

Grundlagenkurs C++

Übungsblatt 9

Aufgabe 9 *DTMF-Erkennung*

In Aufgabe 8 wurde ein Generator für die Erzeugung von DTMF Tönen implementiert. Bei der Übertragung der DTMF Tönen muss nun der Empfänger das übertragene Audiosignal analysieren und ermitteln ob, und wenn ja, welche Töne jeweils aus der unteren und oberen Frequenzgruppe im Signal enthalten sind. Aus den detektierten Tönen kann dann das übertragene Zeichen rekonstruiert werden.

Das übertragene Audiosignal wird vom Empfänger kontinuierlich analysiert, indem es in eine Abfolge kurzer, sich überlappender Zeitfenster unterteilt wird. Jedes Zeitfenster wird dann getrennt auf das Vorhandensein von DTMF Tönen untersucht. Dabei wird folgende Plausibilitätsprüfung angewandt:

Falls in zwei oder mehr Fenstern ein DTMF Signal gefunden wird, wird ein gültiges DTMF Zeichen angenommen.

Das Audiosignal ist üblicherweise einer Kanalverzerrung sowie weiteren Störungen (z.B. überlagerte Sprache) unterworfen, so dass eine Detektion der DTMF Töne nicht immer korrekt möglich ist. Im Rahmen dieser Übungsaufgabe sollen folgende Plausibilitätsprüfungen bei der Detektion der DTMF Töne berücksichtigt werden:

- **Signal-Rauschabstand:** Die Signalenergie der zwei DTMF Töne muss mindestens einen vorgegebenen Anteil der Gesamtenergie des Signals betragen (Parameter `ThrSigNoise`, typisch 0.75, also 75% der Gesamtenergie)
- **Standard Twist:** Die Energie des Tons aus der unteren Frequenzgruppe darf nicht mehr als einen bestimmten Faktor über der Energie des Tones der oberen Frequenzgruppe liegen (Parameter `ThrTwistStd`, typisch 8dB entsprechend einem Faktor von 6.3)
- **Reverse Twist:** Die Energie des Tons der oberen Frequenzgruppe darf nicht mehr als einen bestimmten Faktor über der Energie des Tones der unteren Frequenzgruppe liegen (Parameter `ThrTwistRev`, typisch 4dB entsprechend einem Faktor von 2.5)

Aufgabe 9-1 *Tondetektion mit Goertzel-Algorithmus (Präsenzaufgabe)*

Um das Signal innerhalb eines Zeitfensters, das N Samples lang ist, auf das Vorhandensein von Tönen einer bestimmten Frequenz zu untersuchen, könnte prinzipiell eine diskrete

Fouriertransformation (DFT, bzw. FFT) des Signals vorgenommen werden. Da jedoch für die vorliegende Aufgabenstellung nur 8 bestimmte Frequenzlinien ausgewertet werden müssen, wird ein anderes Verfahren angewandt, um den Energiegehalt des Signals direkt für die interessierenden Frequenzlinien zu berechnen:

Der modifizierte Goertzel-Algorithmus ist geeignet um einzelne Linien der DFT effizient zu berechnen.

Er kann als Differenzgleichung angegeben werden, die für jeden Zeitschritt einen Wert $s(t)$ aus den vorherigen Werten $s(t-1)$, $s(t-2)$ sowie dem aktuellen Sample $x(t)$ berechnet:

$$s(t) = 2 * \cos(2 * \pi * f / f_s) * s(t-1) - s(t-2) + x(t)$$

Diese Gleichung ist für jeden Zeitpunkt t innerhalb des Zeitfensters auszuwerten.

Wenn alle N Samples des Zeitfensters bearbeitet sind, kann der Energiegehalt der Frequenzlinie dann wie folgt berechnet werden:

$$|X(f)|^2 = s^2(N) + s^2(N-1) - 2 * \cos(2 * \pi * f / f_s) * s(N) * s(N-1)$$

Schreiben Sie eine Klasse `CGoertzelDetector`, die die Differenzgleichung realisiert.

- Mit der Methode
`int iSetFrequencyNDFT(double lfFrequency, double lfSampleRate)`
kann der Detektor jederzeit initialisiert werden.
- Die Methode
`void vRestartFilter()`
setzt die internen Speicher für die Werte $y(t-1)$ und $y(t-2)$ zurück.
- Mit der Methode
`void vProcessSamples(double *plfSampleBuffer, int iNofSamples)`
wird für alle `iNofSamples` Samples im Buffer `plfSampleBuffer` die Differenzgleichung berechnet.
- Nachdem N Samples bearbeitet sind, wird mit der Methode
`double lfGetSquaredMagnitude(void)`
aus den Membervariablen des Detektors der Energiegehalt des Signals berechnet.
- Die Methode
`double lfDetectInWindow(double *plfSampleBuffer, int iNofSamples)`
berechnet den Energiegehalt innerhalb des übergebenen Signalstückes bzw. Zeitfensters. Dabei werden nacheinander die o.a. Methoden aufgerufen.

Aufgabe 9-2

DTMF-Detector (Präsenzaufgabe)

Zur Detektion der DTMF Zeichen muss jedes Zeitfenster auf das Vorhandensein von DTMF Tönen untersucht werden.

- Schreiben Sie eine Klasse `CDTMFDetector`, die insgesamt 8 Goertzel-Detektoren für die jeweils vier Töne der oberen und der unteren Frequenzgruppe besitzt. Neben den Detektoren verfügt die Klasse über zwei Arrays, die für die untere und obere Frequenzgruppe die berechneten Energiewerte für alle vier Töne der Gruppe speichern.
Die Gesamtenergie des Signals wird ebenfalls berechnet und in einer Membervariable gespeichert.
Die Parameter zur Plausibilitätsprüfung (Twist und Signal-Rauschabstand) sind ebenfalls Member der Klasse.
Die Frequenzgruppen und die DTMF Zeichentabelle sind bereits in der Klasse `CDTMFMapper` aus Aufgabe 8 definiert.
- Über die Methode
`int iInit(double lfSampleRate, double lfThrTwist, double lfThrTwistRev, double lfThrSigNoise)`
kann der Detektor jederzeit neu initialisiert werden.
- Die Methode
`int iDetectOneWindow(double *plfBuffer, int iNofSamples)`
berechnet die Energien für alle 8 Frequenzlinien, sowie die Gesamtenergie des Signals. **Hinweis:** Die Gesamtenergie wird durch die Summation aller quadrierten Abtastwerte im Zeitfenster berechnet.

Anschließend finden die Plausibilitätsprüfungen statt.

Hinweis: Um die Energie der zwei gefundenen DTMF Linien mit der Gesamtenergie zu vergleichen, muss die Summe der Energie in den beiden Linien berechnet werden. Diese ist dann noch mit dem Faktor $2/N$ zu multiplizieren.

Die Ergebnisse der Plausibilitätsprüfung werden als `int` zurückgeliefert. Die möglichen Werte sind in der Klasse als `enum` definiert:

```
enum eDetectionResult {E_DTMF_OK = 0,
E_REVERSE_TWIST_ERROR, E_STANDARD_TWIST_ERROR,
E_SIG_NOISE_ERROR, E_DTMF_DETECTOR_ERROR};
```

Aufgabe 9-3 *DTMF-Evaluierung (Präsenzaufgabe)*

Um ein DTMF Zeichen sicher zu erkennen, soll dieses mindestens zwei Zeitfenster lang detektiert werden. Im Folgenden soll eine Klasse `CDTMFEvaluator` implementiert werden, die beliebig viele Samples eines Audiosignales stückweise verarbeitet und nach jedem Verarbeitungsschritt zurückmeldet, ob ein gültiges DTMF Zeichen vorliegt oder nicht. Hierzu muss die Klasse den Buffer für die überlappenden Zeitfenster selbst implementieren.

Dazu wird ein Buffer für die gesamte Fensterlänge (Parameter `WindowLength`) alloziert.

Das Überlappingsintervall der Fenster (Parameter `WindowOverlap`) ist einstellbar.

Bei jedem Verarbeitungsschritt werden der Klasse `SignalChunkSize = (WindowLength - WindowOverlap)` neue Werte übergeben.

Bei jedem Verarbeitungsschritt werden die Werte des alten Fensters bis auf die letzten `WindowOverlap` Werte aus dem Buffer herausgeschoben und dann die neuen `SignalChunkSize` Werte hinten an den Buffer angehängt. Somit stehen immer `WindowLength` Werte im Buffer.

Die Prüfung, ob in zwei aufeinanderfolgenden Zeitfenstern ein gültiger DTMF Ton vorliegt, wird ebenfalls von der Klasse durch einen Zustandsautomaten realisiert. Dazu wird eine Zustandsvariable definiert, die einen der folgenden `enum` Werte annehmen kann:

```
enum eDetectorState {E_NO_TONE = 0 ,
E_FIRST_WINDOW_OK,
E_VALID_DTMF,
E_VALID_DTMF_END}
```

- a) Entwerfen Sie einen Zustandsautomaten unter Verwendung der o.a. `enum` Werte. Berücksichtigen Sie folgende Eigenschaften des DTMF Zeichens:

- Mindestens zwei Fenster sind nötig, um ein Zeichen korrekt zu detektieren.
- Die Töne dürfen beliebig lang sein.
- Eine Unterbrechung des Tons von der Länge eines Zeitfensters ist zulässig. Zwei aufeinanderfolgende DTMF Töne müssen durch eine Pause von mindestens zwei Fenstern getrennt sein.

- b) Implementieren Sie den Buffer für die überlappenden Zeitfenster.

- c) Die Initialisierung der Klasse erfolgt über die folgende Methode:

```
int iInit(int iWindowLength, int iWindowOverlap, double
lfSampleRate, double lfThrTwiStd, double lfThrTwiRev, double
lfThrSigNoise)
```

- d) Die fortlaufende Evaluierung des Signals erfolgt mit der Methode

```
int iEvaluate(double* plfSampleBuffer, int iNofSamples);
```

Dabei müssen immer `WindowLength - WindowOverlap` neue Werte übergeben werden. Das Evaluierungsergebnis nach jedem Aufruf wird als `int` zurückgegeben und kann einen der folgenden `enum` Werte annehmen:

```
enum eDTMFResult {E_DTMF_NO_TONE=0, E_DTMF_START, E_DTMF_HOLD,
E_DTMF_STOP};
```

Nach jedem Aufruf kann somit also der Beginn, das Halten und das Ende eines DTMF Zeichens erkannt werden.

Schreiben Sie ein Programm, das ein Audiofile mit DTMF Tönen einliest, welches Sie mit dem Programm aus Aufgabe 8 erzeugt haben. Verwenden Sie dazu die Klasse `CWaveFile` .

Das Signal soll nun in einzelnen Verarbeitungsschritten von der Klasse `CDTMFEvaluator` analysiert werden. Geben Sie die Ergebnisse der Analyse nach jedem Verarbeitungsschritt aus.

Die Klasse `CDTMFEvaluator` ist mit folgenden Werten zu initialisieren:

- Abtaste: 8000 Samples / sec.
- SignalRauschabstand: 0.75 der Gesamtenergie
- StandardTwist: 6.3
- ReverseTwist: 2.5
- Fensterlänge: 160 Samples
- Overlap: 80 Samples

Geben Sie Ihre Lösung zu dieser Aufgabe in einer Datei `9-3.cpp` ab.