sechster Ordung mit $f_{TP} = 10$ Hz glättet den in Abbildung 4 ersichtlichen Graphen zur Kurve in Abbildung 5. Das entstandene, geglättete Signal nennen wir $z_b[n]$.

Die Differenzierung von $z_b[n]$ geschieht folgendermassen:

$$z'_b[n] = \text{HWR}\{z_b[n] - z_b[n-1]\}$$

HWR{ } steht für die Einweg-Gleichrichtung (Half Wave Rectification), welche nötig wird um die Differenzierung brauchbar zu machen. Siehe dazu die Abbildungen 6 und 7.

Als weitere Vorbereitungen für das anschliessende Kamm-Filter wird zuerst ein gewichteter Mittelwert von $z_b[n]$ und seinem Differential $z'_b[n]$ erstellt (siehe Abbildung 8):

[1] ital. für langsamer werdend
[2] ital. für schneller werdend

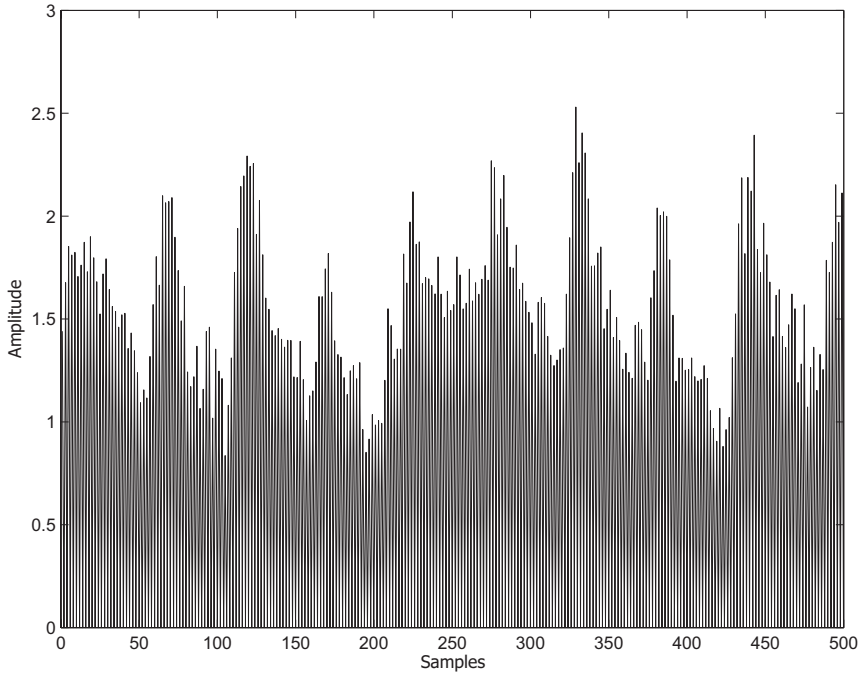


Abbildung 4: Die Werte nach dem Upsampling auf 172 Hz über eine Dauer von knapp drei Sekunden.

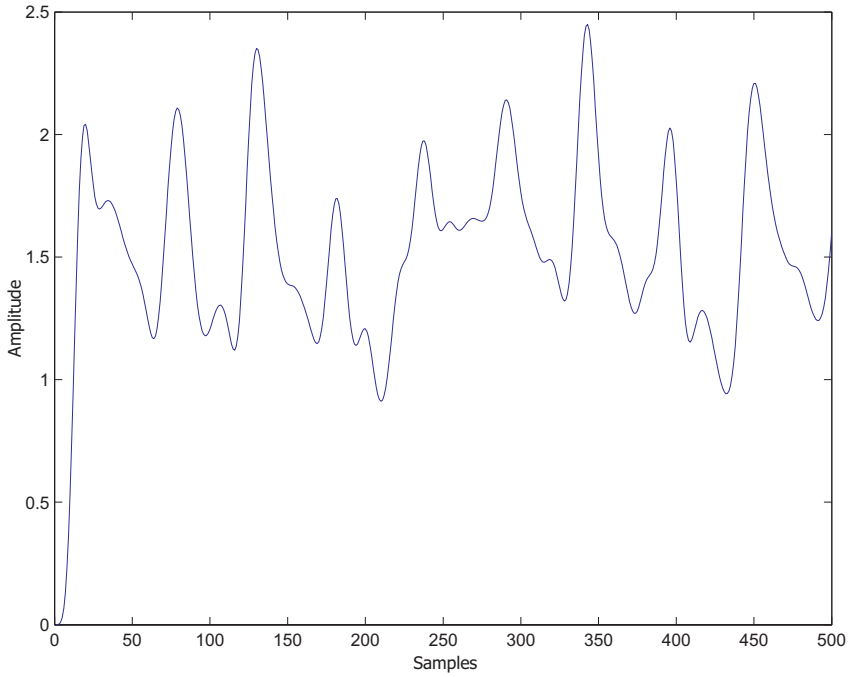


Abbildung 5: Nach der Glättung mit einem Tiefpass-Filter entsteht wieder eine schöne Kurve.

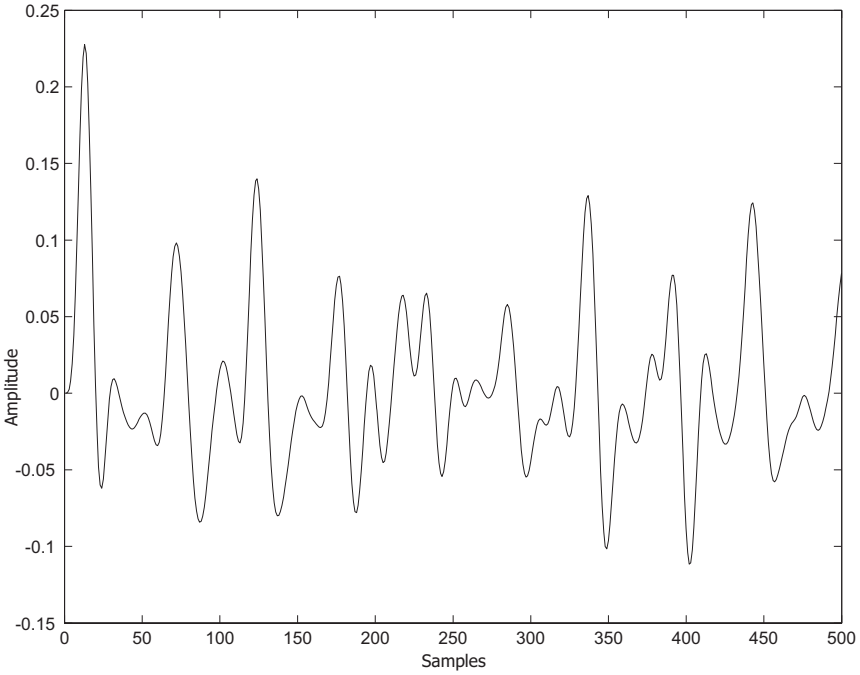


Abbildung 6: Deutlichere und schmalere Peaks entstehen durch die Berechnung der Steigungen.

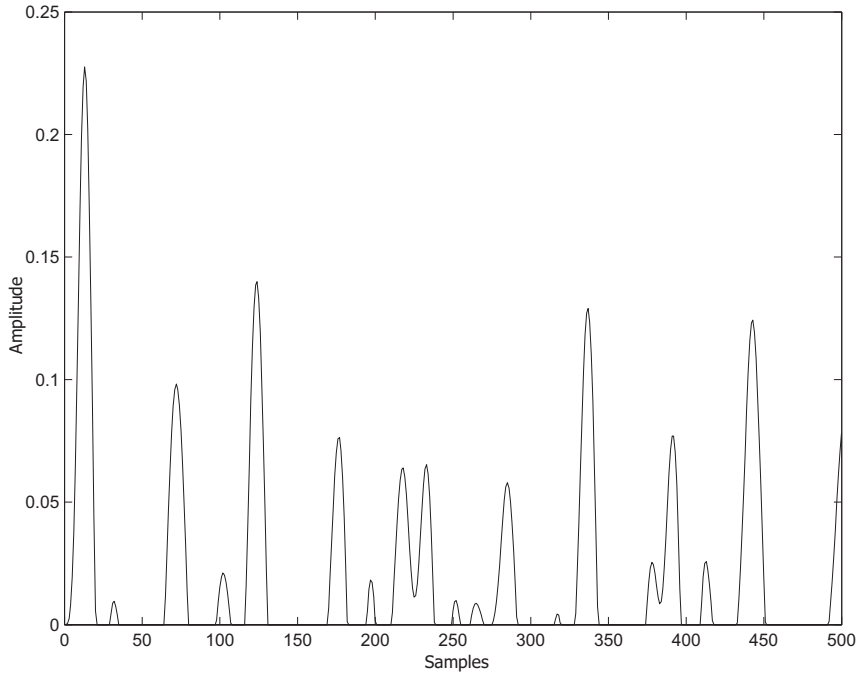


Abbildung 7: Durch die Einweg-Gleichrichtung entstehen sehr deutliche und markante Peaks.

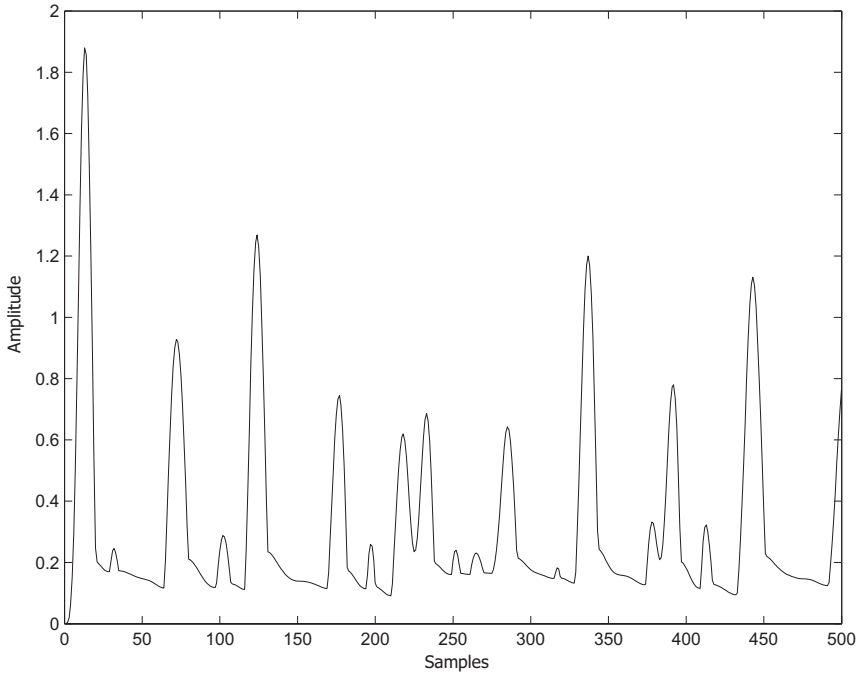


Abbildung 8: Der gewichtete Mittelwert der Signale $z_b[n]$ und $z'_b[n]$.

$$u_b[n] = (1-w)z_b[n] + w\beta z'_b[n]$$

wobei $w = 0.9$ und $\beta = \frac{f_s}{2f_{TP}}$. Der Faktor β kompensiert die kleine Amplitude des Differentials. Danach wird die Bänderanzahl von 36 auf vier reduziert (Abbildung 9):

$$v_c[n] = \sum_{b=(c-1)9+1}^{9c} u_b[n]$$

Nun folgt das Kernstück des Klapuri-Algorithmus: das Kamm-Filter. Es entstehen nun zu jedem Zeitpunkt Werte über die letzten paar Sekunden, welche das vorherrschende IOI sehr eindrücklich darstellen, wie wir in Abbildung 11 sehr schön sehen können. Das Filter reicht vier Sekunden in die Vergangenheit und ergibt somit einen genügend guten Überblick über das Tempo. Abbildung 10 zeigt den Output des Kamm-Filters über einige Sekunden.

Das eigentliche Kamm-Filter ergibt sich aus

$$r_c[\tau, n] = \alpha_\tau r_c(\tau, n-\tau) + (1-\alpha_\tau)v_c[n],$$

wobei $\alpha_\tau = 0.5^{\frac{\tau}{T_0}}$ mit $T_0 = 3f_s$. Gleich nach dem Kamm-Filter werden die Filter-Leistungen

$$\hat{r}_c[\tau, n] = \frac{1}{\tau} \sum_{i=n-\tau+1}^{n} (r_c[\tau, i])^2$$

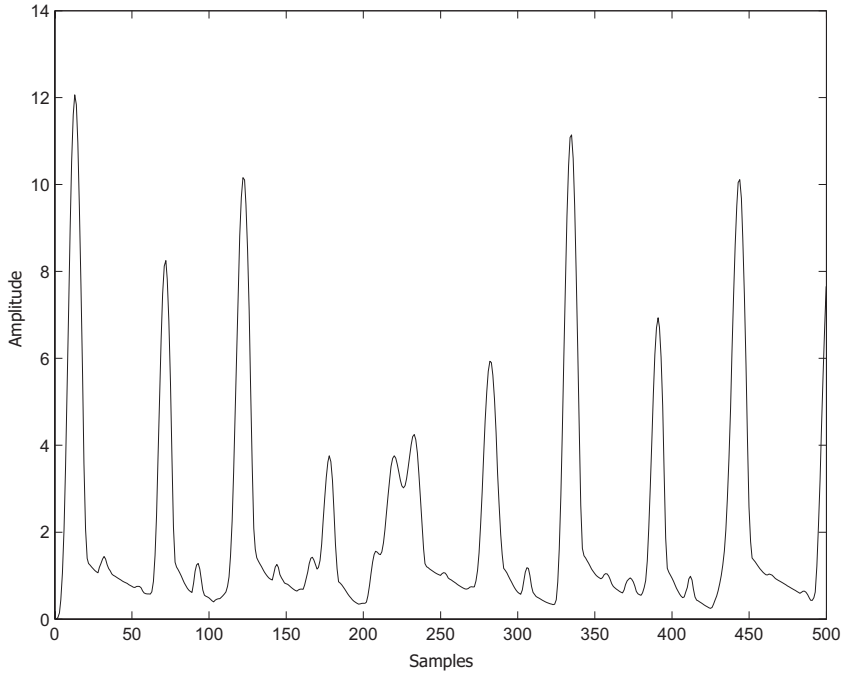


Abbildung 9: Signal nach der Reduktion der 36 Bänder auf vier.

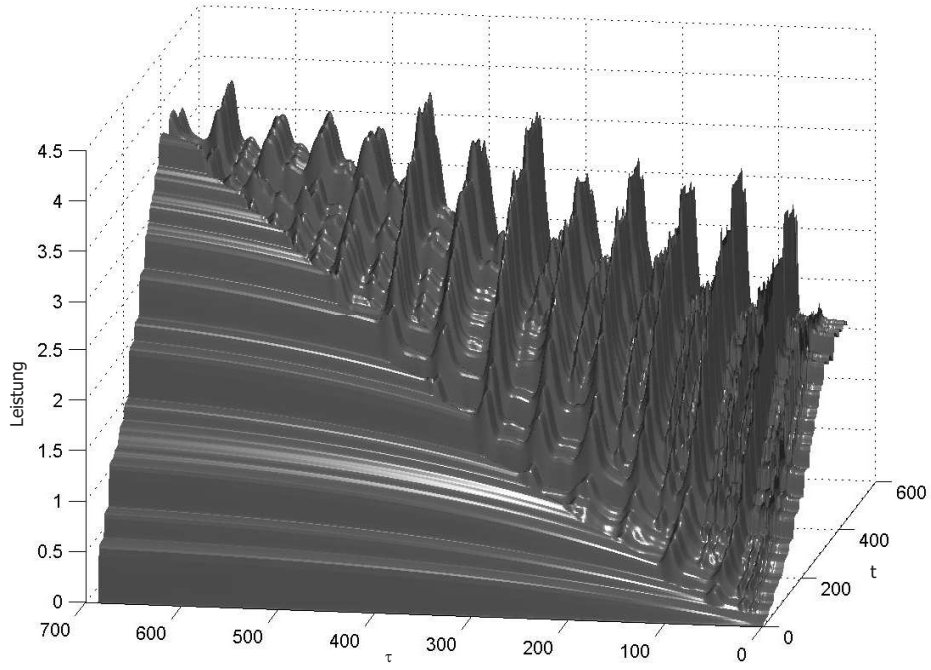


Abbildung 10: Output des Kamm-Filters über die ersten Sekunden eines Musikstücks.

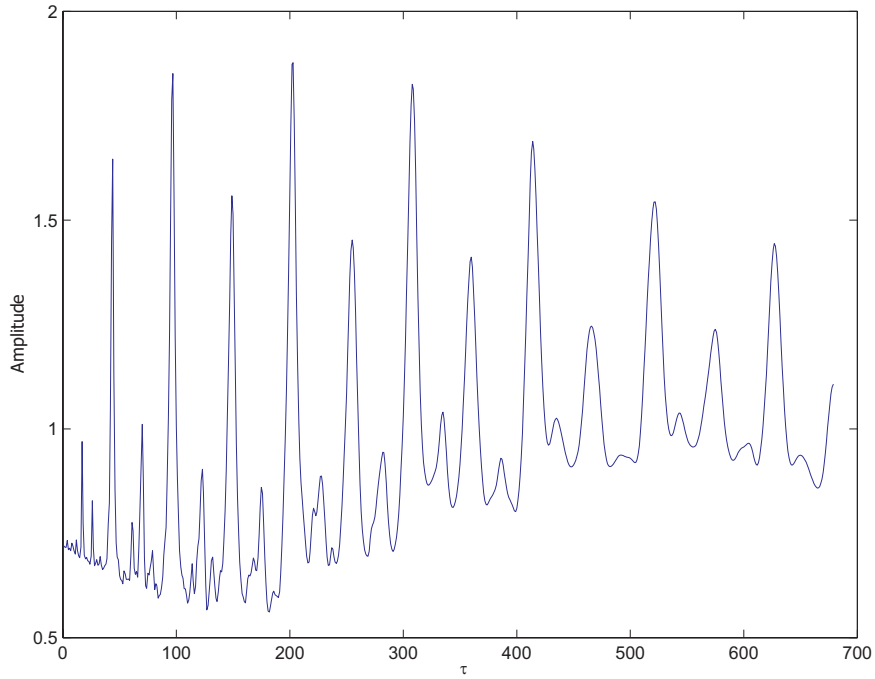


Abbildung 11: Der Graph zeigt das Ergebnis nach dem Kamm-Filter und der Berechnung der Filter-Leistungen zu einem bestimmten Zeitpunkt. Die Abstände zwischen den Peaks zeigen sehr schön die Grösse des IOI an. Die Breite des Graphen entspricht vier Sekunden.

berechnet. Zum Schluss werden die verbliebenen vier Frequenzbänder in einem Band vereinigt:

$$s[\tau, n] = \sum_{c=1}^{4} \hat{r}_c[\tau, n]$$

Aus dieser Kurve berechnet die Funktion FindDominantPeaks, der nächste Schritt in der Tempo Induction, mögliche IOIs.

Find Dominant Peaks Die Funktion FindDominantPeaks wurde von Franz Bachmann als Matlab-Funktion geschrieben und später von Peter Aeschimann in C++ übersetzt [7]. Sie bestimmt, wie der Name sagt, alle dominanten Peaks eines Graphen und eignet sich daher bestens für unsere Zwecke. In Abbildung 12 ist sehr deutlich sichtbar, wie die Funktion ausschliesslich dominante Peaks detektiert, nicht aber die vor allem in der linken Hälfte des Graphen auch sehr deutlichen lokalen Maxima, welche mitten zwischen zwei dominanten Peaks liegen.

Clustering Die Abstände zwischen den von FindDominantPeaks vorgeschlagenen Peaks durchlaufen nun ein Clustering, welches den vorherrschenden Abstand ermittelt (Ein Clustering nach Simon Dixon [3] wurde bereits in der Semesterarbeits-Version verwendet, dieses

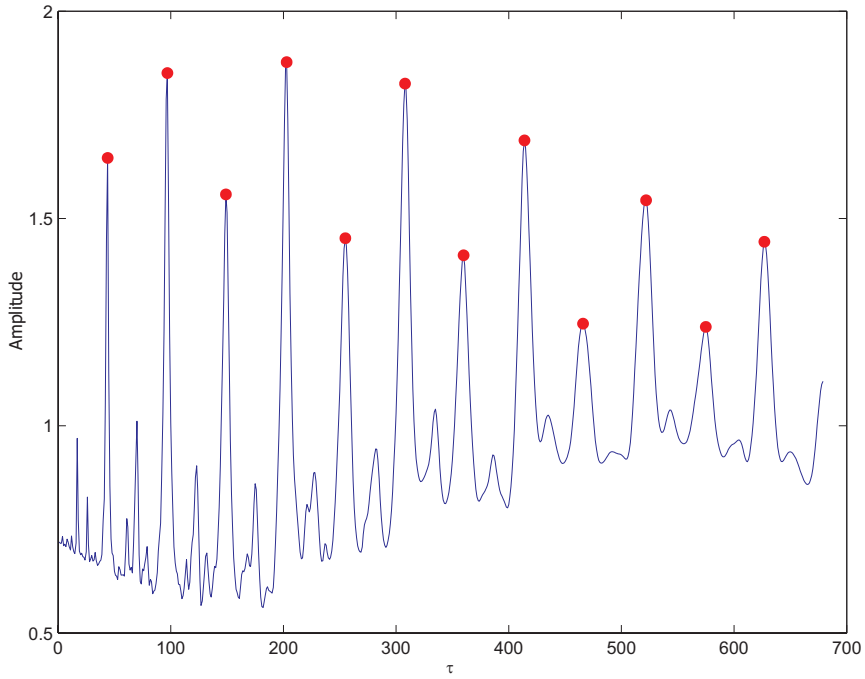


Abbildung 12: Ein Graph, wie er der Funktion FindDominantPeaks als Input dient. Die Tupfer stellen die detektierten Peaks dar.

wurde nun aber vereinfacht). Jeder Abstand wird, falls vorhanden, in einen passenden Cluster eingefügt. Ein Cluster gilt als passend, wenn die Differenz von seinem Mittelwert zum zu prüfenden Abstand kleiner als eine eingestellte Toleranz ist. Der Algorithmus erstellt einen neuen Cluster, falls kein passender vorhanden ist. So kann dann von demjenigen Cluster, welcher über die meisten Einträge verfügt, der Mittelwert über alle Einträge berechnet werden, um das IOI zu bestimmen:

for alle Abstände **do**
  if Abstand passt in bestehenden Cluster **then**
    Abstand in Cluster einfügen
  else
    Neuen Cluster erstellen
  end if
end for

for alle Cluster **do**
  Suche grössten Cluster
end for
Nimm Mittelwert des grössten Clusters als IOI

Damit Ausrutscher nicht unerwünscht berücksichtigt werden, speichern wir die letzten Werte und nehmen daraus den Median.

2.2 Video Algorithmus

2.2.1 Übersicht

Die Auswertung eines Videosignals in Echtzeit erfordert sehr leistungsfähige Computer und hoch optimierte Algorithmen. Mit einem normalen PC ist es nicht möglich, komplizierte Vorgänge in jedem Frame durchzuführen. Man hat jedoch die Möglichkeit den Rechenaufwand massiv zu reduzieren, indem man nur eine kleine Bildauflösung wählt. Jedes Halbieren der Auflösung reduziert die Prozessorbelastung für einen Algorithmus um einen Viertel. Ausserdem hilft ein Reduzieren der Frame-Rate bei einem überforderten Computer. Dies geht dann aber sofort auf die Kosten der flüssigen Darstellung, auf die das Empfinden von gut und schlecht sehr empfindlich reagiert.

Für die Erkennung der Dirigierbewegungen eines Dirigenten wurden einfache Operationen gewählt. Die Hand wird nicht mittels einer Mustererkennung im Bild erkannt, der Algorithmus sucht nur die stärkste Bewegung. Die Bildgrösse wurde auf 160 x 120 Pixel festgelegt, die Frame-Rate beträgt 25 Bilder pro Sekunde.

Da nur die Bewegungen im Videosignal erkannt werden, sind folgende Einschränkungen nötig um ein aussagekräftiges Signal zu erhalten:

- Der Dirigent darf nur eine Hand für die Dirigierbewegungen verwenden.
- Der Dirigent darf seinen Körper nicht zu stark bewegen.
- Auch im Hintergrund muss alles ruhig sein.

Der Algorithmus erkennt die Bewegungen indem er ein Differenzbild vom aktuellen und dem voherigen Frame erstellt. Die bewegte Hand hat die Eigenschaft, dass sie eine grosse Bewegung ausführt, klein und relativ kontrastreich ist. Ähnliche Eigenschaften besitzt jedoch auch der Kopf. Darum ist darauf zu achten, dass nicht zu grosse Kopfbewegungen ausgeführt werden.

Da die Hand nicht mittels Mustererkennung im Bild erkannt wird, ist die ermittelte Koordinate, die die Hand repräsentiert, mit einer relativ grossen Unsicherheit behaftet. Mit Hilfe eines Kalman-Filters werden die Bewegungen des Filters geglättet. Das Kalman-Filter versucht eine eingeschlagene Bewegung zu halten. Es verwendet ein System, das den nächsten Wert vorhersagt und anschliessend diesen Wert mittels der aus dem Bild ermittelten Koordinate korrigiert.

Nun besitzt man eine Koordinate, die die aktuelle Position der Hand repräsentiert. Der nächste Schritt ist das Finden des Takts aus den Bewegungen. Wenn man die Bewegungen des Dirigenten betrachtet, stellt man fest, dass er auf den Schlag die Hand nach unten bewegt und zwischen den Schlägen wieder eine Bewegung nach oben macht. So muss man nur die Y-Auslenkung der Bewegung auf dem Bild betrachten um den Takt zu erkennen. Der Beat tritt beim unteren Umkehrpunkt der Hand auf.

Um den Umkehrpunkt, also ein Maximum, zuverlässig bestimmen zu können, muss man die Umgebung vor und nach dem Maximum kennen. Da wir aber die Bearbeitung in Echtzeit machen, müssten wir eine sehr grosse Verzögerung in Kauf nehmen um den Beat-Zeitpunkt

zu finden. Diesen Kompromiss können wir aber nicht machen, da die Latenz des Algorithmus möglichst Null sein soll. Der einzige Ausweg aus diesem Dilemma ist eine Vorhersage des nächsten Schlages.

Für die Voraussage bietet sich wieder das Kalman-Filter an. Man sagt den nächsten Schlag voraus, der unter Bezug des aktuellen Tempos gesetzt wird und korrigiert das System wieder, wenn der effektive Schlag aus dem Videosignal erkannt wurde. Da die Dirigierbewegung regelmässig ist, stimmt die Voraussage sehr gut.

Ein weiterer Vorteil bei der Verwendung des Kalman-Filters ist, dass das System adaptiv ist und sich immer an eine neue Situation anpassen kann. Natürlich ist dazu eine gewisse Zeit erforderlich, das System reagiert jedoch sehr schnell. Es ermittelt auch selbständig das aktuelle Tempo, da dies eine Eigenschaft des Sytems ist.

Die so erkannten Beat-Zeitpunkte sind nun synchron zu den Bewegungen des Dirigenten. Der Algorithmus eliminiert die Verzögerung, die durch die Berechnung entsteht, indem er das nächste Ereignis voraussagt. Im folgenden noch einmal kurz die Eigenschaften des Sytems:

- Die Berechnungen sind nicht sehr rechenaufwändig, da auf komplizierte Algorithmen verzichtet und eine kleine Bildgrösse gewählt wurde.
- Unter Beachtung gewisser Randbedingungen kann der Bewegung der Hand sehr gut gefolgt werden.
- Sind die Bewegungen genügend ausladend und folgen der Bedingung, dass unten der Schlag ist und die Hand dazwischen wieder gehoben wird, ist die Erkennung des Beats kein Problem.
- Durch Verwendung eines Kalman-Filters ist das System adaptiv und passt sich jeder Situation an.
- Durch Voraussage des Schlages wird die Verarbeitungszeit zum Erkennen eines Schlages eliminiert.

2.2.2 Die verwendeten Algorithmen

Nachdem wir einen Überblick über den Ablauf der Takterkennung eines Dirigenten gewonnen haben, gehen wir in diesem Kapitel auf die verwendeten Algorithmen ein. Als erstes folgt aber ein schematischer Systemüberlick (Abbildung 13). Der Algorithmus kann in zwei Teile gegliedert werden: In das Tracking der Hand und in das Tracking des Beats. Der Hand Tracking Algorithmus hat als Input das Videosignal und liefert am Ausgang die Koordinate der Hand. Diese übernimmt der Beat Tracking Algorithmus, der den nächsten Beat ermittelt.

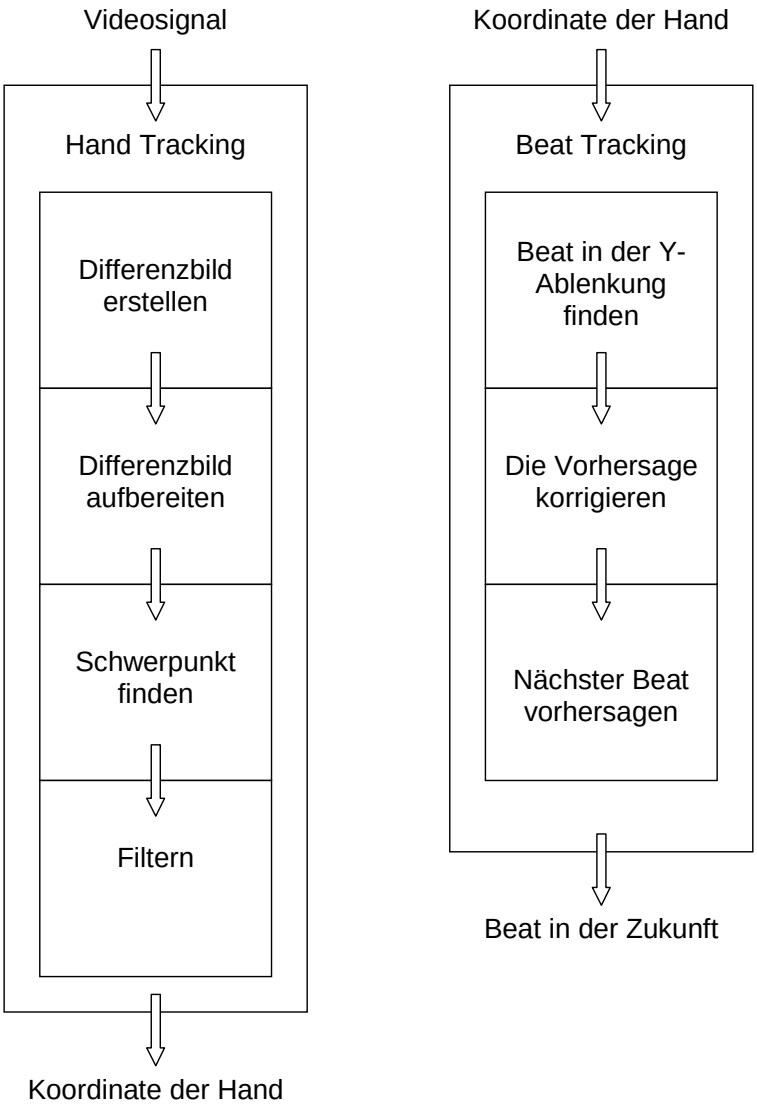


Abbildung 13: Überblick über den Algorithmus

Abbildung 14: Zwei aufeinander folgende Frames

Abbildung 15: Links ist das Differenzbild, rechts ist das selbe Bild mit erhöhtem Kontrast

Hand Tracking

Der Algorithmus, der die Bewegung der Hand detektiert, erhält am Eingang das Videosignal. Abbildung 14 zeigt zwei aufeinander folgende Graustufen-Bilder, die nun in den folgenden Ausführungen verwendet werden.

Differenzbild erstellen Der erste Schritt ist das Bilden des Differenzbildes (Abbildung 15). Da der Wertebereich nur positive Zahlen umfasst, wird noch der Absolutbetrag genommen. Nun wird der Kontrast des Bildes erhöht. Dies dürfen wir aber nur machen, wenn das hellste Pixel einen Schwellwert übersteigt. Man läuft sonst Gefahr ein Bild, das keine Bewegung enthält, kontrastreich zu machen und so praktisch nur das Rauschen zu verstärken. Die Schwelle zeigt bei ca. 15% des Maximalwertes die gewünschte Wirkung.

Differenzbild aufbereiten Wenn man Bild 15 betrachtet, erkennt man sehr viel Änderung im Bereich der Hand und noch ein wenig im Bereich des Armes. Ziel ist es, möglichst nur die Hand als weissen Fleck zu haben. Wenn man nun für jedes Pixel auch seine Umgebung betrachtet, hat man schliesslich nur die Position der Hand (Abbildung 16). Diese Operation wird mit einem Mittelwertfilter erreicht. Eine Filter-Kernelgrösse von 11 x 11 Pixel hat die besten Resultate gezeigt. Zur Weiterverarbeitung wird der Kontrast wieder erhöht. Nun ist

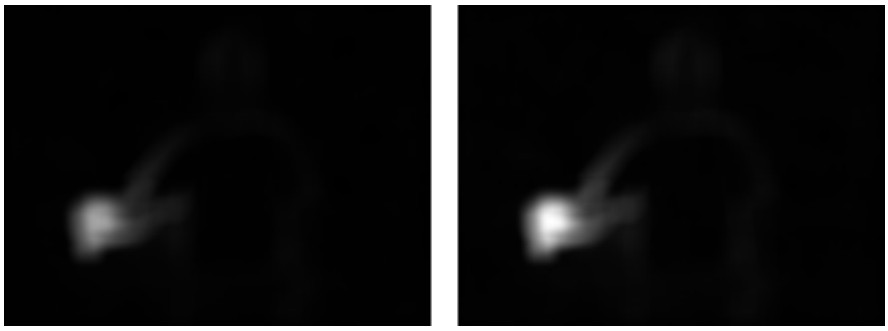


Abbildung 16: Links ist das gefilterte Differenzbild, rechts ist das selbe Bild mit erhöhtem Kontrast

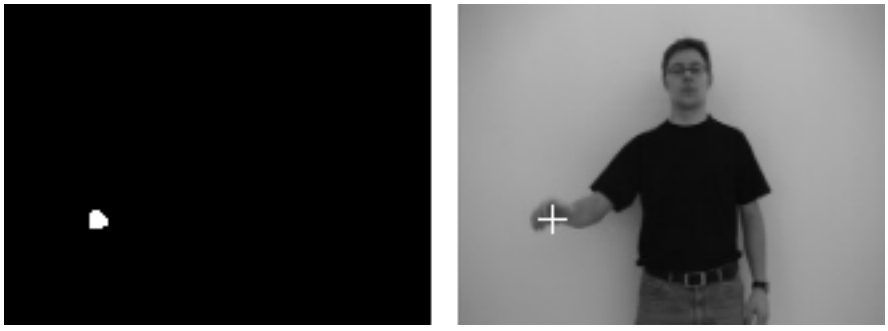


Abbildung 17: Links ist das binarisierte Bild, rechts ist dessen Schwerpunkt im Originalbild eingezeichnet

fast nur noch die Hand zu erkennen.

Schwerpunkt finden Um nun eine Koordinate für die Position der Hand zu erhalten binarisieren wir das Bild und berechnen den Schwerpunkt (Abbildung 17). Die Schwelle für das Binarisieren liegt etwa bei 90% des Maximalwertes.

Der Schwerpunkt wird folgendermassen berechnet:

xsum $= 0$
ysum $= 0$
pixelcount $= 0$
for alle Pixel, die den Wert 1 haben **do**
  xsum $=$ x-Positionen des Pixels
  ysum $=$ y-Positionen des Pixels
  pixelcount $++$
end for
$x =$ xsum$/$pixelcount
$y =$ ysum$/$pixelcount

Filtern Da nicht immer die selbe Position der Hand detektiert werden kann, ist es nötig das Signal zu filtern. Dazu wurde ein Kalman-Filter verwendet. Auf die Theorie des Filters kann hier nicht genauer eingegangen werden. Wir verweisen auf die entsprechende Literatur ([5], [6]).

Hier nur ein kurzer Überblick: Die Theorie des Filters baut auf die Wahrscheinlichkeitsrechnung auf. Man geht davon aus, dass das lineare System und die Messwerte mit einer gewissen Unsicherheit belastet sind. Die Streuungen dieser Werte sind jeweils normalverteilt. Nun lässt sich mit Hilfe von Parametern die Glaubwürdigkeit des Systems und der Messwerte einstellen. Je nachdem glaubt das Filter mehr sich selber, also dem System, oder übernimmt eher den Messwert. Grundsätzlich ist es immer eine Mischrechnung dieser beiden Werte. Die Schwierigkeit besteht darin die Werte für die Glaubwürdigkeit so zu bestimmen, dass sich das Filter wie gewünscht verhält.

Das Kalman-Filter basiert auf dem State-Modell. Das System hat folgende Form:

$$x_k = Ax_{k-1}$$

Das Filter berechnet den neuen State in zwei Schritten. Zuerst sagt er den nächsten Wert x_k^- voraus

$$\begin{aligned} \hat{x}_k^- &= A\hat{x}_{k-1} + Bu_{k-1} \\ P_k^- &= AP_{k-1}A^T + Q \end{aligned}$$

Dann korrigiert es den vorausgesagten State x_k^- mit dem Messwert z.

$$\begin{aligned} K_k &= P_k^- H^T (HP_k^- H^T + R)^{-1} \\ \hat{x}_k &= \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \\ P_k &= (I - K_k H)P_k^- \end{aligned}$$

Der neue State x_k setzt sich nun aus dem vorherigen State x_{k-1}, der mit der Systemunsicherheit Q behaftet ist, und dem Messwert z mit der Messunsicherheit R zusammen.

Für die Glättung der Hand-Koordinaten wurde folgendes System verwendet:

$$\begin{bmatrix} s_{x_k} \\ s_{y_k} \\ v_{x_k} \\ v_{y_k} \\ a_{x_k} \\ a_{y_k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & 1 & 0 & \Delta t & 0 & \frac{\Delta t^2}{2} \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{x_{k-1}} \\ s_{y_{k-1}} \\ v_{x_{k-1}} \\ v_{y_{k-1}} \\ a_{x_{k-1}} \\ a_{y_{k-1}} \end{bmatrix}$$

Die Unsicherheitsmatrizen wurden durch Suchen eines Optimums mit Matlab bestimmt. Das Resultat ist befriedigend und genügt für die Systembedingungen. Abbildung 18 zeigt die Verbesserung der Bewegung der Y-Koordinate.

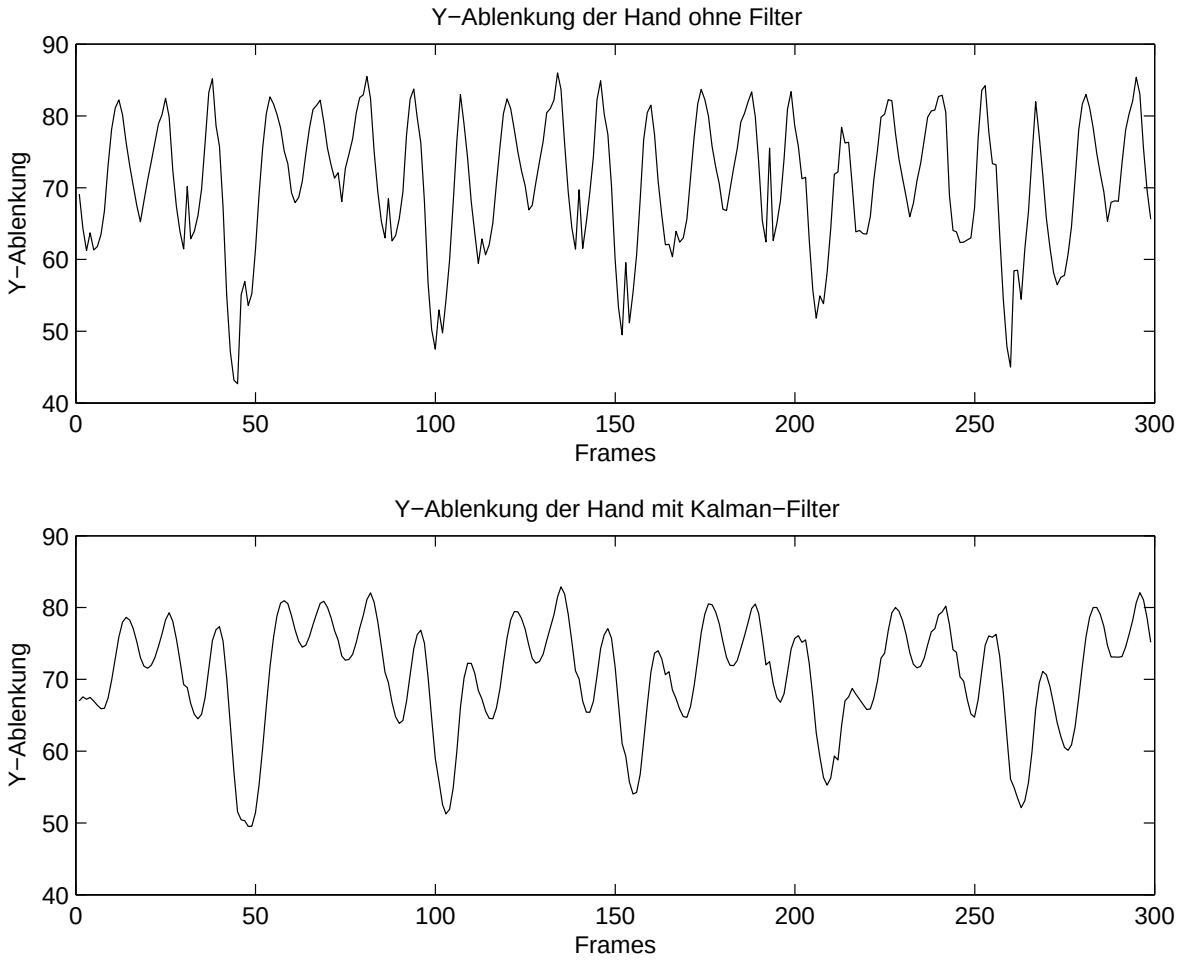


Abbildung 18: Die Y-Koordinate der Hand vor und nach dem Kalman-Filter

Beat Tracking

Die zentralen Algorithmen für das Beat Tracking sind FindDominantPeaks [7] von Franz Bachmann und das Kalman-Filter. Wie schon erwähnt wird der nächste Schlag vorhergesagt, damit der Peak zuverlässig gefunden werden kann. Erst wenn der vorhergesagte Peak auch eingetroffen ist, wird der Wert des Kalman-Filters korrigiert und sofort der nächste vorherbestimmt.

FindDominantPeaks Ein dominanter Peak muss vier Kriterien erfüllen:

- Die Distanz zu dem nächst höheren Peak auf der linken Seite muss mindestens *dthr* betragen.
- Die Distanz zu dem nächst höheren Peak auf der rechten Seite muss mindestens *dthr* betragen.
- Der Quotient der Differenz vom Peak und dem tiefsten Tal bis zum nächst höheren Peak auf der linken Seite und dem Peak muss grösser als *sthr* sein.
- Der Quotient der Differenz vom Peak und dem tiefsten Tal bis zum nächst höheren Peak auf der rechten Seite und dem Peak muss grösser als *sthr* sein.

Da wir aber die Daten in Real-Time verarbeiten müssen, können wir nicht warten bis der nächst höhere Peak auf der rechten Seite auftritt. Die Bedingung mit der Distanz kann aber als erfüllt betrachtet werden, wenn *dthr* überschritten wird. Auf die Tiefe des Tals auf der rechten Seite kann hingegen nicht geachtet werden. In den meisten Fällen genügen aber diese Bedingungen. Durch das adaptive Verhalten kann ein falsch gesetzter Peak auch wieder korrigiert werden.

Beat setzen Dieses Kalman-Filter verwendet folgendes System (IOI bedeutet Inter Onset Interval; es gilt $IOI = 60/\text{BPM}$):

$$\begin{bmatrix} t_k \\ IOI_k \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} t_{k-1} \\ IOI_{k-1} \end{bmatrix}$$

Auch hier wurde für die Unsicherheitsmatrix Matlab zum Suchen des Optimums verwendet. Das Filter ist so eingestellt, dass es dem Takt möglichst rasch folgt.

3 Softwaredesign

3.1 Anforderungen an die Software

Damit die entwickelten Algorithmen auch eingesetzt werden können, muss eine Applikation erstellt werden, die folgende Aufgaben hat:

- Die Videokamera ansprechen
- Die Soundkarte ansprechen
- Auf dem Computer gespeicherte Media-Dateien verarbeiten
- Einen Mechanismus zur Verfügung stellen um mit Streams zu arbeiten
- Eine benutzerfreundliche Bedienung bereit stellen
- Die Darstellung auf dem Bildschirm übernehmen

Diese Aufgaben wären sehr komplex und schwierig zu realisieren, wenn nicht Softwarepakete zur Verfügung stehen würden, welche die Arbeit abnehmen. Sie stellen relativ einfache Schnittstellen zur Verfügung, über die sich die Funktionen steuern lassen.

Im Folgenden stellen wir nun die verwendeten Technologien vor.

3.2 Verwendete Technologien

C++ Die objektorientierte Programmiersprache C++ wird der Anforderung gerecht Modularisierung, Abstrahierung, Bedienungsfreundlichkeit und hardwarenahe Programmierung zu vereinen. Durch das Verwenden von Objekten und Vererbungsstrukturen lassen sich komplizierte Vorgänge kapseln und einfach weiterverwenden. Da der Compiler Maschinencode erstellt, also nicht interpretiert wird, ist das Programm ideal für die Realtime-Anwendungen.

DirectShow Eine grosse Anforderung an die Applikation ist das Verwalten und Ansprechen der Hardware. Dazu kommt das Bereitstellen und Steuern von Daten-Streams. All diese Dinge selber zu implementieren wäre ein Ding der Unmöglichkeit, wenn man nur schon an die Vielzahl der Hardwarekomponenten denkt.

Doch dies ist auch nicht nötig, denn Microsoft stellt ein Softwarepacket zur Verfügung, das eine einfache Programmierung ermöglicht. DirectX abstrahiert den Zugriff auf alle Hardwarekomponenten und nutzt deren Fähigkeiten und Funktionen aus. Dadurch erreicht man zwei Dinge: Unabhängigkeit von einer bestimmten Hardware und optimierte Ausführung der Applikation.

DirectX nutzt das Component Object Model (COM). Ein COM-Objekt stellt eine bestimmte Funktion zur Verfügung. Mit Hilfe von Interfaces kann diese Funktionalität genutzt werden.

Das COM-Objekt kann eine Vielzahl von Schnittstellen implementieren, die alle einen spezifischen Teil der Funktion kontrollieren. Der grosse Vorteil an COM ist, dass eine Softwarekomponente einfach wiederverwendet werden kann.

DirectShow ist eine Komponente von DirectX. DirectShow beinhaltet alle Funktionen zum Arbeiten mit Multimedia. Das heisst:

- Soundkarten und Videokameras, die einen Windows-Treiber besitzen, können über eine einheitliche Schnittstelle benutzt werden.
- Das Rendern von Video-Streams und das Ausgeben von Musik über die Soundkarte ist inbegriffen.
- Media-Streams können auch von einem Speichermedium gelesen und geschrieben werden.
- Es steht eine Vielzahl von Effekten, Coder und Encoder zur Verfügung, mit denen sich der Media-Stream bearbeiten lässt.
- Die Verarbeitung der Daten ist in Echtzeit möglich.
- Die Hardwarebeschleunigung wird ausgenutzt.
- Der Media-Stream kann einfach gesteuert werden.
- Das Erstellen von eigenen Funktionen ist sehr einfach, da ein vorbereitetes Gerüst zur Verfügung steht.
- Über die Schnittstelle der Property Page können Parameter eines DirectShow Filters verändert werden.

DirectShow kapselt jede Funktionseinheit in so genannte Filter. Ein Filter ist ein COM-Objekt, das Schnittstellen zur Verfügung stellt. Mit einem Manager-Objekt, dem Filter-Graph, können die einzelnen Filter verbunden werden und so den Media-Stream durch die Applikation leiten. Über den Filter-Graph lässt sich der Stream steuern.

Dies sei an einem Beispiel illustriert. Wir wollen eine komprimierte Video-Datei ausgeben. In Bild 19 sieht man die einzelnen Filterblöcke und die Verbindungen, die den Stream-Fluss darstellen. Das Video wird decodiert und angezeigt. Durch den Graph-Manager kann das Video gestartet und gestoppt werden. Die graphische Visualisierung ist durch das Programm GraphEdit (graphedt.exe) möglich. Es ist im Software Developement Kit (SDK) von DirectX enthalten.

Wenn man ein eigenes Filter implementiert, muss es dem Betriebsystem bekannt gemacht werden. Jedes COM-Objekt ist mit einem Globally Unique Identifier (GUID), registriert. Diese Registrierung muss man von Hand vornehmen. Mit folgendem Befehl wird die Registrierung durchgeführt:

```
regsvr32.exe MyFilter.ax
```

Um die Registrierung wieder zu löschen, muss die Option `/u` verwendet werden:

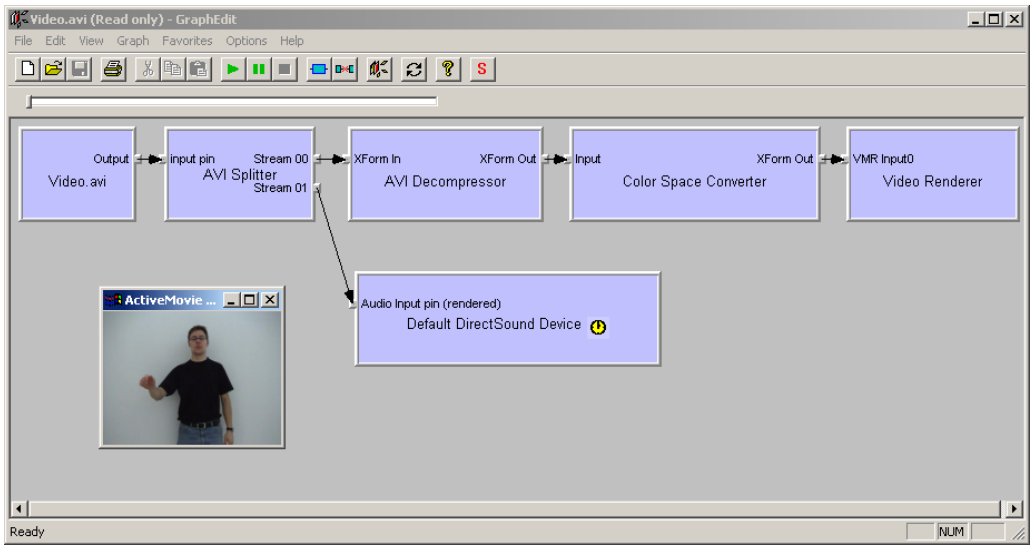


Abbildung 19: Rendern einer AVI-Datei

```
regsvr32.exe /u MyFilter.ax
```

Eine sehr praktische Funktion von DirectShow ist das Unterstützen von Property Pages für ein Filter. Prinzipiell dient die Property Page dazu dem Entwickler die Möglichkeit zu geben die Funktion des Filters während der Laufzeit zu verändern. Auch wir verwenden diese Möglichkeit um unsere Algorithmen zu parametrisieren.

Eine ausführliche Beschreibung von DirectShow findet sich in [8]. Auch die SDK-Dokumentation enthält viel Informationen.

Integrated Performance Primitives In einem Echtzeit-System müssen komplizierte mathematische Operationen immer wieder ausgeführt werden. Mit optimierten Funktionen kann die Leistungsfähigkeit stark erhöht werden. Bei Intel ist eine für Intel-Prozessoren optimierte Library erhältlich, mit der eine Vielzahl von Operationen in der Bild- und Signalverarbeitung zur Verfügung stehen.

Der Produktebeschrieb findet sich unter:
http://www.intel.com/software/products/perflib/index.htm

Microsoft Fundation Class Die MFC-Klassen vereinfachen die Benutzung der GUI-Komponenten. Mit Hilfe eines Assistenten in der Entwicklungsumbegung Visual Studio 6.0 lassen sich Bedienungskomponenten einfach hinzufügen. Die Funktionalität bei Benutzer-Interaktionen kann in dazu erstellten Methoden implementiert werden. Die komplizierte Handhabung der GUI-Komponenten wird so vor dem Programmierer verdeckt und er muss sich nur ums Wesentliche kümmern.

Die ausführliche Dokumentation findet sich in der MSDN-Library unter:
http://msdn.microsoft.com/library/default.asp

Doxygen Eine grosse Problematik beim Erstellen von Software ist deren Dokumentation. Leider ist es oft der Fall, dass eine vorhandene Dokumentation bereits veraltet ist und somit wenig hilfreich für das Verstehen des aktuellen Codes.

Diesem Problem wirkt das Software-Tool Doxygen entgegen. Die Idee von Doxygen ist es, nicht eine separate Dokumentation zu führen, sondern den Kommentar direkt in den Code zu schreiben. So sind beide Dinge in der gleichen Datei vorhanden und der Programmierer hat es einfacher, bei Änderungen die Dokumentation anzupassen.

Doxygen lässt sich über weite Bereiche konfigurieren. Dazu ist ein Konfigurationsfile namens `Doxyfile` vorhanden. Es ist möglich die Ausgabe in LaTeX, HTML oder in anderen Formaten zu erstellen. Für unsere Code-Dokumentation wurde die HTML-Ausgabe verwendet und eine Windows-Hilfedatei erstellt.

Mit folgendem Befehl lässt sich die gesamte Code-Dokumentation neu erstellen:

```
doxygen.exe Doxfile
```

Das Programm und dessen Dokumentation kann unter folgender Adresse heruntergeladen werden: http://www.doxygen.org

3.3 Design

Im Folgenden wird nun das Softwaredesign für die Realtime-Implementation beschrieben. Die Arbeit besteht aus vier Teilen:

1. Applikation
2. Audio Beat Detection Filter für DirectShow
3. Video Beat Detection Filter für DirectShow
4. Audio Scope Filter für DirectShow

Die Applikation erstellt das Graphical User Interface (GUI), verwaltet die DirectShow-Komponenten und reagiert auf die Benutzereingaben. Sie wird im nächsten Abschnitt (3.3.1) beschrieben.

Das Design für die beiden Beat Detection Filter ist identisch. Einige Klassen werden gemeinsam verwendet. Sie sind in 3.3.2 erklärt. Die Vererbungsstruktur für die einzelnen Algorithmen ist in 3.3.3 ersichtlich. Einige für das Verständnis wichtige Abläufe werden im Kapitel 3.3.4 erläutert.

Das Audio Scope Filter stellt die Kurvenform des Audiosignals dar. Dieses Filter wurde von einem Beispiel in der SDK von DirectX übernommen und ein wenig angepasst. Details finden sich in 3.3.5

Die gesamte Dokumentation aller Klassen, Methoden und Attribute ist auf der beiliegenden DVD zu finden. Hier werden nur die Design-Idee und wichtige Abläufe dokumentiert.

3.3.1 Applikation

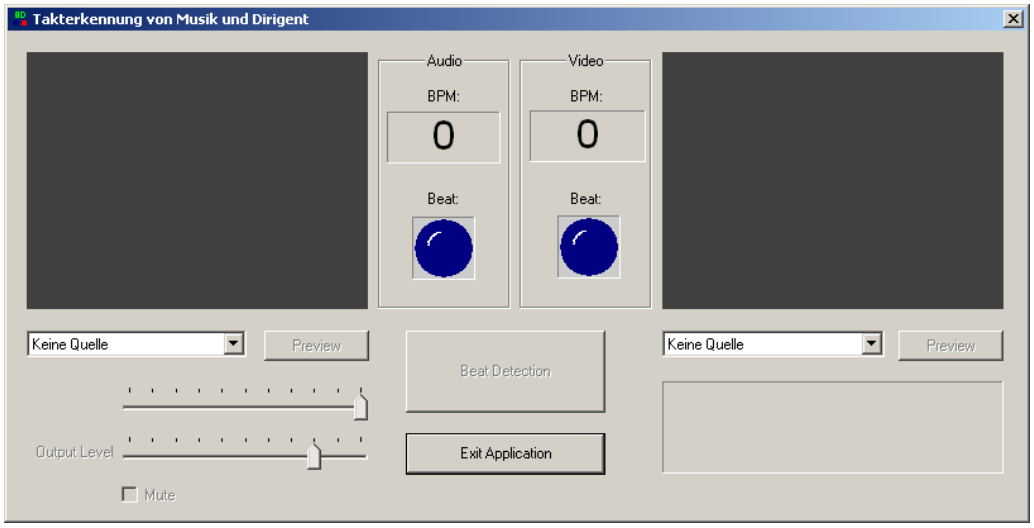


Abbildung 20: Das Benutzerinterface der Applikation

Das Benutzerinterface in Abbildung 20 wurde als Dialogbox mit dem Wizard in Visual Studio 6.0 erstellt. Wie schon erwähnt verdecken die MFC-Klassen das gesamte Handling. Man muss nur die Benutzerinteraktionen implementieren.

Das Verwalten der verschiedenen Funktionen wurde vom GUI gekapselt und in eine eigene Klassenstruktur implementiert. Abbildung 21 zeigt das Klassendiagramm. Im folgenden werden nun die einzelnen Klassen beschrieben.

Die Klasse `CMediaControl` ist die Schnittstelle zum GUI, repräsentiert durch die Klasse `CBeatDetectionDlg`. Die Änderungen durch den Benutzer werden durch diese Klasse ausgeführt. Sie werden zum Teil in dieser Klasse ausgeführt, andere werden an die entsprechenden Methoden in der Klasse `CFilterGraph` weitergeleitet. Zentral ist die Methode `ChangeState()`. Hier wird die gewünschte Funktion gesteuert. Das heisst zum Beispiel, dass eine AVI-Datei geladen wird, die Vorschau gestartet oder die Beat Detection aktiviert wird.

Die Klasse `CFilterGraph` implementiert die Grundinitialisierungen für den DirectShow Filtergraph. Sie implementiert das Interface `IBeatCB`, das dazu verwendet wird, die Beat-Ereignisse von dem Beat Detection Filter an die Applikation zu übermitteln. Der FilterGraph wird erst in den von ihm abgeleiteten Klassen fertig gestellt, da die eigentliche Funktion noch nicht bekannt ist.

`CWAVPlayGraph` erstellt den Filtergraph in Bild 22. Er wird verwendet um eine Audio-Datei abzuspielen.

`CAviPlayGraph` konfiguriert den Filtergraph wie in Bild 23. Er beinhaltet alle Elemente vom Einlesen der AVI-Datei über die Beat Detection Filter zur Ausgabe auf den Bildschirm.

Die letzte Klasse ist noch `CWAVLiveGraph`. Sie bildet den Filtergraph nach Bild 24. Als Eingang hat er die Kamera und den Aufnahme-Mixer der Soundkarte. Da der Mixer verwen-

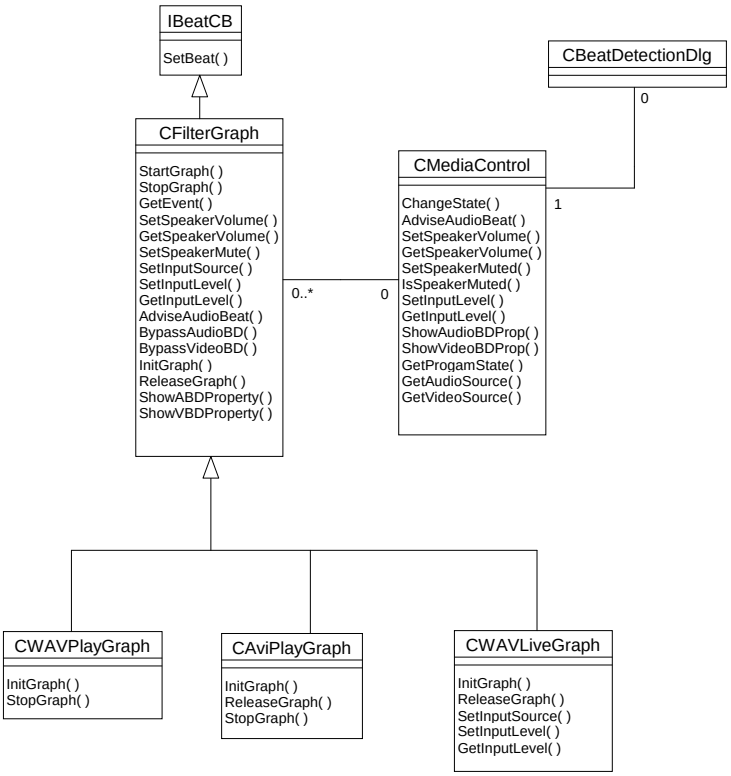


Abbildung 21: Klassendiagramm der Applikation

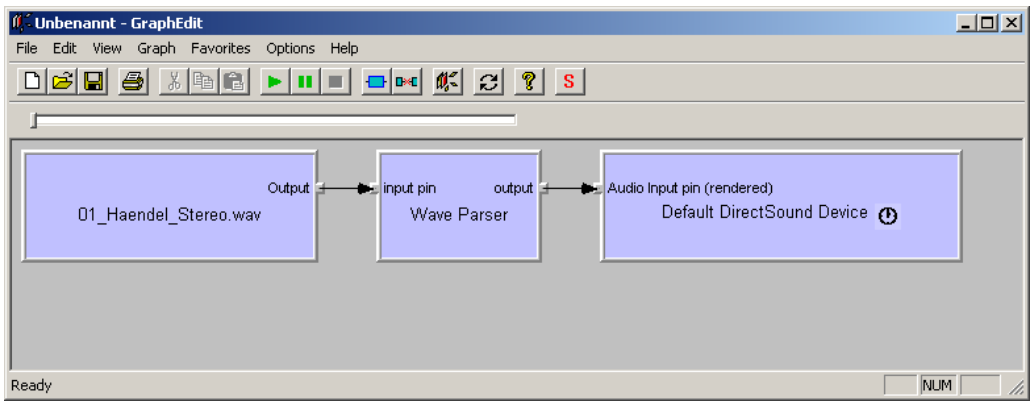


Abbildung 22: Filtergraph für die Klasse CWAVPlayGraph

det wird, ist es nicht nur möglich die durch die in der Klasse `CWAVPlayGraph` abgespielte Audio-Datei auszuwählen, sondern auch jede andere Quelle wie das Mikrofon oder der Line-In Eingang.

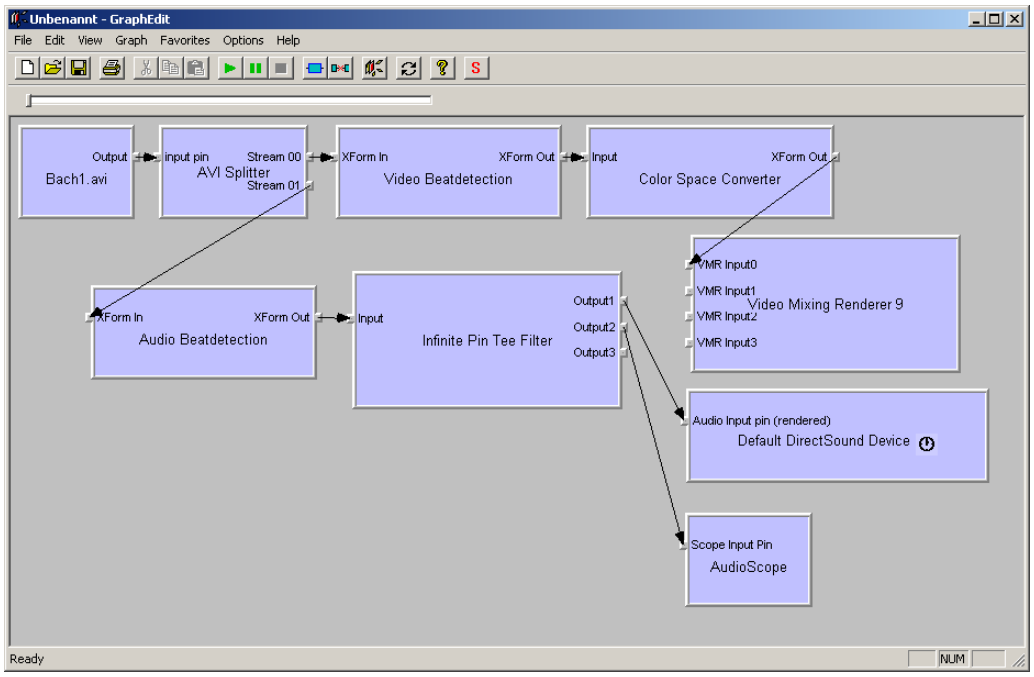


Abbildung 23: Filtergraph für die Klasse CAviPlayGraph

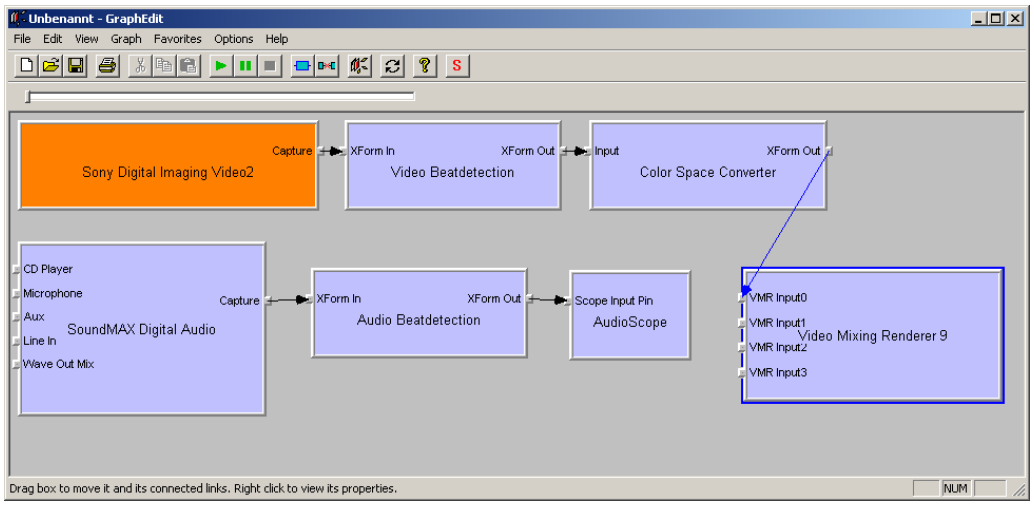


Abbildung 24: Filtergraph für die Klasse CWAVLiveGraph

3.3.2 Hilfsklassen

Beim Entwurf der Software haben wir festgestellt, dass einige Funktionen immer wieder verwendet werden und eine besondere Beachtung nötig haben. Wie im nächsten Abschnitt (3.3.3) erklärt wird, werden alle Algorithmen von der Klasse `CAlgorithm` abgeleitet und erben somit auch die Beziehungen, die in der Grafik 25 dargestellt sind.

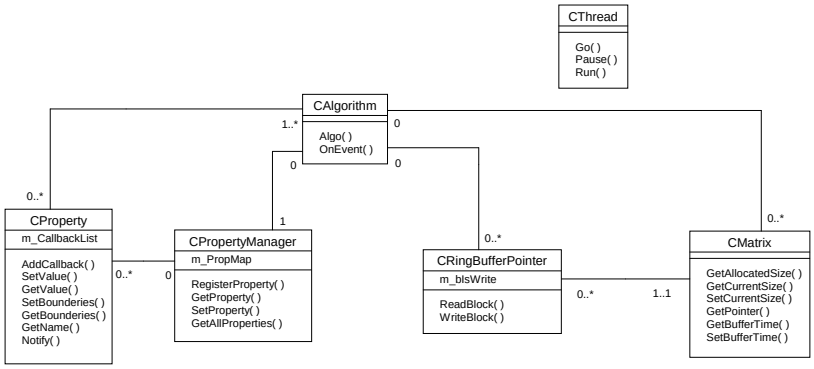


Abbildung 25: Klassendiagramm der Hilfs-Klassen

`CAlgorithm` hat einerseits die Aufgabe als Basisklasse für alle Algorithmen zu dienen, anderseits ist sie auch das Callbackobjekt, durch das ein Property über die Methode `OnEvent()` eine Änderung signalisiert.

`CProperty` kapselt einen Parameter für einen oder mehrere Algorithmen. Diese Klasse wurde erstellt, damit Parameter während der Laufzeit eingestellt werden können. Mit der Methode `AddCallback()` können Callbackobjekte registriert werden, welche Algorithmen automatisch orientieren, wenn sich ihr Wert ändert. Properties können den Wert `double` oder `int` haben, je nachdem für welche Anwendung sie benötigt werden. Eine weitere Eigenschaft sind auch Boundaries, die eine Begrenzung des Wertebereichs zulassen.

Damit die Properties im ganzen Filter bekannt sind, nur einmal erstellt und auch durch die Property Page verändert werden können, benötigt es `CPropertyManager` als Manager-Klasse. Wenn ein Algorithmus ein Property benötigt, instanziert er es nicht direkt. Er ruft die Methode `RegisterProperty()` auf und der PropertyManager erstellt ein Property für ihn, wenn es noch nicht vorhanden ist, oder gibt das vorhandene Objekt zurück. Die Property Page kann mit `GetAllProperties()` alle registrierten Properties holen und mit `GetProperty()` und `SetProperty()` den Wert lesen bzw. verändern.

Die Klasse `CMatrix` kapselt den Datenbuffer, der von Algorithmus zu Algorithmus weitergereicht wird. Sie kann mit verschiedenen Datentypen erstellt werden und speichert alle nötigen Attribute wie aktuelle Grösse und maximale Grösse. Da unsere Anwendung eine Real-Time-Anwendung ist, müssen wir darauf achten, dass nicht unnötig Daten kopiert werden. Darum ist der Datenblock nicht geschützt, mit `GetPointer()` wird der Pointer darauf zurückgegeben. Der Algorithmus ist somit selber verantwortlich, dass nicht über den allozierten Speicher geschrieben wird. Da sich jeder Datenblock zeitlich in den Daten-Stream einfügen lässt hat die Matrix noch ein weiteres Attribut, die BufferTime. Da Algorithmen die Timestamps benötigen ist somit auch dieser Informationsfluss gegeben.

`CRingBufferPointer` erlaubt mit Hilfe einer `CMatrix` einen Ringbuffer zu erstellen. Dieser ist nötig, wenn Algorithmen parallel ablaufen. Mit `WriteBlock()` werden Daten in den Buffer geschrieben, `ReadBlock()` liest sie wieder aus. Eine spezielle Eigenschaft des Lesepointer ist, dass er die Daten überlappend auslesen kann. Dies ist hilfreich für Algorithmen wie die FFT, deren Block sich mit dem vorherigen überlappt. Bedingt durch die Implementa-

tion ist pro Filter nur ein Ringbuffer möglich.

Die Klasse `CThread` kapselt den Windows-Thread in ein Objekt. Beim Instanzieren des Objekts ist der Thread gestoppt. Erst mit `Go()` wird er gestartet und kann mit `Pause()` wieder unterbrochen werden.

3.3.3 Algorithmus

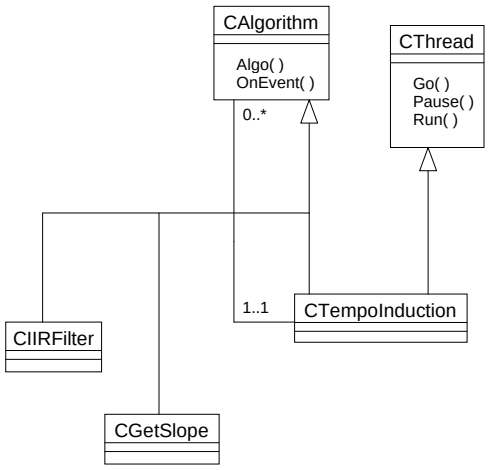


Abbildung 26: Klassendiagramm für den Algorithmus

Bild 26 zeigt die Vererbungsstruktur für einen Algorithmus. Ein Algorithmus ist ein Block, der eine bestimmte Funktion ausführt. Er wird durch die Methode `Algo()` aufgerufen. Je nach Aufgabe speichert er Daten und kann mit Hilfe des Property-Managers Parameter registrieren. Wenn er unabhängig zu anderen Algorithmen ausgeführt werden muss, erbt er die Klasse `CThread`. So entsteht ein flexibles System, mit dem man beliebige Algorithmen erstellen kann.

3.3.4 Sequenzdiagramme

Die Objekte `Algo1` und `Algo2` in Abbildung 27 laufen in verschiedenen Threads. `Algo1` besitzt den Write-Pointer, `Algo2` den Read-Pointer. `Algo1` kann mit `WriteBlock()` die Daten dem Write-Pointer übergeben, der sie in den Buffer schreibt. Analog dazu fordert `Algo2` die Daten mit `ReadBlock()`, der sie aus dem Buffer ausliest. Alle Zugriffe auf den Ringbuffer sind geschützt, so dass es nicht möglich ist, dass Lesen und Schreiben zum selben Zeitpunkt geschehen.

Abbildung 28 visualisiert den Prozess für die Verwendung von Properties. Der Algorithmus `Algo1` registriert ein Property. Nun kann er es verwenden und den Wert setzen, abrufen usw.

Als letztes wird in Abbildung 29 noch die Veränderung eines Property über die Property Page aufgezeigt. Die Property-Page, symbolisiert durch das Objekt `PropPage`, holt sich die Liste

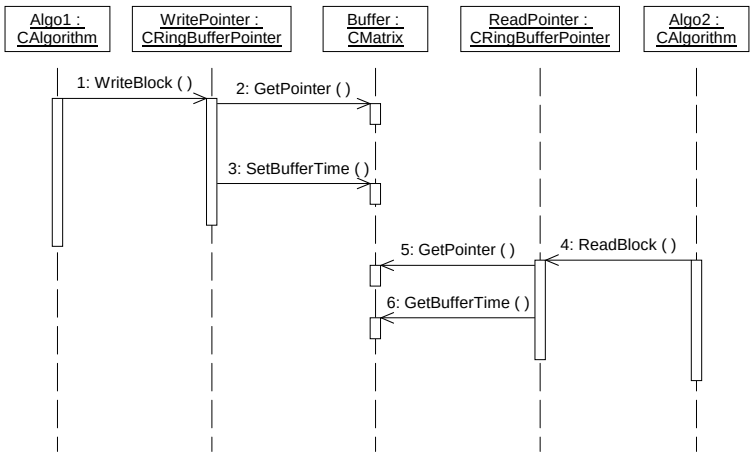


Abbildung 27: Sequenzdiagramm, das den Zugriff auf den Ringbuffer zeigt

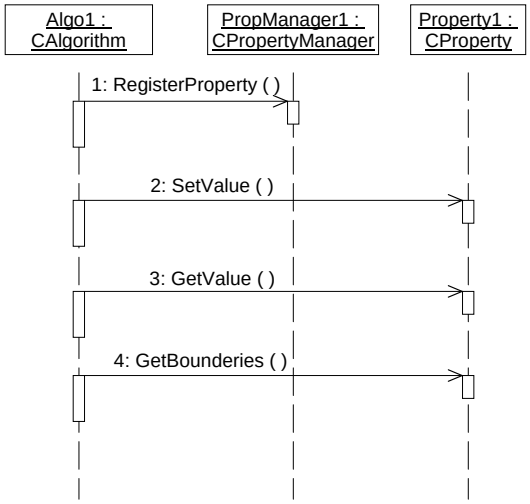


Abbildung 28: Sequenzdiagramm, das die Verwendung der Property zeigt

aller Properties mit `GetAllProperties()`. Nun kann sie mit `GetProperty()` den Wert eines Properties holen und mit `SetProperty()` einen neuen setzen. Der Propery-Manager ruft dann die entsprechenden Methoden des Property auf. Dieses wiederum benachrichtigt den Algorithmus über `OnEvent()`, dass sich der Wert verändert hat.

3.3.5 Audio-Scope

Wie bereits erwähnt, wurde für die Darstellung des Audio-Signals ein bereits bestehendes Filter verwendet. In der SDK-Dokumentation ist ein Oscilloscope vorhanden, das eine Bedienung ähnlich seinem grosser Bruder erlaubt. Gegenüber diesem Beispiel-Filter hat das verwendete Audio-Scope folgende Änderungen (der Unterschied ist in Bild 30 ersichtlich):

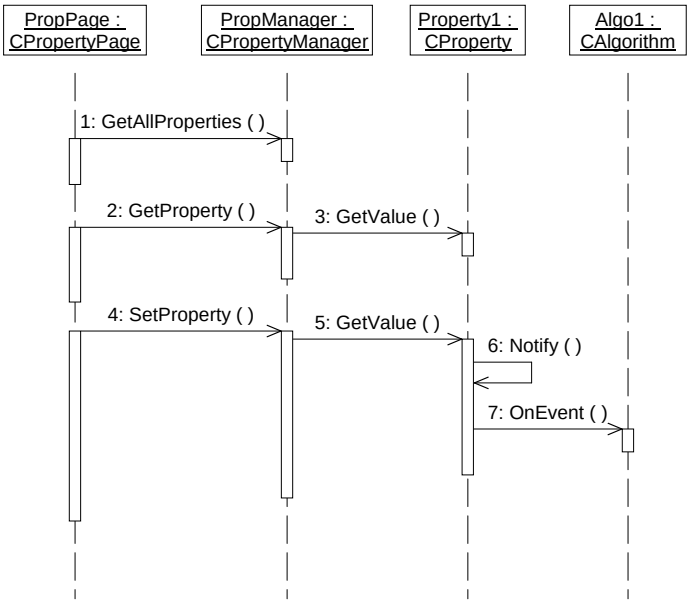


Abbildung 29: Sequenzdiagramm, das das Verändern eines Properties zeigt

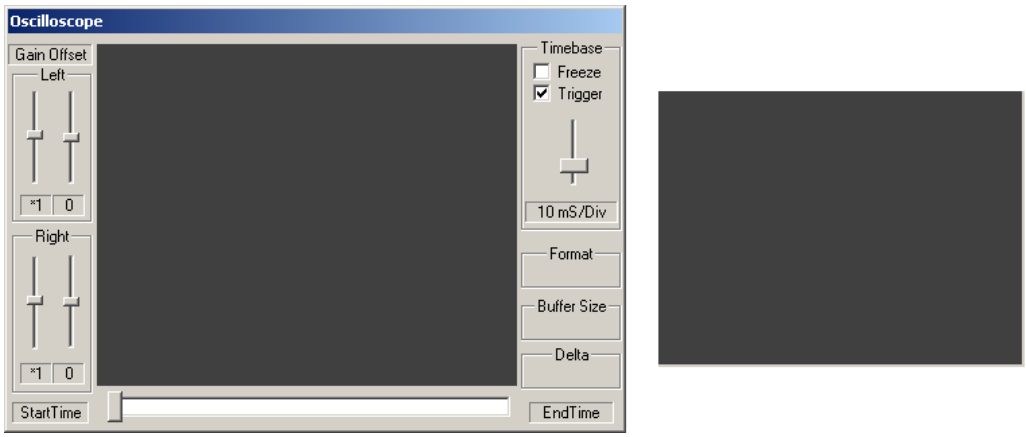


Abbildung 30: Links ist das Scope aus dem SDK-Beispiel, rechts das verwendete Audio-Scope

- Alle Bedienungselemente wurden entfernt.
- Die Dialogbox ist nicht mehr eigenständig, sondern muss in ein bestehendes Fenster eingefügt werden.
- Ein neues Interface, welches das Parent-Fenster dem Filter übergibt, wurde hinzugefügt.
- Stereo Audio-Daten werden nicht als zwei Signale, sondern als ein Summensignal dargestellt.

4 Systemtests

Zur Auswertung unseres Systems und auch zum späteren Gebrauch wurden Test-Videos erstellt. Diese sind, wie auch die benützten Audio-Dateien, auf der beiliegenden DVD enthalten.

4.1 Audio Beat Detection

4.1.1 Einleitung

Vor der Betrachtung der einzelnen Stücke seien an dieser Stelle einige allgemeine Bemerkungen angebracht. Schon von Beginn weg war klar, dass eine „allgemeine“ Beat Detection, die nicht auf eine Musiksparte oder -stilrichtung spezialisiert ist, nicht zu hundert Prozent fehlerfrei arbeiten kann. Dafür sind die Musikstile zu verschieden. So arbeitet unser Algorithmus bekanntlich mit der Amplitude des musikalischen Signals, was zwar auf viele Sparten anwendbar ist, aber eben nicht immer zuverlässig. So liessen sich Parameter für ein Musikstück einstellen, welches dann fehlerfrei liefe, ein anderes Stück erforderte aber ganz andere Einstellungen. Der Algorithmus ist also sozusagen ein Kompromiss, der für möglichst viele verschiedene Musikstücke möglichst gute Resultate liefern soll.

Nach einer Analyse aller von uns verwendeten Musikstücke folgt eine Erläuterung verschiedener Probleme und danach ein Fazit auch in tabellarischer Form. Ein Teil der Stücke kam erst in der Erprobungsphase dazu, viele Files benützten wir aber schon zu Beginn der Semesterarbeit. Wir entfernten bewusst keine heiklen Stellen in Musikstücken oder verzichteten ganz auf schlecht detektierbare Stücke, denn auch ein Misserfolg bietet in unserem Fall interessante Einblicke in die Thematik.

Nun ist noch zu sagen, dass die Resultate trotz den erstellten Video-Files nicht zu hundert Prozent reproduzierbar sind. Je nach Vorgeschichte, also je nach dem, ob das Tempo des vorangegangenen Stücks höher, tiefer oder ähnlich war als beim zu detektierenden Stück, oder wenn die Filter gänzlich mit Null initialisiert waren, variieren die Resultate ein wenig.

4.1.2 Analyse der Detektionen einzelner Musikstücke

Händel Ein Teil aus einem Concerto Grosso für Streicher von G.F. Händel steht einerseits für alte Musik, andererseits für Musik ganz ohne Schlagzeug oder scharfe Instrumente. Trotzdem funktioniert die Takterkennung nicht schlecht. Es ist möglich, dass das System in den Offbeat wechselt, findet aber meist wieder zurück. Erzwingt man mit einem Mausklick einen Beat und greift so zur rechten Zeit ins Geschehen ein, lässt sich das Offbeat-Problem gänzlich lösen. Das Ritardando vor der Wiederholung ist zum Teil ein Problem, während die kleineren Temposchwankungen eher eine Herausforderung bilden. Alles in allem ein recht gutes Ergebnis.

Purcell Auch unser Abschnitt aus „Dido and Aeneas“ ist, ähnlich wie der Händel, alte Streichermusik. So liegen auch die Probleme ähnlich. Das recht freie Musizieren hat einige Temposchwankungen zur Folge, welche nicht alle tadellos gemeistert werden. Entsteht

nach einer Phrase eine kleine Pause, so weiss das der Algorithmus natürlich nicht und prompt kommt ein Beat zu früh. Trotzdem findet er aber immer in den richtigen Takt zurück. Auch bei diesem Stück hilft ein Mausklick über solche Hindernisse hinweg, so dass das Resultat auch hier als recht gut bezeichnet werden kann.

Bach1 Klaviermusik ist für unsere Takterkennung insofern schwierig, als Pianisten meist sehr frei musizieren und Phrasen mit viel Ritardando abschliessen, um so ein deutliches a Tempo zu erzielen. Dies ist auch in unserem ersten Ausschnitt aus Bachs französischer Suite anzutreffen. Der Algorithmus fällt normalerweise in den Offbeat, was mit den durchgehenden Sechzehntel-Bewegungen zu erklären ist. Nach einiger Zeit findet er aber den Onbeat und behält diesen sogar über alle Temposchwankungen hinweg. Zeigt man dem Algorithmus den Onbeat per Mausklick, gibt es fehlerfreie Versionen. Allerdings fällt das System, je nach dem wo ein Beat erzwungen wird, nach einer Temposchwankung in den Offbeat zurück. Nicht zu schaffen ist aber in jedem Fall das Schlussritardando. Trotzdem ein gutes Ergebnis.

Bach2 Auch der zweite Ausschnitt aus der französischen Suite bietet die oben erwähnten Probleme. Zeigt man dem Algorithmus den Onbeat, so sind gute Resultate möglich.

Bergamasca Bei diesem Ausschnitt handelt es sich um den Schluss der Bergamasca aus einer von Respighis antiken Suiten. Das Orchester wird von Pauken begleitet, was sich natürlich positiv auf die Takterkennung auswirkt. Arbeitet der Algorithmus ganz ohne fremde Hilfe, fällt er nur kurze Zeit in den Offbeat um danach wieder richtig weiter zu fahren. Das zuerst nur zögerliche Schlussritardando macht er gut mit, und nach einem kurzen Aussetzter erwischt der Algorithmus die Schläge dort wieder, wo das halbe Tempo erreicht wird. Ein erfreuliches Resultat.

Supplice Diese Stelle aus Berlioz' „Marsch zum Schafott“ wird von den hellen Blechbläserklängen dominiert und fängt für die Takterkennung sehr gut an. Wo aber die Bläser etwas Tempo verlieren, wird es kurz schwierig, was natürlich auf die Beatdetektion Einfluss hat. Dort werden dann die Offsets detektiert, was bis zum Schluss anhält. Dem kann man natürlich mit einem Mausklick an geeigneter Stelle entgegen steuern.

Brahms Der Anfang von Brahms' erster Sinfonie sollte, wenn man nur oberflächlich hinhört, kein Problem sein. Tatsächlich werden alle Paukenschläge als Peaks detektiert. Die Tempo Induction ist aber mit dem „Noise“, den die Streicher erzeugen, überfordert und findet kein sinnvolles Tempo. So werden die Beats dann mehr zufällig gesetzt.

Mogul Emperors Der „March of the Mogul Emperors“ aus Elgars Suite „Crown of India“ freut einem nicht nur wegen der ungewohnten Klangfarben, auch an der Beatdetektion kann man seine Freude haben. Ohne äussere Hilfe beginnt der Algorithmus mit dem Offbeat, fällt dann aber in den Onbeat und hält diesen bis zum Schluss. Startet man auf dem Beat, gelingt gar ein Null-Fehler-Ritt.

Bolero Ravels Bolero wird bekanntlich vom durchgehenden Rhythmus dominiert, welcher dem Stück ja auch seinen Namen gegeben hat. So ist es nicht erstaunlich, dass die Takterkennung über weite Strecken ausgezeichnet funktioniert. An zwei Stellen gibt die Tempo Induction das doppelte Tempo an, was den Beat erst in den Achtel fallen lässt um dann bei seiner Rückkehr zum Viertel im Offbeat zu landen. Hilft man mit

einem Mausklick nach ist das allerdings kein Problem. Nur bei der Dur-Wendung zum Schluss des Stücks ist die Musik dann etwas zu chaotisch für den Algorithmus und er findet sich nicht mehr ganz zurecht.

America Dieses kurze Stück aus dem Schlussteil von Charles Ives' Variationen über „America" bieten dem Algorithmus gar kein Problem. Es ist aber möglich, dass das System, je nach „Vorgeschichte", im Offbeat beginnt.

Bulgarisches Lied Ein Problem in diesem Bulgarischen Lied über die Füchsin mit ihre Jungen sind die Tempowechsel. Der gesungene Teil ist nicht nur stilistisch ruhiger als der Bläserteil, das Tempo ist tatsächlich etwas langsamer. Da wir aber mit unserem Kamm-Filter eine gewisse Verzögerung haben, kommen wir bereits an die Grenzen des Systems. Das andere Problem ist die grosse Unruhe mit relativ wenig rhythmischen Gewichten im Bläserteil, welches aber nicht schlecht gemeistert wird.

Fränzlis Marcha Dieser volkstümliche Marsch, von einer Bündner Kapelle gespielt, bietet ausser dem allgemeinen Offbeat-Problem keinerlei Schwierigkeiten. Die kleineren Temposchwankungen sind bewältigbar.

Bluegrass Diese Version von Greensleeves ist ein typisches Beispiel für den Bluegrass. Zwar ist ein klarer und regelmässiger Bass zu hören, das schnelle Banjo bringt aber eine Unruhe ins Spiel und die Geigenmelodie beinhaltet auch recht viele Verzierungen. Trotzdem wird der Takt gut erkannt. Auch im letzten Teil unseres Abschnittes, wenn sogar die Basslinie sehr virtuos wird.

Maple Leaf Rag Auch bei etwas moderneren Klavierstücken als den oben erwähnten haben wir immer noch die selben Schwierigkeiten: Eine sehr freie Gestaltung des Stücks mit vielen und zum Teil recht grossen Temposchwankungen. Bei einem Rag kommt noch hinzu, dass die Betonungen generell auf dem Offbeat liegen und darüber hinaus auf den Schlag häufig Pausen kommen. So erstaunt es nicht, dass unser Algorithmus mit dem Maple Leaf Rag seine Mühe hat. Einzig ein fleissiges Erzwingen des richtigen Beats per Mausklick führt zu einigermassen befriedigenden Resultaten.

Danke schön Das Offbeat-Problem entfällt natürlich, wenn der Algorithmus die Achtelnoten statt die Viertel detektiert. Dieser Jazz-Standard beginnt auf diese Weise, nach einiger Zeit wechselt aber das Tempo auf seine Hälfte. Die Beats werden jedoch sauber erkannt und getroffen, was natürlich auch dem konstanten Tempo zuzuschreiben ist.

Yellow Sub Mit Yellow Submarine tut sich vor allem die Tempo Induction schwer. Zwar bestimmt sie den Viertel in der Strophe korrekt mit Tempo 112, beim Refrain detektiert sie aber beide Male Tempo 80. Somit wird der Takt natürlich nur in der Strophe richtig bestimmt.

Let it Be Der Takt des Beatles-Spätwerks wird im allgemeinen gut erkannt. Der Algorithmus verliert sich allerdings gegen den Schluss des Ausschnitts, wobei man aber sehr schön hört, wie er nach zu vielen selbst gesetzten Beats einen Neustart ausführt und so den richtigen Takt wieder erwischt.

Saigon Twist Bei diesem Twist von Perez Prado ist es interessant, wie die Tempo Induction zwischen dem Tempo des Viertels und dem Tempo der halben Note wechselt, je nach

dem, wie dominant die Shuffle-Basslinie gegenüber den anderen Instrumenten ist. Die Beats werden aber sauber gesetzt.

Yolanda Das Stück „Donde estas Yolanda“ der Combo Pink Martini bietet einzig das Offbeat-Problem, wenn bei einem Übergang etwas Zeit oder Tempo verloren geht, das dann wieder aufgeholt werden muss.

Sympathique Bei diesem Pink Martini-Stück sieht man sehr schön, wie die Spielweise Auswirkungen auf die Takterkennung hat. Unser Ausschnitt beginnt mit dem Refrain, wo die halbe Note als Tempo detektiert wird. Bei der Strophe spielt das Klavier kurze Viertel, was die Tempo Induction veranlasst, auf den Viertel zu wechseln. Zurück beim Refrain wird dann wieder das halbe Tempo bestimmt. Die Beats werden durchgehend schön bestimmt, wobei beim Refrain natürlich die Gefahr besteht, in den Offbeat zu fallen.

W.Nuss Mit Patent Ochsners „W.Nuss vo Bümpliz“ gelänge beinahe ein perfekter Lauf, hätte nicht die Tempo Induction einen kurzen Aussetzer. Trotzdem sehr schön.

Our House Stücke wie „Our House“ von Madness sind einfache Kandidaten für unsere Takterkennung. Ein dominantes und regelmässiges, vor allem aber gerades Schlagzeug sorgen für ein leichtes Erkennen des richtigen Beats.

Be my Baby Auch „Be my Baby“ von Vanessa Paradis, hier eine Live-Aufnahme, meistert das System fehlerfrei.

Chieftains Obwohl nicht Mainstream-Pop, bietet auch dieser Ausschnitt aus einem Chieftains-Stück keine Probleme. Die gute Akzentuierung auf die Taktanfänge und die durchgehende Trommel sorgen dafür.

Mull of Kentyre Auch hier spielt uns die Tempo Induction einen Streich. Was anfangs einen sehr guten Eindruck macht, trübt die Freude durch die Bestimmung eines falschen Tempos. Die Ursache für die Fehlentscheidung liegt bei der Gitarre, welche durch die ternäre Begleitung eine Art Triolen vortäuscht, was prompt detektiert wird.

Perotine Dank der dominanten und regelmässigen Basslinie der Walliser Band „Glen of Guinness“ gelingt hier der Null-Fehler-Ritt. Die Basstöne entstehen übrigens durch digital um eine Oktave vertiefte Posaunentöne.

Spacil Hill Ein interessantes Phänomen ist hier zu beobachten. Das Tempo eines Taktes von „Spancil Hill“ ist etwa 68. Die Tempo Induction erkennt die Achtel mit Tempo 204, denn es hat drei davon in jedem Takt. Da ihr dieses Tempo zu schnell ist, halbiert sie es auf 102 und die Takterkennung detektiert zwar Viertel, was aber im Dreiachteltakt keinen Sinn ergibt.

4.1.3 Generelle Probleme

Das Offbeat-Problem Ein immer wieder auftauchendes Problem ist die Verwechslung von Onbeat und Offbeat. Selbst für einen unmusikalischen Menschen ist es nicht schwierig, bei einem Stück im Takt auf den Schlag zu klatschen. Es ist für „volkstümliche“ Leute zum Teil

sogar schwierig den Nachschlag zu erwischen. Ein Computer hat aber weder musikalisches Verständnis noch Erfahrung im Musik hören. Somit kann er nicht wissen, ob ein Schlag Onbeat oder Offbeat ist.

Natürlich gibt es bei vielen Musikstilen leicht zu implementierende Merkmale. So sind bei Techno, House und generell bei moderner Musik Basstöne und tiefe Schlagzeug-Beats meist Onbeats, während scharfe Klänge von Becken oder Snares den Offbeat anzeigen. Da wir aber auch Musik ohne Schlagzeug benützen wollten, kam eine solche Lösung des Problems nie in Frage.

Würde man immer die kleinen Einheiten, also Achtel statt Viertel oder Viertel statt halbe Noten, detektieren, hätte man das Offbeat-Problem auch nicht. Es würden nämlich immer On- und Offbeat bestimmt. Da wir aber eine grosse Spannweite von Tempi haben, fällt auch dieser Ansatz weg.

So entschieden wir uns zu einem kleinen Kunstgriff: Ein Mausklick erzwingt einen Beat, und von diesem aus berechnet der Algorithmus dann alles Weitere. Unser System arbeitet so zwar nicht immer selbständig, dies ist aber ein Kompromiss, den wir eingegangen sind um eine möglichst grosse Musikvielfalt zu haben.

Die Temposchwankungen Die Bestimmung des Tempos nach Klapuri [4] ist sehr zuverlässig und auch interessant, hat aber einen Mangel. Das Kamm-Filter speichert eine Vergangenheit von vier Sekunden. Diese Zeit ist auch nötig um das Tempo konstant und ohne Sprünge und Flatterhaftigkeiten zu bestimmen. Schliesslich erlaubt ein Echtzeit-System nur den Blick zurück. Daraus ergibt sich aber auch eine gewisse Trägheit. Leichte Temposchwankungen, wie sie fast überall vorkommen und auch natürlich sind, ergeben keine Schwierigkeiten. Diese werden auch durch die Toleranzen beim Setzen der Beats ausgeglichen. Auch leichte Accelerandi und Ritardandi vermag das System zu bewältigen. Werden diese aber zu stark oder ändert das Tempo abrupt, so braucht der Algorithmus etwas Zeit um das neue Tempo zu bestimmen und die Beats danach an die richtigen Orte zu setzen.

Minime Abweichungen der Beats Nicht immer sitzt der gesetzte Beat haargenau dort, wo er hingehört. Um das System so nahe wie möglich an der Verarbeitung in Echtzeit zu halten setzt FindBeat die Beats in der Zukunft. Das heisst, der Beat wird vorausgesagt und erst später wird kontrolliert, wo genau er hätte sein sollen. Dies ist nötig, damit keine unnötigen Verzögerungen entstehen, und sollte nicht störend auf die Benutzerin und den Benutzer wirken.

Taktarten Bisher haben wir die Taktarten nicht erwähnt. Theoretisch sind alle Taktarten detektierbar, welche Schläge konstanter Länge haben. Dies ist bei Zweiviertel-, Dreiviertel-, Vierviertel-, Fünfviertel[3]-, Sechsachtel- und ähnlichen Takten der Fall. Siebenachtel- oder Dreizehnsechzehnteltakte können nicht verwendet werden, da die Schläge innerhalb eines Taktes verschiedene Längen haben.

[3] wenn die Viertel ausdirigiert werden

4.1.4 Fazit

Musikstück	**Stil**	**Tempo Induction** [1]	**Takterkennung**
Händel	Kammermusik	100	gut [4]
Purcell	Kammermusik	100	gut, mehr Hilfe nötig
Bach1	Klassik (Klavier)	95 [2]	gut, mehr Hilfe nötig
Bach2	Klassik (Klavier)	100	viel Hilfe nötig
Bergamasca	Sinfonik	95 [2]	gut
Supplice	Sinfonik	100	gut
Brahms	Sinfonik	0	—
Mogul Emperors	Sinfonik	100	gut
Bolero	Sinfonik	95	gut
America	Sinfonik	100	sehr gut [5]
Bulg Lied	Bulgarische Volksmusik	100 [3]	schlecht
Fränzlis Marcha	Schweizer Volksmusik	100	sehr gut
Bluegrass	Bluegrass	100	gut
Maple Leaf Rag	Jazz (Klavier)	95	schlecht
Danke schön	Jazz	100 [3]	gut
Yellow Sub	Pop	40	gut, wo Tempo Induction korrekt
Let it Be	Pop	95 [2]	gut
Saigon Twist	Latin	100 [3]	gut
Yolanda	Latin	100	sehr gut
Sympathique	Chanson	100 [3]	gut
W.Nuss	Pop	98	sehr gut
Our House	Pop	100	sehr gut
Be my Baby	Pop	100	sehr gut
Chieftains	Irish Folk	100	sehr gut
Mull of Kentyre	Folk Pop	50	gut
Perotine	Irish Folk	100	sehr gut
Spancil Hill	Irish Folk	0	—

Tabelle 1: Übersicht Auswertung

Allgemein lässt sich also sagen, dass der Algorithmus den Erwartungen entspricht. Trotz der grossen Vielfalt an Musikrichtungen wird der Takt verschiedener Musikstücke recht gut

[1] Die Zahlen geben die ungefähre Zeit in Prozent an, in welcher das richtige Tempo herausgefunden wurde.
[2] Gut, bis auf das Schlussritardando
[3] Tempo Induction wechselt zwischen Viertel- und Achtel-Erkennung.
[4] gut: Erkennung der meisten Beats. Aussetzer möglich, findet sich aber wieder.
[5] seht gut: Erkennung aller Beats ohne fremde Hilfe.

bestimmt, ohne dass Parameter verstellt werden müssen. Bei Stücken mit dominantem und regelmässigem Schlagzeug oder Bass wird der Takt generell am besten bestimmt, wobei auch schlagzeuglose Musik kein ernsthaftes Problem darstellt, sofern sie eine gewisse Regelmässigkeit aufweist. Ein Überblick über alle analysierten Musikstücke gibt Tabelle 1 (Angaben bei teilweisem Erzwingen des Beats per Mausklick).

4.2 Video Beat Detection

4.2.1 Einleitung

Als Erstes sollte gesagt werden, dass man bei unserer Arbeit nicht ohne weiteres von Dirigieren sprechen kann. Vor allem mit dem Dirigieren eines Sinfonieorchesters, wo der Maestro vor allem Phrasen anzeigt und den gewünschten Stil auf die Musiker überträgt, hat mit unserer Art von Dirigieren wenig zu tun. Am ehesten entspricht unser Dirigieren dem Stil vieler Leiter von schlechten Blaskapellen, wo vor allem das Tempo angezeigt wird und so alle möglichst zusammen spielen sollen. Denn genau dem entspricht ja unsere Aufgabe; wir haben ausschliesslich mit Tempo und Rhythmus zu tun. So könnte man in unserem Fall von Taktieren sprechen.

In den nächsten Abschnitten wollen wir zuerst definieren, wie jemand dirigieren soll, damit unsere Takterkennung zuverlässig funktioniert. Weiter verlieren wir ein paar Worte über Taktarten und vor allem über Tempi und Tempoänderungen. An einigen Beispielvideos wird gezeigt, was gut funktioniert und wo das System seine Grenzen hat. Schliesslich folgt ein Fazit über die Analyse des Video-Teils.

4.2.2 Wie man dirigiert

Das gelbe Quadrat in der GUI-Anzeige folgt dem am schnellsten vor der Kamera bewegten Objekt. Deshalb ist es unabdingbar, dass man nur mit einer Hand den Takt schlägt, und nicht etwa mit beiden Händen. Auch sollte man sich sonst still verhalten und nicht herum zappeln, denn dies erschwert die Arbeit für den Algorithmus nur unnötig. Da der Algorithmus auf vertikale Bewegungen anspricht, empfiehlt es sich „rund“ zu dirigieren, und nicht eckig oder zackig, weil so rein horizontale Bewegungen entstehen. Da dies ohnehin der natürlichsten Bewegung entspricht, werden die meisten Leute wohl von sich aus so dirigieren. Als letztes ist es wichtig, dass die dirigierende Person so im Bild steht, dass der bewegte Arm immer sichtbar ist. Ansonsten ist das System aber sehr geduldig mit mehr oder weniger gut ausgebildeten Dirigenten und detektiert munter verschiedene Tempi und Taktarten.

4.2.3 Tempi und Taktarten

Bei den Taktarten gibt es nur eine Einschränkung: Die Schläge müssen immer die selbe Länge haben. So eignen sich Viervierteltakt, Dreivierteltakt, Zweivierteltakt, Sechsachteltakt und so weiter, aber auch Fünfvierteltakt und ähnliche, wenn der Viertel ausdirigiert wird. Ungerade Taktarten wie Siebenachtel- oder Elfsechzehnteltakt funktionieren nicht, da die Schläge unterschiedliche Längen haben.

Auch beim Tempo gibt es Einschränkungen. Das höchste noch gut erkennbare Tempo liegt etwas über 130 Schläge pro Minute. Dabei ist aber zu sagen, dass mit den benötigten ausladenden Bewegungen höhere Tempi kaum zu bewältigen sind, ohne dass einem der Arm zu schmerzen beginnt. In diesem Fall wechselt man besser ins halbe Tempo und dirigiert alla breve.

Eine untere Grenze gibt es nicht. Die Takterkennung hat selbst bei Tempo 30 kein Problem, wenn der Maestro schön und regelmässig dirigiert. Dies dürfte aber äusserst selten der Fall sein, normalerweise werden bei solch langsamen Tempi die Schläge verdoppelt.

Auch Tempoänderungen macht der Algorithmus gut mit. Bei Accelerandi wie bei Ritardandi ist er praktisch ohne merkliche Verzögerung dabei, was im Video-File „AccRit.avi“ belegt ist. Ändert das Tempo jedoch plötzlich, braucht das System einige Zeit, bis es sich auf die neue Situation eingestellt hat. Bei einem plötzlichen Wechsel von Tempo 60 auf 120 brauchte der Algorithmus 8 Schläge, was 4 Sekunden entspricht. In der umgekerten Richtung waren es mit 5 Schläge gerade 5 Sekunden.

4.2.4 Beispiele und Erläuterungen anhand Video-Files

Da die verschieden Musikstücke in diesem Kapitel nicht interessieren, begnügen wir uns mit der Analyse einiger Video-Files, um möglichst viele Dirigiertechniken abzudecken.

Händel Die Detektion ist kein Problem, auch das Ritardando vor der Wiederholung macht der Algorithmus gut mit. Auch kein Problem war die Kopfdrehung des Dirigenten, als dieser vorübergehend abgelenkt wurde. Solche Bewegungen beeinträchtigen den Algorithmus also nicht, solange der Arm weiter bewegt wird.

Bach1 Abgesehen davon, dass wohl zum ersten Mal ein Solo-Klavierstück dirigiert wurde, sieht man hier, dass Tempo 160 für den Algorithmus zu schnell ist. Ausserdem ist das Dirigieren bei diesem Tempo nur noch möglich, wenn die Bewegungen klein gehalten werden, was für die Takterkennung auch wieder ein Problem ist.

Bergamasca Dieser Zweizweiteltakt wird problemlos erkannt. Sogar das Schlussritardando lässt sich einigermassen sehen, wenn von der halben Note abrupt in den Viertel gewechselt wird.

Mogul Emperors Ein auf diese Weise dirigierter Dreivierteltakt wird vom System gut erkannt.

Bolero Bei der Aufnahme „BoleroSchlechtDirigiert.avi“ hat der Algorithmus Mühe, was aber mit dem Dirigenten zu tun hat. Seine Bewegungen sind weder fliessend noch regelmässig, sondern eher unnatürlich und zackig. Auch ein Orchester würde dazu nicht schön spielen. Nach etwas Übung gelang es aber doch, eine akzeptable Version aufzunehmen, was mit dem File „BoleroGutDirigiert.avi“ belegt ist. Somit wird deutlich, dass bei der Video-Takterkennung vor allem die Qualität des Dirigierens ausschlaggebend ist.

Bulgarisches Lied Auch hier ist ersichtlich, wie das zu schnelle Tempo nicht erkannt wird, das langsame jedoch schon. Der Bläserteil, im Viervierteltakt dirigiert, wird nicht detektiert, der gesungene Teil im Zweizweiteltakt jedoch schon.

Let it Be Dieser Ausschnitt ist ein Beispiel für einen schön dirigierten Viervierteltakt, der infolgedessen auch gut erkannt wird.

Perotine Auch unkoordinierte Bewegungen führen zu einem guten Resultat, solange sie regelmässig, also im Takt, und deutlich genug sind.

Alle anderen Video-Files wiederholen das oben Dargestellte und eine Auflistung derselben ergäbe daher keinen Sinn.

4.2.5 Fazit

Zusammenfassend lässt sich also sagen, dass die Video-Takterkennung unter folgenden Bedingungen sehr gut funktioniert:

- Deutlich und mit grossen Bewegungen dirigieren
- „Rund“ dirigieren, damit stets eine Vertikalbewegung ausgeführt wird
- Nur mit einer Hand dirigieren, den restlichen Körper ruhig halten

Dabei sind „natürliche“ Tempoänderungen kein Problem, abrupte Wechsel benötigen ein wenig Zeit.

5 User's Guide

5.1 Systemanforderungen

Damit das Demosystem zuverlässig läuft, sind einige Bedingungen an das System gestellt. Auf unserem Entwicklungs-PC hat alles gut funktioniert. Er weist folgende Kenndaten auf:

- Fujitsu Siemens Motherboard
- Intel Pentium IV Prozessor 2.4 GHz
- Matrox Millenium G550 Grafikkarte
- USB 2.0 Schnittstelle
- Onboard SoundMax Soundkarte
- Sony Digital Handycam DCR-TRV19E
- Windows 2000 Service Pack 4
- Microsoft DirectX 9.0b
- Intel Performance Library IPP 3.0

Bei laufendem System konsumiert die Applikation ca. 50% - 60% der Rechenleistung. Das bedeutet, dass auch Rechner mit weniger CPU-Leistung den Rechenaufwand bewältigen.

Die USB 2.0 Schnittstelle ist für die Sony Handycam nötig. Da DirectX verwendet wird, ist die Applikation nicht zwingend an diese Kamera gebunden. Damit es aber funktioniert, muss sie einen Video-Stream mit der Auflösung von 160 x 120 Pixel und den Formaten RGB24, UYVY oder YUY2 liefern.

Ein Problem stellt die Soundkarte dar. Nicht, dass die Funktion mit einer anderen Karte nicht gewährleistet wäre. Das Problem besteht darin, dass die Namen der Soundkarten-Eingänge nicht bei allen Herstellern gleich sind. Die Software sucht die Eingänge mit Hilfe von folgenden Zeichenketten: „CD“ für „CD Player“, „Mic“ für „Microphone“, „Line“ für „Line-In“ und schliesslich „Out“ für „Wave Out Mix“. Abbildung 31 zeigt den Windows Aufnahme-Mixer für die auf dem Enwicklungssystem verwendete Soundkarte. Ist ein Kanal anders benannt, kann er nicht automatisch ausgewählt werden. Wenn man jedoch von Hand im Windows-Mixer den gewünschten Eingang wählt, funktioniert auch der Audio Beat Detection Teil.

Bei der Verwendung von Windows 2000 ist das Service Pack 4 notwendig. Auf anderen Betriebssystemen wurde das Demosystem nicht getestet. Laut der Beschreibung von DirectX sind die verwendeten Funktionen auch auf Windows XP, Windows 98SE und Windows ME funktionsfähig.

DirectX 9.0b muss zwingend installiert werden, da Funktionen, die erst ab dieser Version vorhanden sind, verwendet werden.

Die Grafikkarte muss von der neusten Generation sein und DirectX 9 unterstützen. Sonst ist es nicht möglich, das Demosystem zu betreiben.

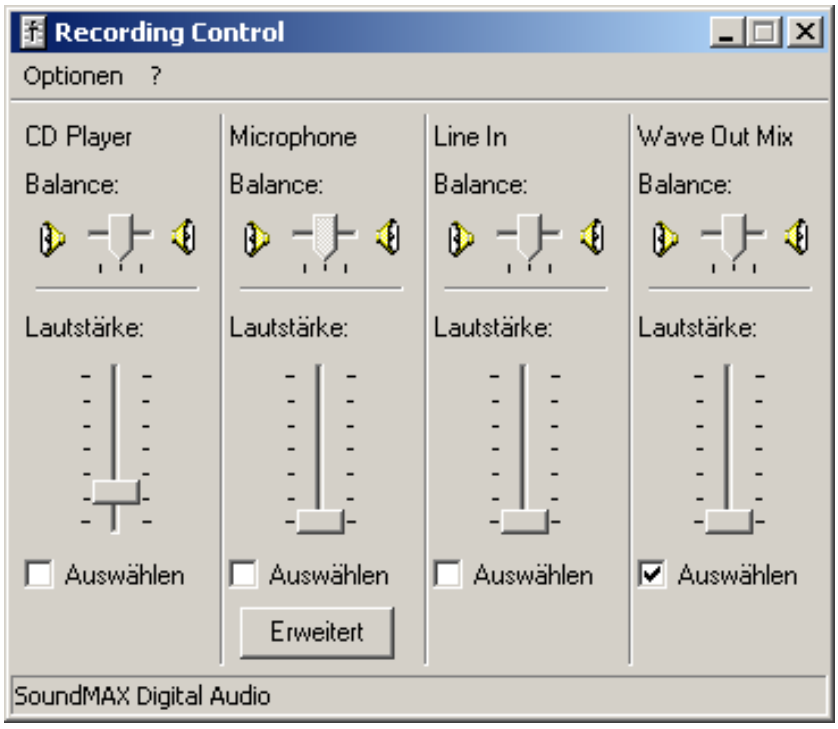


Abbildung 31: Der Windows Aufnahme-Mixer

5.2 Installation

Für die Installation müssen folgende Schritte ausgeführt werden:

1. Als Erstes muss eine funktionsfähige Soundkarte vorhanden sein. Will man live einen Dirigenten filmen, muss auch eine Videokamera im System verfügbar sein.
2. Wenn Windows 2000 verwendet wird, muss das neuste Service Pack (zur Zeit sp4) installiert werden. Es lässt sich im Internet herunter laden von http://www.microsoft.com.
3. DirectX 9.0b Run-Time installieren. Auf der DVD befindet sich die Installations-Datei `dx90update_redist.exe`. Eventuell auch den Grafikkarten-Treiber auf den neusten Stand bringen.
4. Intel Performance Library installieren. Auf der DVD befindet sich die Datei `setup.exe`
5. Dateien für die Beat Detection Applikation in ein Verzeichnis auf der lokalen Harddisk kopieren.
6. Von dort die Batch-Datei `install.bat` ausführen. Sie installiert die notwendigen DirectShow Filter.
7. Als letztes kann die Beat Detection Applikation (`BeatDetection.exe`) gestartet werden.

5.3 Fehlermeldungen

Es kann der Fall auftreten, dass eine Funktion nicht wie gewünscht ausgeführt werden kann. Es erscheint eine Fehlermeldung wie in Bild 32. Der Text sagt etwas über die Art des Fehlers aus, ist aber mehr für den Entwickler gedacht als für den Benutzer.

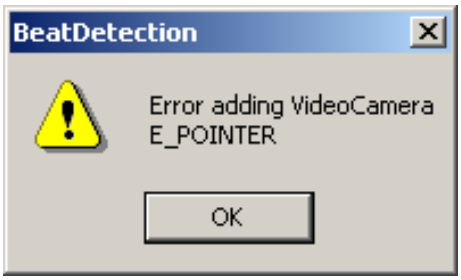


Abbildung 32: Fehlermeldung

Nachdem der Fehler bestätigt wurde, kann das Programm weiter benutzt werden. Die Quellenwahl geht auf „Keine Quelle“.

Ein Fehler kann zum Beispiel auftreten, wenn keine Kamera angeschlossen wurde und als Quelle „Live Video“ gewählt wurde. Dies ist nicht weiter schlimm. Man schliesst einfach die Kamera an und wählt wieder diese Quelle. Jetzt funktioniert es.

5.4 Bedienungsanleitung

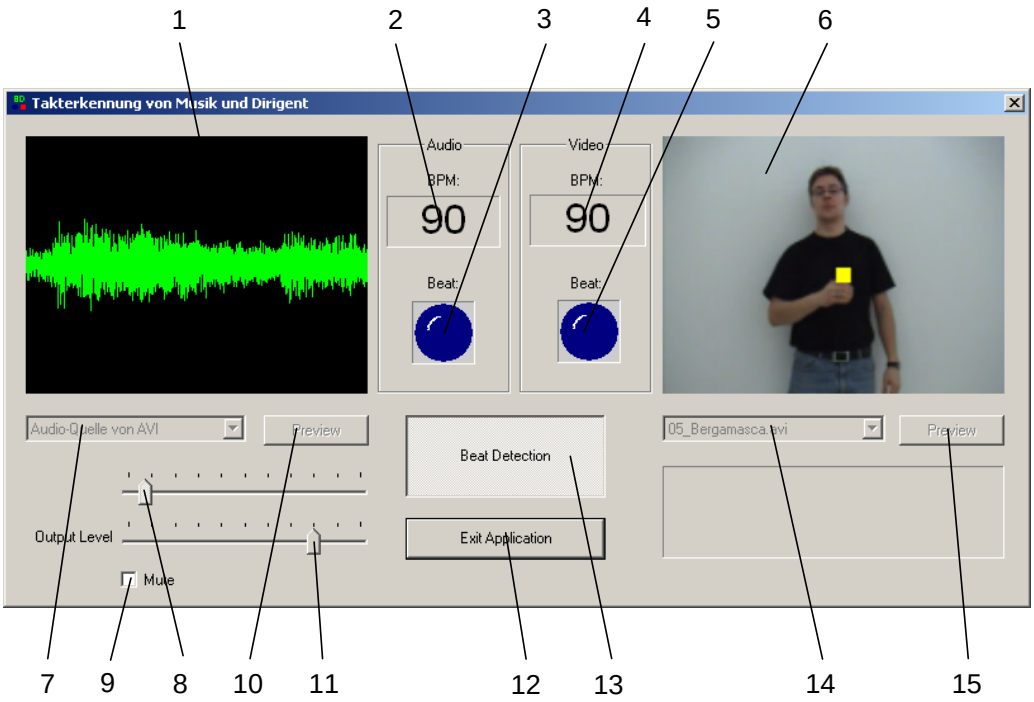
Die Bedienung der Applikation (siehe Abbildung 33) ist sehr einfach und eigentlich selbsterklärend. Trotzdem seien hier die wichtigsten Funktionen und einige Besonderheiten kurz erklärt. Mehr Platz wird die Erläuterung der verschiedenen Properties einnehmen.

Beat Detection mit Video-Datei Wählen Sie „Video-Datei laden“ aus (14). Es wird automatisch „Audio-Quelle von AVI“ gewählt. Mit „Preview“ (10, 15) kann das Video angesehen und das Stück angehört werden, mit „Beat Detection“ (13) startet der Algorithmus. Wollen Sie vor dem Ende des Files stoppen, drücken Sie „Beat Detection“ (13) erneut.

Format Video: unkomprimierte AVI-Datei mit 160 x 120 Pixel Auflösung.
Format Audio: 44.1 kHz, 16 bit, Stereo.

Beat Detection mit Live-Video Die Audio-Quellen „CD“, „Mic“ und „Line-In“ sowie eine Audio-Datei funktionieren nur im Zusammenhang mit einer angeschlossenen Kamera. Wählen Sie dazu „Live-Video“ (14) und danach die von Ihnen gewünschte Audio-Quelle (7). Weiteres vorgehen wie oben.

Format der Audio-Datei: 44.1 kHz, 16 bit, Stereo.



1 Waveform-Anzeige
2 BPM Audio / Property Page
3 Beat-Anzeige Audio / Beat setzen
4 BPM Video / Property Page
5 Beat-Anzeige Video
6 Video-Anzeige
7 Audio-Quelle wählen
8 Einlese-Lautstärke
9 Ton aus
10 Vorschau
11 Ausgabe-Lautstärke
12 Applikation beenden
13 Start / Stop Beat Detection
14 Video-Quelle wählen
15 Vorschau

Abbildung 33: Das Benutzer-Interface mit seinen Anzeigen und Bedienelementen

Beat setzten Im Audio-Teil kann ein Beat gesetzt werden, falls der Algorithmus von sich aus gar keinen Beat oder den Offbeat findet. Klicken Sie dazu mit der linken Maustaste auf das linke Takt-Blinklicht (3).

Lautstärke einstellen Die Einlese-Lautstärke für den Algorithmus lässt sich im Live-Modus mit dem oberen der beiden Regler (8) einstellen. Die Output-Lautstärke ist in beiden Modi am Output Level-Regler einstellbar (11). Ausserdem lässt sich der Ton gänzlich abschalten. Klicken sie hierfür in das Kästchen „Mute“ (9).

Property einstellen Bei beiden Teilen lassen sich verschiedene Properties einstellen. Klicken Sie dazu in die BPM-Anzeige des betreffenden Teils. Es erscheint eine Anzeige wie in Abbildung 34 dargestellt. Wählen Sie aus (1) ein Property und stellen Sie es auf den von Ihnen gewünschten Wert ein. Entweder indem Sie den Regler (2) mit der Maus betätigen, oder indem Sie einen Wert in des Feld (5) eingeben und diesen mit „Set“ (6) bestätigen.

Wollen Sie ein Property überwachen, so wählen Sie dieses aus und klicken Sie in das Feld „Observe“ (3). Der aktuelle Wert wird an der Stelle (4) angezeigt.

5.5 Properties

Alle verwendeten Algorithmen sind so eingestellt, dass sie mit dem verwendeten Material gut funktionieren. Will man jedoch die Eigenschaften verändern, neue Dinge austesten oder Fehler suchen, lassen sich die Algorithmen über die Properties beeinflussen.

Es existieren zwei Kategorien von Properties: Diejenigen, die eine Eigenschaft eines Algorithmus verändern und die, die nur zur Weiterleitung eines Wertes von einem Algorithmus zu ändern sind. Bei der ersten Kategorie macht es Sinn mit den Werten zu spielen, die zweite Kategorie dient für den Benützer nur zur Überwachung der internen Abläufe.

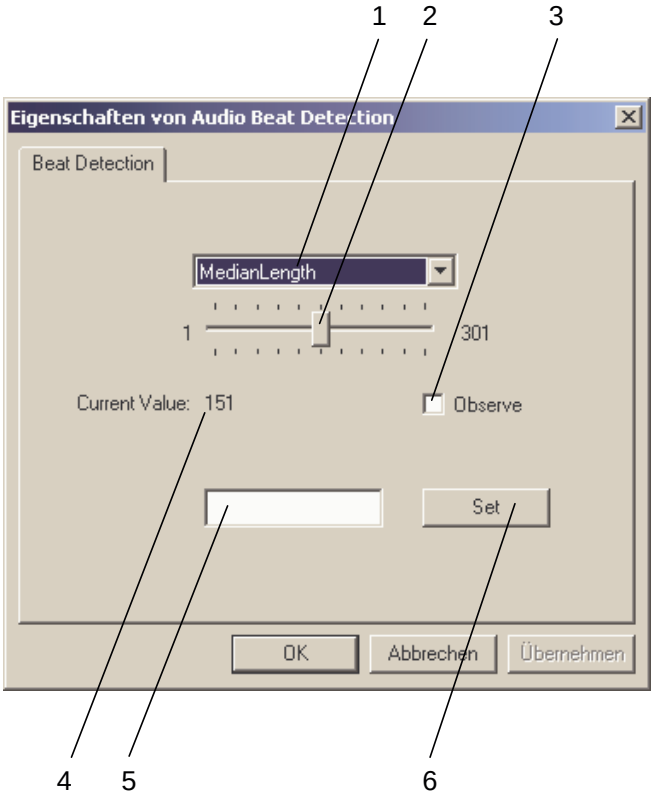
Die Bedeutung der Porperties wird nun im Folgenden beschrieben. Wir unterscheiden nach den beiden Beat Detection Algorithmen.

5.5.1 Audio Beat Detection: Properties, die den Algorithmus beeinflussen

BeatSetTolerance Maximale Distanz zwischen dem nächsten berechneten Beat und dem zu prüfenden Peak, damit ein Peak als Beat gesetzt wird.
(Defaultwert: 3000 Samples bei $f_s = 44.1$ kHz)

ClusterTolerance Maximale Abweichung eines Wertes zum Cluster-Mittelwert, wenn er in den Cluster passen soll. Die Verwendung ist im Kapitel 2.1.3 im Abschnitt „Clustering“ erklärt.
(Defaultwert: 5 Samples bei $f_s = 172$ Hz)

Dthr Kleinst mögliche Distanz zum nächsten dominanten Peak.
(Defaultwert: 60)



1 Property wählen
2 Wert einstellen
3 Property überwachen
4 Aktueller Wert
5 Wert eingeben
6 Wert setzten

Abbildung 34: Die Property Page mit ihren Funktionen

EnablePing Akustische Ausgabe der Beats (nur bei AVI-Files als Quelle) sowie Anzeige der Beats als abklingende Schwingung in der Waveform ein- und ausschalten.
(Defaultwert: 0)

JustPeaks Ausgabe ausschliesslich der detektierten Peaks ein- und ausschalten.
(Defaultwert: 0)

LogThres Werte unterhalb dieser Schwelle werden als Noise interpretiert und nicht für die Beat Detection berücksichtigt.
(Defaultwert: -2)

MatchingPeakTolerance Maximale Distanz eines Peaks zu der in der Funktion „NewStart“ berechneten Stelle, damit ein Peak als an der richtigen Stelle liegend betrachtet wird.
(Defaultwert: 3000 Samples bei $f_s = 44.1$ kHz)

MaxThres Zu überschreitende Schwelle, damit ein Wert beim nächsten Nulldurchgang des Graphen als Peak detektiert wird. Näheres im Kapitel 2.1.2 im Abschnitt „Peaks bestimmen (FindPeak)“.
(Defaultwert: 0.1)

MedianLength Länge des Vektors zur Berechnung des Medians in der Funktion „TempoInduction“.
(Defaultwert: 151)

MinIPI Minimal Inter Peak Interval. Mindestdistanz zwischen zwei Peaks.
(Defaultwert: 16 Samples bei $f_s = 100$ Hz)

NumOfMatchingPeaks Anzahl verwandter vergangener Peaks, damit ein in „NewStart“ zu prüfender Peak als gut befunden wird. Die Verwendung ist im Kapitel 2.1.2 im Abschnitt „Beats bestimmen (FindBeat)“ erklärt.
(Defaultwert: 3)

sthr Verhältnis zwischen der Höhe eines dominanten Peaks und seinen angrenzenden Tälern.
(Defaultwert: 0.1)

TauStep Schrittweite in Samples von Tau bei der Funktion „CombFilter“.
(Defaultwert: 1)

5.5.2 Audio Beat Detection: Properties, die zur Überwachung dienen

AdviseBeat Zeigt die Zeit des zuletzt von Hand gesetzten Beats an.

InterOnsetInterval Zeigt das aktuelle Inter Onset Interval an.

NewBeat Zeigt die Zeit des zuletzt vom Algorithmus gesetzten Beats an.

5.5.3 Video Beat Detection: Properties, die den Algorithmus beeinflussen

BinaryThres Stellt die Schwelle der Graustufe zum Binarisieren des Bildes ein. Die Verwendung ist im Kapitel 2.2.2 im Abschnitt „Schwerpunkt finden“ erklärt.
(Defaultwert: 229)

KernelSize Definiert die Kernel-Grösse für das Blur-Filter. Die Verwendung ist im Kapitel 2.2.2 im Abschnitt „Differenzbild aufbereiten“ erklärt.
(Defaultwert: 11)

NormThres Stellt die Schwelle ein, die der Maximalwert des Bildes überschreiten muss, damit es normalisiert wird. Die Verwendung ist im Kapitel 2.2.2 im Abschnitt „Differenzbild erstellen“ und „Differenzbild afbereiten“ erklärt.
(Defaultwert: 40)

ShowPoint Schaltet die Anzeige der ermittelten Koordinate der Hand ein und aus.
(Defaultwert: 1)

SwapChannels Schaltet die Debug-Ansicht ein und aus. Im Takt des Dirigierens werden die Farbkanäle vertauscht. Auf einem Kanal ist das Grauwertbild, auf dem zweiten das Bild nach dem Blur-Filter und auf dem letzten das binarisierte Bild.
(Defaultwert: 0)

dthr Distanz zum nächst höheren Peak und die Anzahl Frames bis ein Peak als dominant betrachtet wird. Die Verwendung ist im Kapitel 2.2.2 im Abschnitt „FindDominantPeaks“ erklärt.
(Defaultwert: 10)

sthr Verhältnis der Abstiegtiefe zum Peak. Die Verwendung ist im Kapitel 2.2.2 im Abschnitt „FindDominantPeaks“ erklärt.
(Defaultwert: 0.1)

5.5.4 Video Beat Detection: Properties, die zur Überwachung dienen

BPM Zeigt das aktuelle BPM an.

New Dominant Zeigt die Zeit des aktuellen dominanten Peaks an.

New Peak Zeigt den Zeitpunkt des zuletzt vorausgesagten Peaks an.

6 Ergebnis und Schlusswort

6.1 Resultat der Diplomarbeit

Während dieser Diplomarbeit ist also ein Demoprogramm entstanden, welches den Benützer auf unterhaltsame Weise an die Problematik der akustischen und optischen Beat Detection heran bringt. Mit einer an einen Computer angeschlossenen Digitalkamera und einer beliebigen Audio-Quelle lassen sich der Beat eines Musikstücks sowie die Bewegungen einer Dirigentin oder eines Dirigenten detektieren.

Über die genauen Erfolgserlebnisse und erlittenen Fehlschläge gibt das Kapitel „Systemtests“ Auskunft. Ganz allgemein kann man aber sagen, dass das Ergebnis den Erwartungen Stand hält. Der Videoteil funktioniert sehr gut, wenn ein paar grundsätzliche Dirigier-Regeln eingehalten werden. Im Audiobereich sind mehr Einschränkungen nötig, was aber nicht anders zu erwarten war. Das Ziel war bekanntlich eine möglichst gute Takterkennung bei Musikstücken verschiedenster Stilrichtungen, und nicht etwa eine perfekte Erkennung bei einer bestimmten Musiksparte. Da Musik unendlich vielfältig ist, scheint dieser Ansatz nicht nur herausfordernder, sondern auch interessanter als die Festlegung auf ein kleines musikalisches Teilgebiet.

6.2 Verbesserungsvorschläge

Während der Erprobung unseres Systems sind uns Mängel aufgefallen, die sich noch beheben liessen, was wir aber vor allem aus Zeitgründen sein lassen mussten. Vor allem in der Audio Beat Detection gibt es Verbesserungsvorschläge, die vielleicht in einer späteren, weiterführenden Arbeit berücksichtigt werden:

Kalman-Filter Auch im Audio-Teil, genau gesagt in der Funktion „FindBeat“, könnte ein Kalman-Filter eventuell bessere Ergebnisse erzielen. Die Voraussage des nächsten Beats geschieht in unserer Version rein auf Grund des Inter Onset Intervals, das vom Algorithmus „Klapuri“ geliefert wird. Dieses kann zu wenig schnell dem aktuellen Tempo folgen. Ein richtig eingestelltes Kalman-Filter würde die Vorhersage präziser machen, indem auch andere Faktoren berücksichtigt würden.

Tempo Induction Die Tempo Induction wechselt bei einigen Musikstücken zwischen dem Tempo des Viertels und demjenigen des Achtels oder der halben Note hin und her. Dies ist eigentlich nicht schlimm, widerspiegelt es schliesslich den Stil der gerade analysierten Partie eines Stücks. Es ist aber eine kleine Unschönheit, die sich vermeiden liesse. Vielfache und Bruchteile des zuvor detektierten Tempos sollten als das selbe Tempo angesehen werden, und dieses sollte in diesem Fall nicht angepasst, respektive verändert werden.

Im Videoteil haben wir einen Verbesserungsvorschlag:

Mehrdimensionale Bewegungsdetektion Beim Dirigieren werden nur die Bewegungen in vertikaler Richtung berücksichtigt. Eigentlich sind aber die Koordinaten der Hand vorhanden, was eine Berechnung mit vertikaler und horizontaler Bewegungsänderung

möglich macht. Dies ergäbe weniger Einschränkungen und somit weniger Ansprüche an die dirigierende Person.

Bei der Applikation gäbe es natürlich sehr viele Möglichkeiten zur Verbesserung oder Ergänzung unserer Arbeit:

AVI-File aufnehmen Ein Vorteil für die Archivierung von Testläufen wäre, wenn man direkt mit unserer Applikation ein AVI-File aufnehmen könnte. Diese Erweiterung liesse sich recht einfach machen, da die Programm-Struktur schon vorhanden ist.

Auswahl von Kamera und Soundkarte Momentan wählt der Computer selbständig die erste erreichbare Kamera und Soundkarte. Sind mehrere davon angeschlossen, sollte aus den angebotenen Modellen das gewünschte ausgewählt werden können.

Namensverwirrungen Die Eingänge der Soundkarte werden nach deren Namen ausgewählt. Leider ist die Namensgebung von Soundkarte zu Soundkarte verschieden (z.B. Mikrofon statt Microphone). Darum ist es nicht möglich auf allen Systemen automatisch den richtigen Kanal zu wählen.

Weitere Auswertungen Der Zahl der weiteren Möglichkeiten der Darstellung sind keine Grenzen gesetzt: So könnten der Tempoverlauf, Temposchwankungen, Übereinstimmungen Audio/Video und vieles mehr ermittelt und dargestellt werden. Auch könnte man eine animierte Figur im Takt tanzen oder dirigieren lassen et cetera.

6.3 Schlusswort und Dank

Zum Schluss dieses Berichts blicken wir auf eine ereignisreiche Diplomarbeitszeit zurück. Wir konnten die gestellten Aufgaben erfüllen und erreichten somit unser Ziel. Darüber hinaus durften wir aber auch innerhalb eines hervorragend ausgestatteten Labors mit äusserst kompetentem und hilfsbereitem Personal einen lehrreichen Abschluss unseres Studiums erleben.

Abschliessend gebührt den folgenden Personen unser herzlichster Dank:

- Ein grosser Dank gilt unserem Betreuer Franz Bachmann, der uns mit seiner kompetenten, unkomplizierten und angenehmen Art beistand.
- Auch Peter Aeschimann unterstützte uns bei Fragen rund um C++.
- Ivo Oesch liess sich im vorbeigehen viele Male aufhalten und mit einem Problem konfrontieren, für welches er meistens eine Antwort wusste.
- Schliesslich danken wir noch Flöru für das Aufsetzten der PCs und die Matlab-Tips, Chrigu für die Verpflegung, Phöbs fürs heisse Wasser und allen Laborkollegen für die nötige Ablenkung.

Literatur

[1] W. Andrew Schloss: On the Automatic Transcription of Percussive Music – From Acoustic Signal to High-Level Analysis

[2] Simon Dixon: A Beat Tracking System for Audio Signals;
http://www.ai.univie.ac.at/∼simon/papers/oefai-tr-2000-06.pdf

[3] Simon Dixon: Automatic Extraction of Tempo and Beat from Expressive Performances;
http://www.ai.univie.ac.at/∼simon/pub/2001/jnmr.pdf

[4] Anssi P. Klapuri: Musical Meter Estimation and Musical Transcription;
http://www.cs.tut.fi/sgn/arg/klap/cambridge.pdf

[5] Greg Welch and Gary Bishop: An Introduction to the Kalman Filter;
http://www.cs.unc.edu/∼welch/media/pdf/kalman_intro.pdf

[6] Peter S. Maybeck: Stochastic Models, Estimation, and Control, Vol. 1;
http://www.cs.unc.edu/∼welch/media/pdf/maybeck_ch1.pdf

[7] Peter Aeschimann, Franz Bachmann, Florian Kühni, Hans-Christof Maier:
Prisma Report 2: Analysis of Musical Signals;
http://www.prisma-music.ch/downloads/prisma_report2.pdf

[8] Mark D. Pesce: Programming Microsoft Directshow for Digital Video and Television;
Microsoft Press; ISBN 0-7356-1821-6;

A Aufnahme einer Video-Datei

Natürlich gibt es eine Vielzahl von Programmen, die eine Aufnahme von Videos ermöglichen. Da uns diese High-end Software aber nicht zur Verfügung stand, haben wir uns mit der Freeware „VirtualDub“ (www.virtualdub.org) beholfen. Die verwendete Version ist 1.5.10. Hier eine kleine Anleitung:

1. Audio-Datei, mit der das Video aufgenommen werden soll, im Media-Player laden.
2. Im Windows Aufnahme-Mixer (Bild 35) die Quelle **WAV-Out** wählen und den Level so einstellen, dass die Aufnahme nicht übersteuert wird. (Dazu sind vieleicht ein paar Testaufnahmen nötig.)
3. Kamera anschliessen und einschalten.
4. VirtualDub starten und unter **File** | **Capture AVI**. . . in den Aufnahmemodus versetzen.
5. Unter **Video** | **Format** die Auflösung 160 x 120 Pixel und das Format RGB24 wählen.
6. Unten links im Programfenster (Bild 36) die Audio-Qualität 44.1 kHz, 16bit, stereo und die Framerate 25fps wählen.
7. Wichtig ist, dass unter **Audio** | **Compression** das Format „PCM“ und unter **Video** | **Compression** „Uncompressed RGB“ gewählt ist.
8. Unter **File** | **Set Capture file**. . . die Zieldatei wählen.
9. Nun die Aufnahme mit **Capture** | **Capture video** starten.
10. Die Wiedergabe der Audio-Datei starten.
11. Nun das Stück dirigieren
12. Mit der Esc-Taste die Aufnahme beenden und unter **File** | **Exit capture mode** den Aufnahme-Modus verlassen.
13. Mit **File** | **Open video file**. . . die eben aufgenommene Datei laden. VirtualDub präsentiert sich wie in Bild 37.
14. Nun kann, wenn nötig, das Video noch zugeschnitten werden (unter **Video** | **Select range**).
15. Darauf achten, dass unter **Audio** | **Interleaving**. . . das Interleaving nach jedem Frame eingeschaltet ist (Bild 38). Dies ist nötig, damit die Audio-Darstellung in der Applikation flüssig läuft.
16. Mit **File** | **Save as AVI**. . . das Video in seiner definitiven Form speichern.

Nun kann das fertige Video in der Beat Detection Applikation geladen werden!

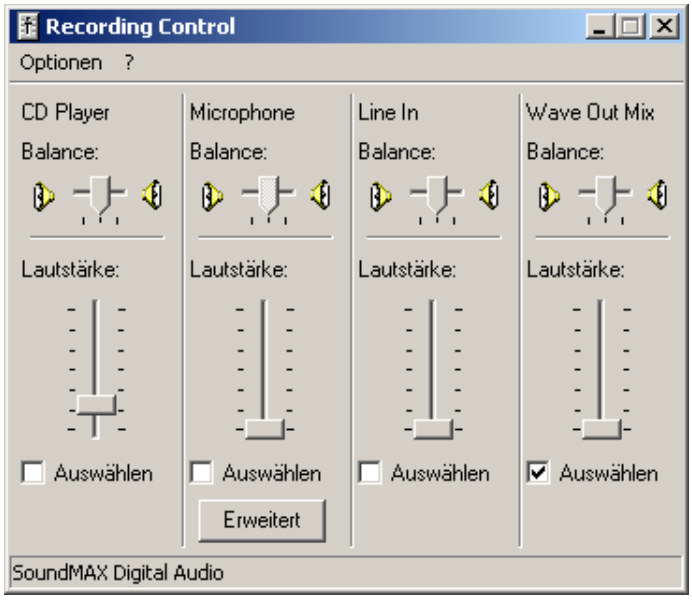


Abbildung 35: Der Windows Aufnahme-Mixer

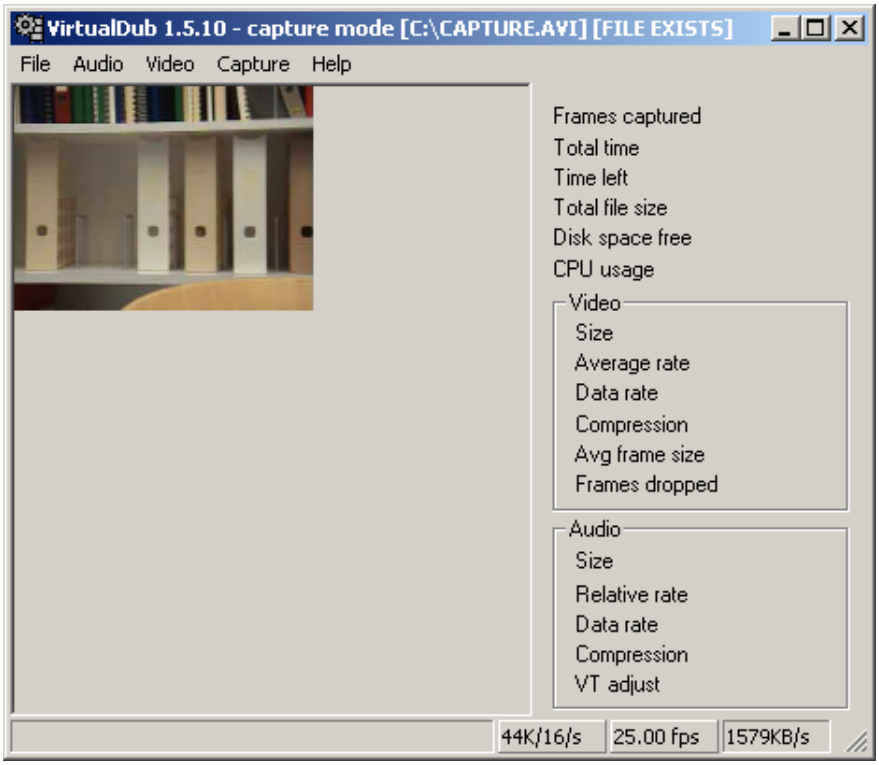


Abbildung 36: VirtualDub im Capture-Modus

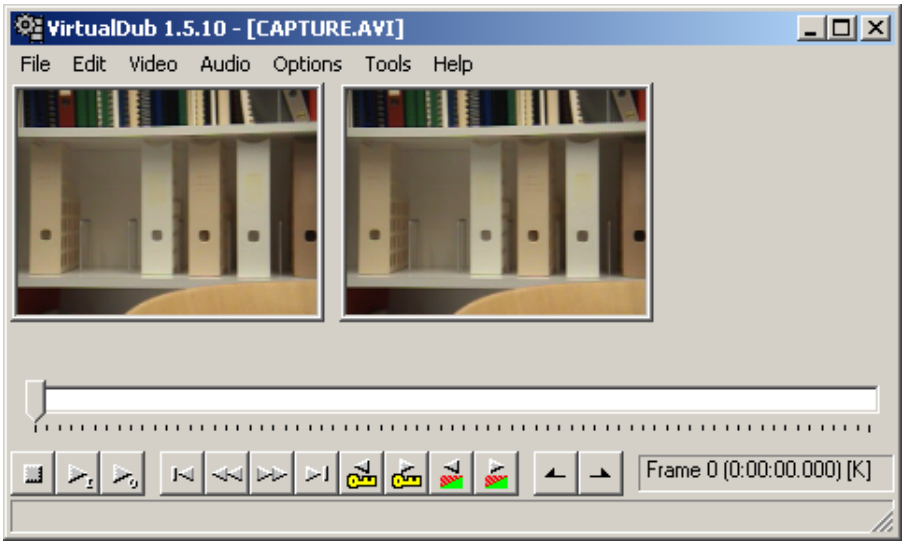


Abbildung 37: VirtualDub im Bearbeitungs-Modus

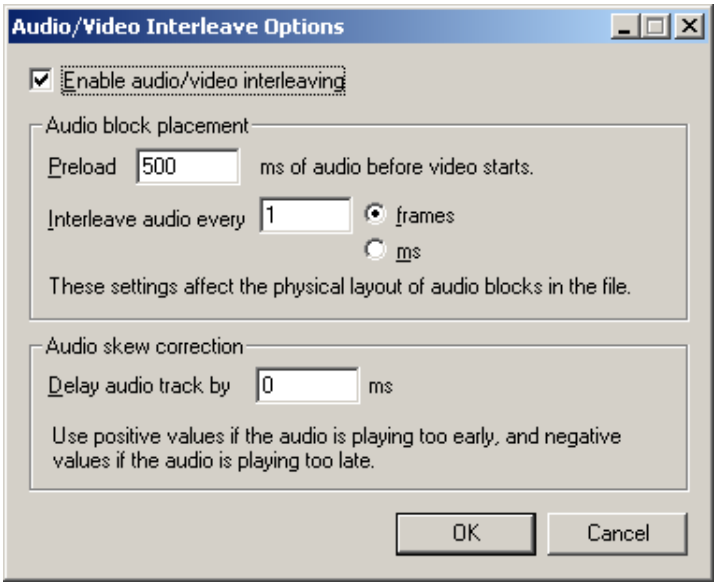
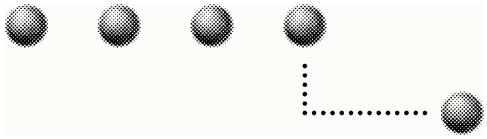


Abbildung 38: VirtualDub: Interleaving einstellen

B Ordnerstruktur DVD

- Development
  - BeatDetection Software
  - LaTex Documentation
  - Matlab Files
    - Audio
    - Video
- Dokumentation
- Install
  - BeatDetection
  - DirectX9 Runtime
  - IPP Runtime
- Papers
- Semesterarbeit
  - Beatdetection Matlab
  - Beatdetection Realtime
    - Prisma
      - Debug
      - res
  - Documentation
  - Songs
- Test Videos
- VirtualDub
  - aviproxy
  - plugins
- WAV Sounds

C Erklärung der Diplomanden



Berner Fachhochschule

Hochschule für Technik und Informatik HTI
Fachbereich Elektro- und Kommunikationstechnik

Erklärung der Diplomandinnen und Diplomanden

Rechte an Diplomarbeiten

Ich nehme zur Kenntnis, dass alle Rechte der durch die HTA Burgdorf geleiteten Arbeiten grundsätzlich bei der Schule bleiben. Besondere Abmachungen zwischen Dozenten und beteiligten Unternehmungen und Institutionen bleiben vorbehalten.

Selbständige Arbeit

Ich bestätige mit meiner Unterschrift, dass ich meine Diplomarbeit selbständig durchgeführt habe. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.), die wesentlich zu meiner Arbeit beigetragen haben, sind in meinem Arbeitsbericht im Anhang vollständig aufgeführt.

Name, Vorname	Bernhard Michael
Datum	Burgdorf, 12. Januar 2004
Unterschrift	.....

Dieses Formular ist dem Diplomarbeitsbericht beizulegen.

Berner Fachhochschule

Hochschule für Technik und Informatik HTI
Fachbereich Elektro- und Kommunikationstechnik

Erklärung der Diplomandinnen und Diplomanden

Rechte an Diplomarbeiten

Ich nehme zur Kenntnis, dass alle Rechte der durch die HTA Burgdorf geleiteten Arbeiten grundsätzlich bei der Schule bleiben. Besondere Abmachungen zwischen Dozenten und beteiligten Unternehmungen und Institutionen bleiben vorbehalten.

Selbständige Arbeit

Ich bestätige mit meiner Unterschrift, dass ich meine Diplomarbeit selbständig durchgeführt habe. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.), die wesentlich zu meiner Arbeit beigetragen haben, sind in meinem Arbeitsbericht im Anhang vollständig aufgeführt.

Name, Vorname	Straubhaar Adrian
Datum	Burgdorf, 12. Januar 2004
Unterschrift	.....

Dieses Formular ist dem Diplomarbeitsbericht beizulegen.

D Aufgabenstellung

Berner Fachhochschule

Hochschule für Technik und Informatik
Fachbereich Elektro- und Kommunikationstechnik
Labor für Signal- und Bildverarbeitung

Diplomarbeit für Michael Bernhard und Adrian Straubhaar

Beat Detection – akustisch und optisch

1 Allgemeines

Eine wichtige Kenngrösse von Musikstücken ist ihr *musikalischer Puls.* Jeder halbwegs begabte Zuhörer ist im Stande, diesen Puls zu erkennen und mit dem Fuss zu klopfen. Kann das ein Computerprogramm auch? Diese Frage nach der automatischen Beat Detection ist nicht nur theoretisch interessant, sondern hat auch praktische Bedeutung, zum Beispiel bei der Suche nach Musikstücken in Datenbanken. Als Resultat Ihrer Semesterarbeit (Sommersemester 2003) liegt eine erste Version eines Programms vor, das den Puls von Musikstücken in Echtzeit bestimmt. In Ihrer Diplomarbeit werden Sie diese Arbeit weiterführen, und zwar in zwei Richtungen: Ein erstes Ziel ist die Perfektionierung Ihres Programms zur akustischen Beat Detection. Zweitens werden Sie den *optischen Puls* einer dirigierenden Person studieren. Der Dirigent einer Musikformation signalisiert mit seinen Bewegungen den Puls und kann damit die verschiedenen Spieler synchronisieren. Beide Teile sollen in einem Demosystem vorgeführt werden können.

2 Aufgaben

2.1 Weiterführung und Perfektionierung der vorhandenen Software zur akustischen Beat Detection.

2.2 Einarbeiten in die Techniken der Videobild-Analyse. Entwicklung und Implementation eines Algorithmus zur Erkennung des optischen Pulses.

2.3 Entwicklung eines Demosystems, welches die Funktionsweise des akustischen und optischen Teils anschaulich zeigt. Den exakten Aufbau des Systems bestimmen Sie selbst; wir stellen uns aber etwa folgendes vor: Der Benützer wählt aus einem Repetoire von Musikstücken eines aus, stellt sich vor die Kamera und beginnt zum laufenden Musikstück zu dirigieren. Das Programm bestimmt in Echtzeit den akustischen und optischen Puls und stellt beide auf dem Bildschirm dar. Das Demosystem soll für die Zuschauer attraktiv sein und soll auch einen gewissen Unterhaltungswert haben.

Bemerkung: Beat Detection ist keine leichte Aufgabe. Es kann deshalb nicht Ihr Ziel sein, *universelle* Algorithmen zu entwickeln, welche für jedes Musikstück bzw. für jede Art von Bewegung vor jedem Hintergrund einwandfrei korrekte Resultate liefern. Eine wichtige Aufgabe für Sie ist es, die Bedingungen klar zu definieren, unter denen Ihr System funktionieren soll.

3 Bericht

Ihr Bericht soll erstens eine Zusammenfassung der theoretischen Grundlagen und eine exakte Beschreibung Ihrer Algorithmen enthalten. Zweitens soll die Leistungsfähigkeit der Algorithmen anhand von zahlreichen Beispielen illustriert werden. Schliesslich verlangen wir eine sorgfältige Dokumentation der Software, sowie eine Bedienungsanleitung für das Demosystem.

4 Formalitäten

Arbeitsort	Labor für Bildverarbeitung
Betreuer	Dr. F. Bachmann und P. Aeschimann
Experte	H.J. Klock
Beginn der Arbeit	Montag, 20. Oktober 2003
Abgabe des Berichts	Montag, 12. Januar 2004, 17.00 Uhr

Burgdorf, September 2003

Franz Bachmann

Peter Aeschimann

E Semesterarbeit

Beat Detection

HTA Burgdorf

Studenten: Michael Bernhard und Adrian Straubhaar
Betreuer: Dr. Franz Bachmann und Peter Aeschimann

Juli 2003

Übersichtsverzeichnis

Inhaltsverzeichnis

1 Einleitung

Wo ist der Beat? Oder anders gesagt: Wo sind bei einem Musikstück die Schläge oder Taktgewichte? Für die meisten Menschen ist die Bestimmung des Takts in der Regel kein Problem. Im Gegenteil, viele wippen ganz automatisch mit dem Fuss oder mit dem Kopf, wenn nebenher irgendwo eine Melodie aus einem Lautsprecher ertönt.

Für einen Computer ist das aber gar nicht selbstverständlich, denn ihm fehlt das musikalische Verständnis. So war es das Ziel dieser Semesterarbeit, mittels eines Programms dieses musikalische Verständnis soweit zu simulieren, dass ein Computer in der Lage ist, die Beats eines Musikstückes selbständig zu bestimmen.

Dies ist auch gelungen. In *Matlab* ist ein Programm realisiert worden, welches die Onbeats verschiedener Musikstücke von Klassik bis Pop bestimmen kann. Eine Umsetzung des Algorithmus in ein Real Time fähiges Programm sowie das Ausmerzen zweier Probleme werden in die anschliessende Diplomarbeit integriert.

Burgdorf, Juli 2003

Michael Bernhard Adrian Straubhaar

2 Onset Detection

Um den Takt eines Musikstückes erkennen zu können, müssen zuerst genügend aussagekräftige Daten zur Verfügung stehen. Diese Daten bestehen aus den Notenanfängen, sogenannten Onsets, der Musik. Das Ziel der Onset Detection ist, möglichst viele aussagekräftige Onsets herauszufinden. In unserer Detektion wird das folgendermassen ausgeführt:

1. Enveloppe berechnen und Samplingrate reduzieren
2. Steigungen an der Enveloppe berechnen
3. Peaks in der Steigung suchen

In den folgenden Unterkapitel werden die einzelnen Arbeitschritte nun detailliert beschrieben.

2.1 Enveloppe berechnen

Die Samplingrate von 11025Hz, wie sie bei unseren Teststücken vorkommt, oder auch die CD-Qualität mit 44100Hz liefern recht viele Daten, die alle verarbeitet werden müssen. In Anbetracht dessen, dass das System echtzeitfähig werden soll, muss eine Datenreduktion vorgenommen werden. Wie man in verschidener Literatur nachlesen kann [1] [4] reicht eigentlich die umhüllende Kurve, die Enveloppe, der Musik. Die Bewegungen, die durch die Onsets verursacht werden, sind auch in der Enveloppe ersichtlich.

Es gibt verschiedene Methoden die Enveloppe zu berechnen:

- Tiefpassfiler
- Mittelwert des gleichgerichteten Signals
- RMS-Wert-Filter
- Maximalwertfilter

Gemäss der Arbeit von Dixon [4] haben wir uns für das RMS-Wert-Filter entschlossen.

Das Musiksignal sei im Vektor $x[n]$ mit der Samplingrate R. Wir berechnen die Amplitude $A[k]$, mit k als einen Integer, der ein vielfaches der Blockgrösse b ausdrückt. Die Samplingrate von $A[k]$ ist um den Faktor b/R kleiner. Die RMS-Amplitude wird über h Blöcke berechnet.

$$A[k] = \sqrt{\frac{1}{hb} \sum_{i=kb}^{(k+h)b-1} x[i]^2}$$

2.2 Steigung berechnen

Grundsätzlich kann man sagen, dass ein starker Impuls eine schnelle und starke Amplitudenänderung zur Folge hat. Das heisst, die Information für mögliche Onsets liegt in der Steigung. Schloss beschreibt in seiner Arbeit [1] eine Möglichkeit die Steigung zu berechnen. Er legt eine Regressionsgerade durch m Punkte der Amplitude $A[k]$ und berechnet die

Steigung $S[k]$. Da für einen Steigungswert m Punkte verwendet werden, wird eine Glättung erreicht, welche die Aussagekraft der Steigungskurve verbessert.

$$S[k] = \frac{m\sum_{i=1}^{m} iA[k+i-m] - (\sum_{i=1}^{m} i)(\sum_{i=1}^{m} A[k+i-m])}{m\sum_{i=1}^{m} A[k+i-m]^2 - \sum_{i=1}^{m} i^2}$$

2.3 Peaks suchen

Nun müssen aus der Steigungskurve die Peaks $P[n]$ bestimmt werden. Dies wird im Moment recht einfach gemacht, indem die lokalen Peaks gesucht werden, die höher als eine Schwelle liegen. Damit durch ein Onset nicht mehrere Peaks entstehen, werden Peaks, die zu schnell aufeinander folgen, gelöscht.

$$\text{Wenn} \quad \text{maxbpm} < \frac{60}{P[n] - P[n-1]} \quad \text{wird } P[n] \text{ gelöscht}$$

Die gefundenen Peaks sind Zeitpunkte für mögliche Onsets. Die nachfolgende Tempo Induction versucht aus den gewonnen Daten eine Regelmässigkeit herauszufinden.

3 Bearbeiten der Onsets

Nach der Detektion einer Anzahl Onsets müssen diese ausgewertet und zu regelmässigen Beats umgewandelt werden. Dies erledigen die Funktionen *findbeat* und *tempoinduction*, welche in den folgenden Unterkapiteln näher betrachtet werden.

3.1 Findbeat

Die Funktion *findbeat* erhält als Eingabewerte die Onset-Zeitpunkte *t_surf_peak* der Funktion *findpeak*. Aus diesen Onsets werden die falschen Beats gestrichen, die korrekten beibehalten und die fehlenden hinzugefügt.

Nachdem aus einer Anzahl Startbeats das Tempo des Stückes ermittelt wurde (*tempoinduction*) wählt das Programm einen Onset aus *t_surf_peak* als Beat aus. Es achtet dabei darauf, ob dieser Beat Vorgänger im richtigen, berechneten Abstand hat. Von diesem Beat ausgehend wird beim nächsten Onset von *t_surf_peak* geprüft, ob der Beatabstand stimmt. Kommt der Onset viel zu früh, wird er gestrichen. Liegt er innerhalb der Toleranz, erfolgt die Prüfung, ob noch andere tolerierbare Onsets auftreten. Ist dies der Fall, wird der bessere Onset akzeptiert, der schlechtere gestrichen. Falls nun nach der Intervallzeit in *t_surf_peak* kein passender Onset erscheint, wird an der betreffenden Stelle ein Beat eingefügt. Da die eingefügten Beats der Tempo Induction zur Erkennung des Tempos übergeben werden, ergibt sich so eine grössere Konstanz, als wenn nur die Onsets als Anhaltspunkte verwendet werden.

Nachdem fünf Beats von selbst eingefügt werden mussten, beginnt das Programm von neuem, indem es aus den letzten Werten von *t_surf_peak* ein neues Beatintervall berechnet und die Startschleife beim nächsten Onset ausführt. So findet das Programm in den meisten Fällen in den Takt zurück.

Algorithmus

```
Tempo mit tempoinduction berechnen
for alle Anfangswerte von t_surf_peak do
  if Wert hat Vorgänger mit Abstand beat_interval then
    Wert ist erster detektierter Beat
  end if
end for
for alle weiteren Werte von t_surf_peak do
  Tempo mit tempoinduction berechnen
  if nächster Wert passt then
    if übernächster Wert passt auch then
      besseren Wert als neuen Beat übernehmen
    else
      Wert als neuen Beat übernehmen
    end if
  else if nächster Wert kommt zu früh then
    Wert löschen
  else if nächster Wert kommt zu spät then
    Beat an passender Stelle einfügen
```

if zu viele Beats von selbst eingefügt **then**
Anfangsprozedur wiederholen
end if
end if
end for

3.2 Tempo Induction

Die Funktion *tempoinduction* berechnet das Tempo des Musikstückes in der Art, wie sie Dixon [5] verwendet. Dabei werden zuerst sämtliche Intervalle zwischen den Onsets, wie sie von der Funktion *findpeak* her kommen, bestimmt und danach ähnliche Intervalle in jeweils den selben Cluster geschrieben. Als nächstes werden ähnliche Cluster zusammengefügt, bevor die eigentliche Selektion des Tempos beginnt.

Falls ein Cluster dem beim vorherigen Beat festgestellten Tempo entspricht, wird dieses beibehalten. Ist dies nicht der Fall, werden die Cluster mit einem Gewicht versehen. Dieses berechnet sich aus der Grösse der Cluster und dem Verwandtschaftsgrad untereinander. Ist also das Tempo eines Clusters ein Vielfaches eines anderen, so bekommen beide ein grösseres Gewicht, als wenn ein Cluster nichts mit einem anderen gemeinsam hat.

Wurde nun ein Cluster auserwählt, muss sein Tempo zwischen *mintempo* und *maxtempo* liegen. Trifft dies nicht zu, kommt der nächst tiefer gewichtete Cluster zum Zug. Ausgegeben wird dann das auserkorene Intervall in Sekunden.

Definitionen

IOI = Intervalle zwischen direkt nebeneinander liegenden und auch weiter benachbarten Onsets (Inter Onset Intervals)
clu{i}.int = Cluster ähnlicher Intervalle
clu{i}.mean = Mittelwert des Clusters i
clu{i}.score = Gewichtung des Clusters i
beat_interval = vom Algorithmus berechnetes, passendstes Intervall
old_beat = zuletzt auserkorenes beat_interval
i, j, k, n = positive Integervariabeln

Algorithmus

for alle IOI **do**
if $|IOI_k - clu\{i\}.mean| < toleranz$ **then**
$clu\{i\}.int = IOI_k \cup clu\{i\}.int$
end if
end for
for alle Cluster **do**
if $|clu\{i\}.mean - clu\{j\}.mean| < toleranz$ **then**
$clu\{i\}.int = clu\{j\}.int \cup clu\{i\}.int$
$clu\{j\}.int\ löschen$
end if
end for
for alle Cluster **do**
if $|clu\{i\}.mean - old\_beat| < toleranz$ **then**

$beat\_interval = clu\{i\}.mean$
$programm\ beenden$
end if
end for
for alle Cluster **do**
if $|clu\{i\}.mean - n \cdot clu\{j\}.mean| < toleranz$ **then**
$clu\{i\}.score = clu\{i\}.score + f(n) \cdot grösse(clu\{j\}.score)$
end if
end for
for alle Cluster **do**
if Cluster i mit grösstem Gewicht liegt zwischen mintempo und maxtempo **then**
$beat\_interval = clu\{i\}.mean$
end if
end for

Der Faktor $f(n)$ berechnet sich wie folgt:

$$f(n) = \begin{cases} 6 - n, & 1 \leq n \leq 4 \\ 1, & 5 \leq n \leq 8 \\ 0, & \text{sonst} \end{cases}$$

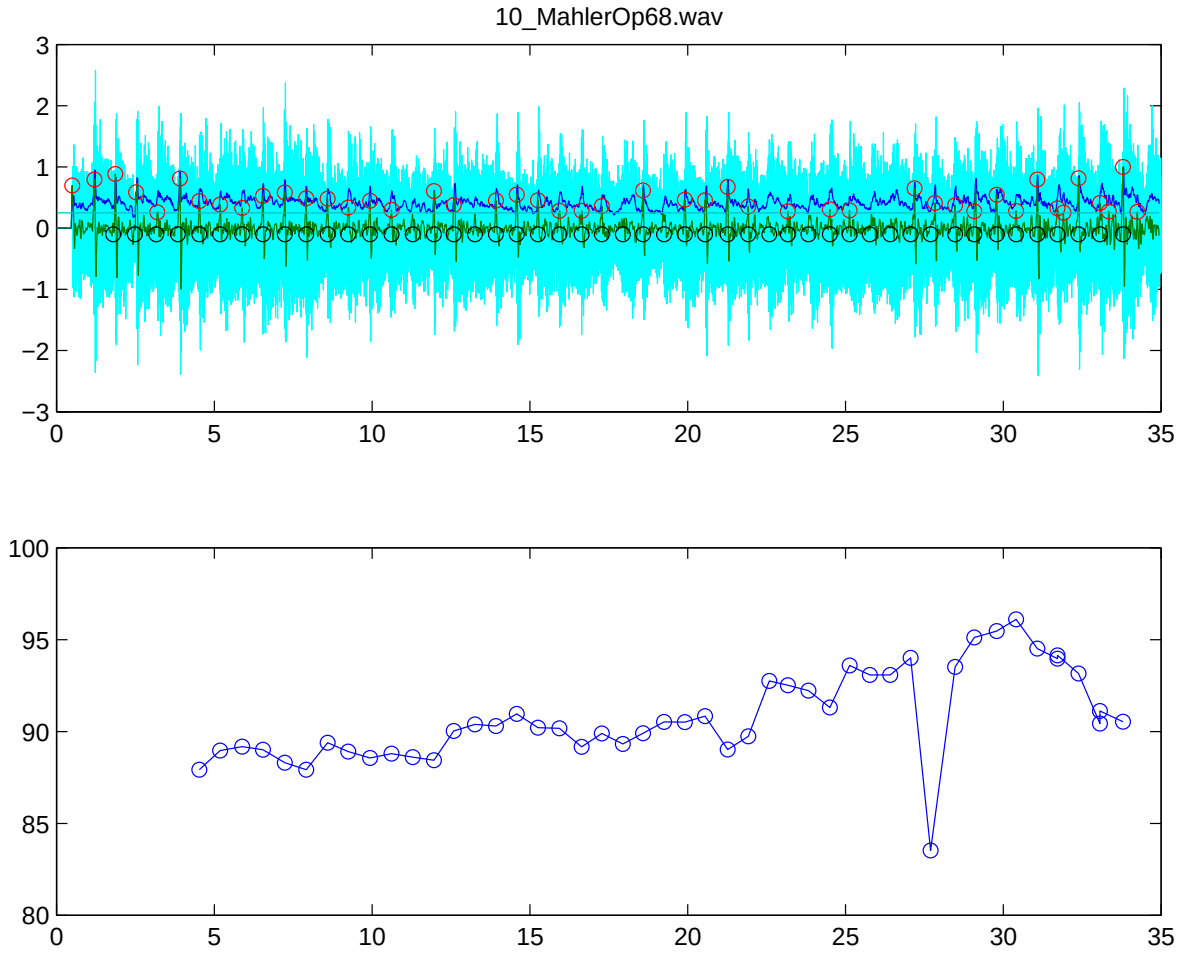


Abbildung 1: oben: Beat Detection; unten: Tempo in bpm

4 Beispiele

Nachfolgend wollen wir die Beat Detection anhand einiger Beispiele betrachten. In Abbildung 1 sehen wir einen Ausschnitt eines sinfonischen Stückes. Lange, ausgehaltene Töne der Streicher erzeugen ein grosses, für den Algorithmus undefinierbares Rauschen, während der Paukist den Takt schlägt, was die Peaks in der türkisfarbenen Wavedatei erzeugt. Diese Peaks werden von der Funktion *findpeak* detektiert und als rote Kringel dargestellt. Die schwarzen Kringel erzeugt dann die Funktion *findbeat*. Zwischen t = 10s und t = 15s werden vereinzelt Beats eingefügt, wohingegen zum Schluss des Ausschnitts Beats gelöscht werden.

Interessant ist auch der Verlauf der berechneten Tempi in der unteren Hälfte der Abbildung. Während 35 Sekunden steigt das Tempo recht deutlich an und sinkt gegen Schluss der Zeit wieder auf den Anfangswert. Der Ausrutscher gegen unten bei t = 27s kommt davon, dass das Programm vorher eine kurze Zeit lang selbst Beats einfügen musste und an der betreffenden Stelle wieder einen korrekten Onset findet.

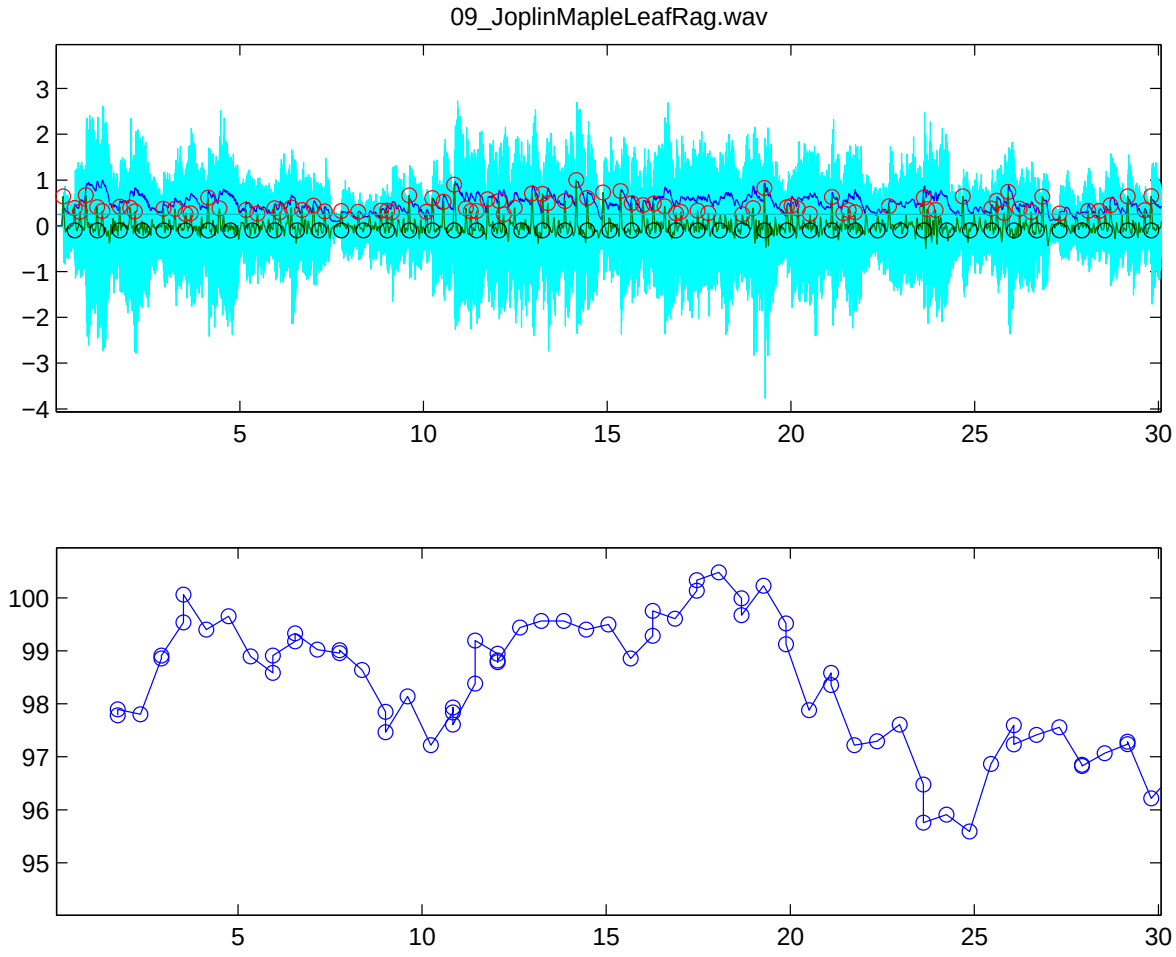


Abbildung 2: oben: Beat Detection; unten: Tempo in bpm

Der *Maple Leaf Rag* ist ein Klavierstück und wird als Rag vor allem synkopisch betont. Steigt das Programm anfangs am richtigen Ort ein, gelingt es trotzdem, den Onbeat zu halten. In Abbildung 2 ist deutlich zu sehen, wie findbeat aus vielen Beats seine Favoriten bestimmen muss. Auch die Temposchwankungen sind gut sichtbar.

Be my Baby steht hier für die rockigen und popigen Stücke mit dominantem Schlagzeug. Die Schwierigkeit bei diesen Stücken ist, dass die Offbeats durch Snare oder Becken des Schlagzeuges oft lauter sind als die Onbeats und darum fälschlicherweise als Beats betrachtet werden. Zu beachten ist auf Abbildung 3 auch das stabile Tempo. Im Gegensatz zu den vorderen Beispielen haben wir hier viel kleinere Temposchwankungen.

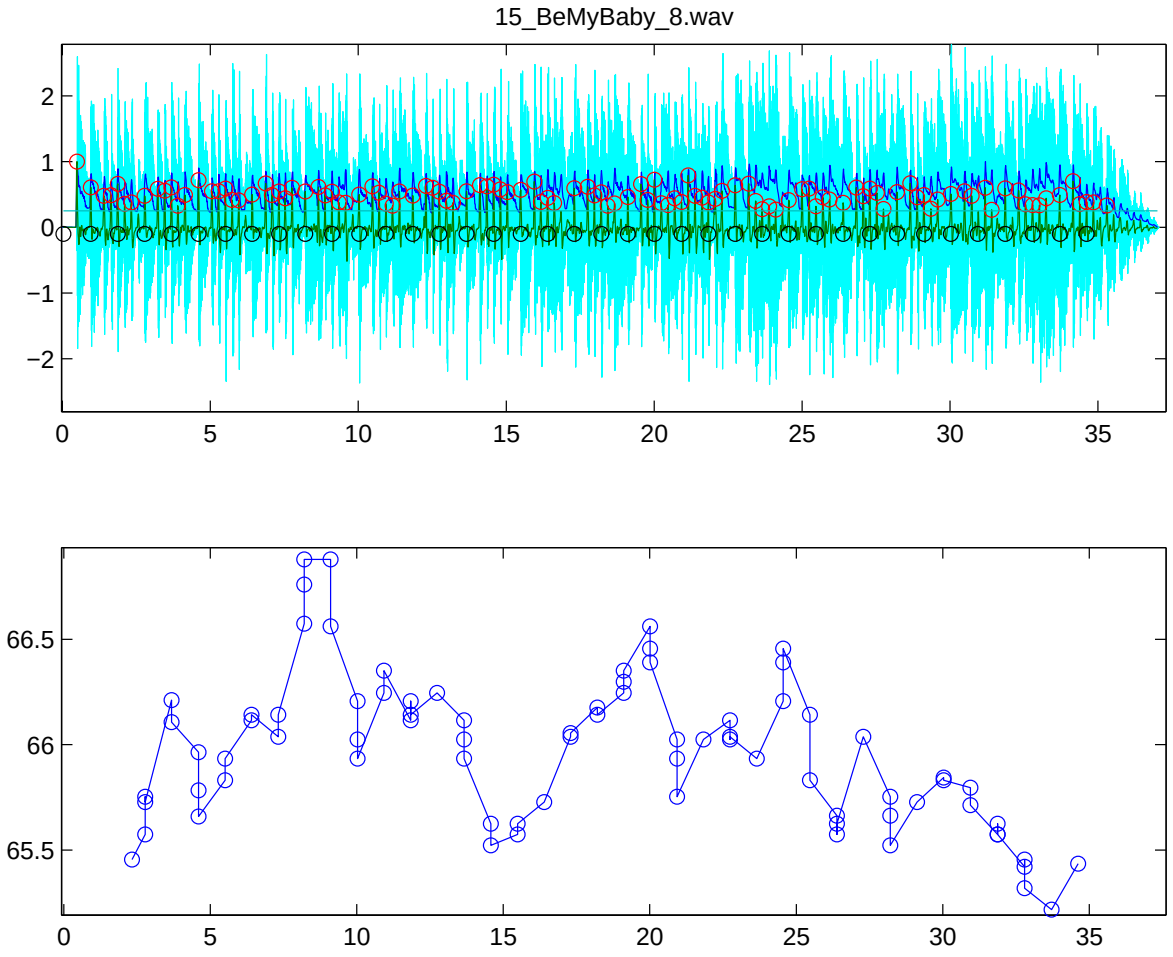


Abbildung 3: oben: Beat Detection; unten: Tempo in bpm

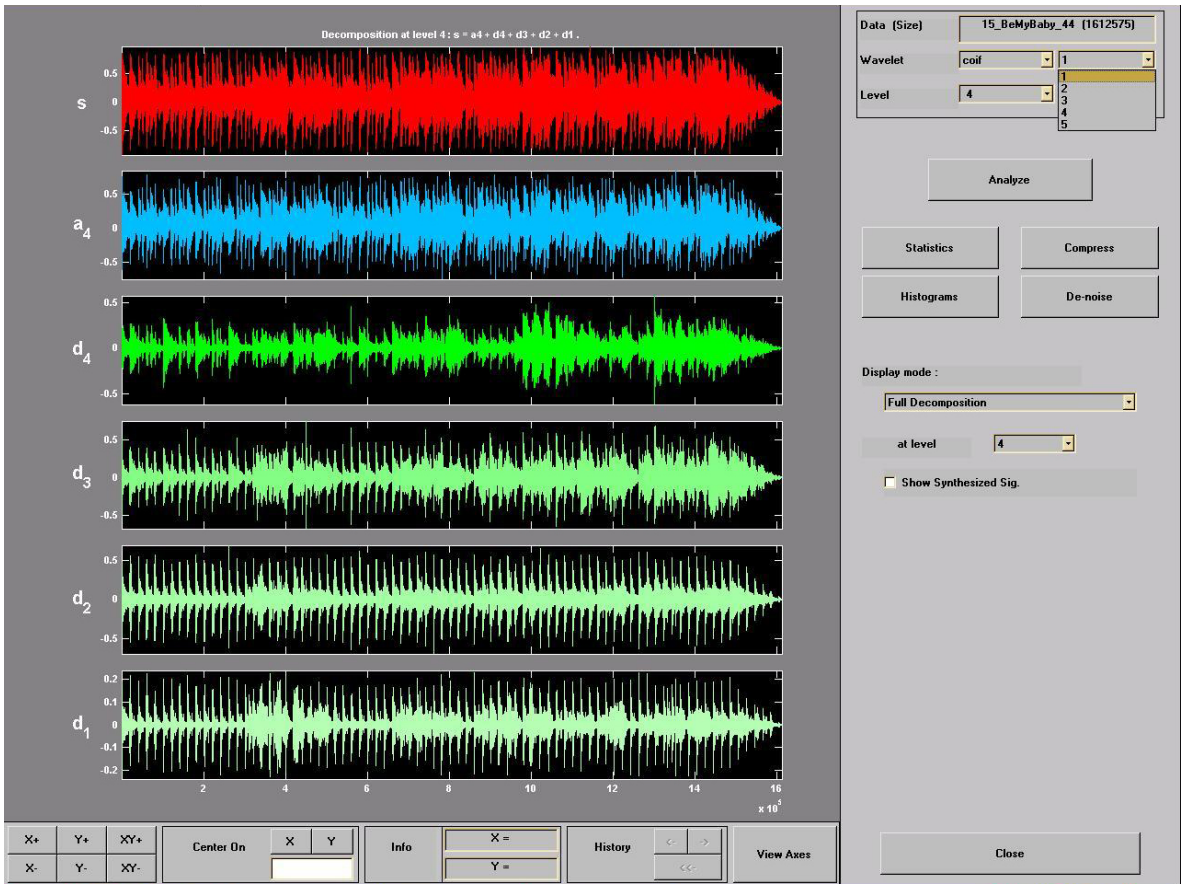


Abbildung 4: Diskrete Wavelet-Transformation

5 Verworfene Ideen

5.1 Wavelets

In der Hoffnung, Peaks und schnell ansteigende Flanken könnten damit besser detektiert werden, untersuchten wir die Wavelets und im speziellen die diskrete Wavelet-Transformation. Dabei stellten wir die Transformation mit verschiedenen Wavelets in ihren verschiedenen Levels dar. Obwohl zum Teil sehr steile und recht ausgeprägte Onset-Flanken entstanden, war die Kurve aber im Allgemeinen nach der Transformation nicht deutlicher als die Hüllkurve, und darum gaben wir die Wavelet-Idee wieder auf. In Abbildung 4 wurde ein Pop-Stück transformiert. Die Flanken sind zwar steil, zur weiteren Verarbeitung eigneten sich die erhaltenen Daten aber nicht besser als ohne Transformation.

5.2 Filter

Um Onbeats und Offbeats besser voneinander unterscheiden zu können, liessen wir die Wave-Datei zuerst filtern, Um die verschiedenen Frequenzbänder unanbhängig von einander zu verarbeiten. Die Idee war die folgende: Wenn bei drei Frequenzbändern ein Onset

an der gleichen Stelle detektiert wird, kann davon ausgegangen werden, dass dies ein Onbeat ist. In der Realität sieht dies aber etwas anders aus. Bei Musikstücken mit Schlagzeug erfolgt häufig auf jeden Taktschlag ein Base-Drum-Schlag. Die scharfen Instrumente wie Snare-Drum oder die Becken werden vor allem auf die Offbeats angeschlagen. Somit hatten wir auf den Offbeats meist mehr detektierte Onsets als auf den Onbeats.

Ein weiterer Nachteil der Filter-Idee erkannten wir in der recht undefinierten Mittellage: In mittleren Frequenzlagen geschieht vergleichsweise wenig rhythmisches, worauf in der Folge entweder fast keine oder unsinnig viele Onsets detektiert wurden.

6 Realtime-Implementation

Das Endziel unserer Arbeit ist, die Beat Detection echtzeitfähig zu machen. Ein Programm soll also, währenddem die Musik gespielt wird, laufend den aktuellen Beat bestimmen.

Unser Betreuer, Peter Aeschimann, hat ein Grundgerüst erstellt, das uns ermöglicht direkt mit der Implementierung der Beat Detection zu beginnen. Das Programm stellt bereits folgende Funktionalitäten zur Verfügung:

- Die Musik kann entweder von einer Wavedatei abgespielt oder direkt vom Eingang der Soundkarte empfangen werden.
- Die verarbeitete Musik kann in einer Datei gespeichert werden.
- Die Musik wird direkt an der Soundkarte ausgegeben.
- Die Applikation wurde mit der MFC-Libary von Microsoft erstellt, was eine relativ einfache Einbindung von Grafikkomponenten erlaubt.

In dieser Semesterarbeit konnte die Real-Time-Version der Beat Detection nicht fertiggestellt werden, da zuerst eine funktionsfähige Version in Matlab realisiert wurde. Die Onset Detection ist jedoch bereits implementiert und auch eine erste Version des Clusterings nach Dixon [5] ist fertiggestellt. Zudem wurde die Grafikausgabe ausgetestet. Abbildung 5 zeigt einen Screenshot von *Let it Be*. Man sieht schön, wie die möglichen Onsets erkannt wurden (kurze senkrechte Striche). Der momentane Clustering-Algorithmus, der zur Tempo Induction verwendet wird, funktioniert jedoch noch nicht genügend stabil.

6.1 Programmaufbau

Jede Funktion, die im Kapitel 2 beschrieben ist, wurde in einem eigenen Thread implementiert. Jeder Thread reicht dem nächsten die Daten in einer Queue weiter. Dazu besitzt jeder Thread einen Ringbuffer, in dem die letzten fünf Sekunden gespeichert werden, die dann für die Anzeige verwendet werden. Abbildung 6 zeigt den schematischen Aufbau des Programms.

Die momentane Version funktioniert gut in Echtzeit. Man erkennt in der RMS-Amplitude die Onsets und hört sie auch gleichzeitig über die Lautsprecher. Das Clustering für die Tempo Induction ist aber noch nicht ausgereift. Der stärkste Cluster bleibt nicht immer auf dem selben Tempo. Das heisst aber nicht, dass das korrekte Tempo nicht in einem Cluster vorhanden ist.

6.2 Zukünftige Erweiterungen

Da nun die Beat Detection in der Matlab-Version funktioniert, kann mit der Echtzeit-Implementierung begonnen werden. Dazu sind einige Änderungen notwendig. Der Clustering-Algorithmus zum Beispiel muss zusätzlich auch Daten von der Funktion verarbeiten, die den Beat sucht. Mit der aktuellen Architektur ist das im Moment nicht möglich.

Zudem muss man sich über die Art der Anzeige Gedanken machen. Grundsätzlich muss die Anzeige sehr resourcensparend sein. Bei der aktuellen Version funktioniert die Anzeige ohne Probleme in Echtzeit. Die Qualität könnte jedoch noch verbessert werden.

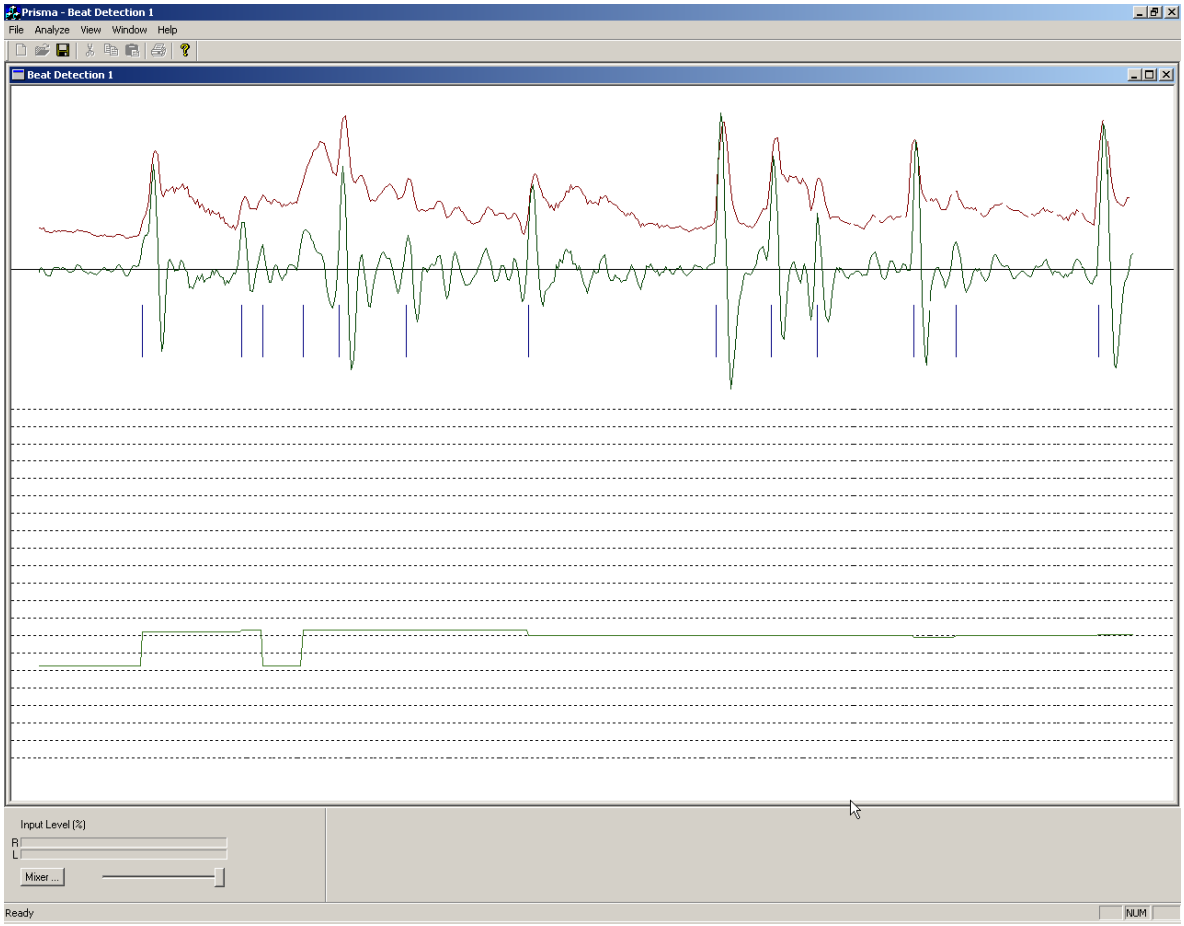


Abbildung 5: Ein 5 Sekunden langer Ausschnitt von *Let it Be*. In der oberen Hälfte sieht man die RMS-Amplitude, die Steigung und die Peaks. In der unteren Hälfte des Bildes ist der stärkste Cluster dargestellt. Jede gestrichelte Linie steht für ein Vielfaches von 10 bpm. Die unterste Linie ist 0 bpm

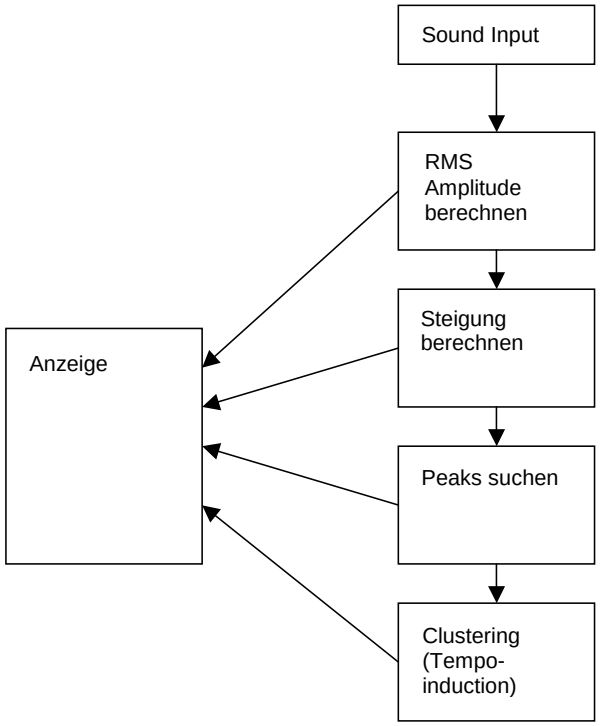


Abbildung 6: Schematischer Aufbau des Programms

7 Funktionstest

7.1 Parameterliste

Nachfolgend erfolgt eine Zusammenfassung aller Musikstücke, wie sie zum Ende der Semesterarbeit detektiert wurden. Die Parameter waren defaultmässig wie folgt eingestellt:

anz_startbeats = 7 bestimmt die Anzahl Onsets, die zur ersten Berechnung des Tempos gebraucht werden.

minfreq = 100 bestimmt den Grad der Glättung für die umhüllende Kurve. *minfreq* gibt ungefähr die Samplefrequenz der Umhüllenden an. Ein zu kleiner Wert ist ungünstig, da sonst die zeitliche Auflösung für die Peaks ungenau wird.

regblock = 4 gibt an, wieviele Werte zum Bestimmen der Steigung verwendet werden. *regblock* hat auch einen gewissen glättenden Effekt bei der Steigung, beeinflusst aber keine Samplerate. Ein zu grosser Wert ergibt eine zusätzliche Verzögerung und schnelle Änderungen kann die Steigung nicht mitmachen.

maxthers = 0.25 definiert die Schwelle für die Peakdetektion.

maxbpm = 800 verhindert, dass zwei Peaks zu schnell nacheinander auftreten. *maxbpm* gibt eigentlich das schnellste Tempo an, welches in der Onset Detection erlaubt ist.

maxtempo = 150 gibt das maximale Tempo an, welches bei der Beat Detection in *findbeat* erlaubt ist.

mintempo = 50 gilt analog dazu als das tiefste erlaubte Tempo.

toleranz = 0.06 wird an verschiedenen Stellen im Algorithmus benötigt, um kleine Abweichungen zu tolerieren. Angegeben wird der Wert in Sekunden.

7.2 Resultate des Funktionstestes

HaendelConcertoGrosso Funktioniert ohne Probleme.

PurcellDidoundAeneas Findet zu wenig Punkte; kein gutes Resultat möglich.

FraenzlisMarcha Gut; findet Viertelnote.

BeatlesLetItBe Gut; findet halbe Note.

BeatlesYellSubm Findet halbe Note als Offbeat. Gut bei anz_startbeats = 8.

BlueGrass Hat zu viele Punkte und damit zu viele falsche Intervalle. Funktioniert nur bei starker Eingrenzung des Schlages (mintempo = 100, maxtempo = 120).

HomelandPolka Gut; findet halbe Note.

BulgLied Bei anz_startbeats = 4 und maxthers = 0.3 fällt das Programm am Schluss aus dem Takt. Das subito meno mosso [1] nach einem Drittel des Ausschnitts wird erkannt, das subito piu mosso [2] gegen Ende ist aber zu schwierig.

JoplinMapleLeafRag Gut; allerdings ganz am Schluss des Ausschnitts aus dem Takt.

MahlerOp68 Gut bei maxthers = 0.2.

BachFzSuite Findet die punktierte Viertelnote. Keine gute Einstellung der Parameter gefunden.

ChieftainsIrish1 Gut bei anz_startbeats = 9.

Chieftainsirish2 Kurze Unsauberkeit, findet sich aber wieder.

OurHouse Bei Default-Einstellung werden gegen Schluss zu viele Punkte detektiert. Bei maxthers = 0.4 und anz_startbeats = 5 ist das aber kein Problem mehr.

BeMyBaby Gut bei anz_startbeats = 9.

Bergamasca Auch bei maxthers = 0.3 sind im Mittelteil zu wenig Punkte vorhanden. Ansonsten funktioniert es.

Sympathique Gut bei anz_startbeats = 6.

DankeSchoen Gut.

[1] ital. für plötzlich langsamer
[2] ital. für plötzlich schneller

8 Ergebnis und Ausblick

Von den vorgegebenen 18 Musikstücken konnte das Programm bei zweien den Beat gar nicht bestimmen. Alle anderen Stücke konnten je nach Einstellung der Parameter recht gut bis ausgezeichnet detektiert werden. Dabei spielt weniger die Musikrichtung eine Rolle als viel mehr der Stil des Stücks im speziellen. Am meisten Probleme bietet ein Stück mit viel „monotonem Geräusch“ und wenig bis gar keinen klar definierten Beats (z. B. *BlueGrass*). Einfach hingegen sind vor allem moderne Pop- und Rockstücke, aber auch Stücke anderer Stilrichtungen mit gut hörbarem Puls (z. B. *HaendelConcertoGrosso*).

Ein Problem stellt immer noch die Unterscheidung zwischen Onbeat und Offbeat dar. Da die Onbeats nicht zwingend lauter sind als die Offbeats, und da ein Stück auch nicht unbedingt mit einem Onbeat beginnt, besteht die Gefahr, dass sich das Programm für die Offbeats entscheidet. Würden überall die Viertelnoten detektiert, statt die doppelt so langen halben Noten, würde sich dieses Problem nicht zeigen. Nun kann der Computer aber nicht zwischen langsamen und schnellen Stücken unterscheiden, da ihm auch hier das nötige musikalische Verständnis dafür fehlt. Somit betrachtet er manchmal die Viertelnote als Beat, ein andermal die halbe Note.

Ein anderes Problem ist die Menge der Punkte, welche die Funktion *findpeak* der Funktion *findbeat* zur Verfügung stellt. Je nach Lautstärke des Stückes variiert diese, was aber mit einer adaptiven Schwelle anstelle des fix eingestellten Parameters *maxthers* zu beheben wäre. Eine erste Version wurde erprobt, bot aber noch keinen Gewinn für den Algorithmus. Beide Probleme sollen zu Beginn der anschliessenden Diplomarbeit noch einmal untersucht und behoben werden.

Literatur

[1] W. Andrew Schloss: On the Automatic Transcription of Percussive Music – from Acoustic Signal to High-Level Analysis

[2] Simon Dixon: A Lightweight Multi-Agent Musical Beat Tracking System; http://www.ai.univie.ac.at/~simon/pub/pricai2000.ps.gz

[3] Simon Dixon: Real Time Tracking and Visualisation of Musical Expression; http://www.ai.univie.ac.at/~simon/pub/icmai2002.pdf

[4] Simon Dixon: A Beat Tracking System for Audio Signals; http://www.ai.univie.ac.at/~simon/pub/diderot.ps.gz

[5] Simon Dixon: Automatic Extraction of Tempo and Beat from Expressive Performances; http://www.ai.univie.ac.at/~simon/pub/jnmr.pdf