beleg

May 4, 2022

1 Wavelettransformation zur Rauschreduktion in Audiosignalen

Dieser Beleg ist eine Zusammenarbeit von Johannes Müller und Willy Müller. Er wurde für die Hochschule Anhalt, Fachbereich 6 bearbeitet und erstellt. Für den erfolgreichen Abschluss des Moduls Signaltheorie sollte ein tangierendes Thema für einen Beleg ausgesucht werden, welches die Kenntnisse der Studierenden in den gelehrten Themen reflektiert.

1.1 Einleitung und Motivation

Wir haben uns daher gefragt: Inwiefern kann die Wavelettransformation als Mittel zur Reduktion von Hintergrundgeräuschen in Audiosignalen verwendet werden? In der Zeitdomäne ist das sogennante *Ducking* eine häufig verwendete Methode und in der Frequenzdomäne wird oft das Spektrum des Störsignals von dem des Eingangssignals abgezogen. Mit Wavelets könnte eine Kombination dieser Effekte erreicht werden.

Um diese Hypothese zu überprüfen, müssen zuerst die Rahmenbedingungen gesetzt werden.

1.2 Analyse der Problemstellung

Es soll ein System erstellt werden, dass ein zeitdiskretes Eingangssignal $x(t)$ entgegennimmt und ein Signal $y(t)$ ausgibt. Die Amplitude der Signale ist über das Interval

$$-1 \leq x \leq 1; -1 \leq y \leq 1$$

begrenzt. Die Diskretisierung in der Zeitachse ist auf die übliche Rate von $\mathsf{sr} = 22050$ Hz festgelegt, so dass Audiofrequenzen bis ca. 10 kHz abgebildet werden können. Die Dauer des Eingangssignals ist in folgeden Tests auf $T = 5$ s festgelegt, wobei das System dieses Signal in noch kleineren zeitlichen Abschnitten bearbeiten wird und somit auch ein zeitlich unbeschränktes Signal verarbeiten kann.

Das Eingangssignal $x(t)$ setzt sich hier beispielsweise aus dem Nutzsignal $source(t)$ und dem weißen Rauschen $r(t)$ zusammen, wobei das Rauschen nur zu bestimmten Segmenten des Nutzsignals addiert wird. Zusätzlich zum Eingangssignal $x(t)$ steht dem System das Referenzsample $r(t)$ zur Verfügung, welches zur Bestimmung des Pegels des Störsignals herangezogen werden kann.

[ ]:
```
# Erzeugung des Störsignals

import numpy
sr = 22050 # sample rate
T = 5.0    # seconds
t = numpy.linspace(0, T, int(T*sr), endpoint=False) # time variable
```

```
r = numpy.random.normal(loc=0, scale=1e-2, size=int(T*sr)) # white noise

from IPython.display import Audio
import librosa
import librosa.display
Audio(r, rate=sr)
```

[ ]: <IPython.lib.display.Audio object>

```
# Erzeugung des Eingangssignals

# Quelle: https://ccrma.stanford.edu/workshops/mir2014/audio/
source, _ = librosa.load('audio/T39-piano.wav', sr=sr, duration=T)
x = source + r * (numpy.cos(t*2) < 0)
Audio(x, rate=sr)
```

[ ]: <IPython.lib.display.Audio object>

1.2.1 Zeitliche Unterteilung

Das Eingangssignal wird nun in kleinere überlappende Abschnitte unterteilt, welche anschließend zur Rauschminderung weiterverarbeitet werden. Die Länge der Abschnitte so gewählt, dass für die minimalste Frequenz des Signals mindestens eine volle Sinuskurve im Abschnitt enthalten ist. Sie darf aber auch nicht zu lang sein, denn in der Echtzeitverarbeitung müssten alle enthaltenen Werte gepuffert werden, was zusätzliche Latenz herbeibringt. Schließlich sollte sie eine volle Zweierpotenz betragen, um die Abschnitte in optimierten Algorithmen, wie folgend FFT- oder Waveletalgorithmen, effizient verarbeitet zu können. Die Länge der Abschnitte ist hier auf 2048 Sample festgelegt, was einem Zeitinterval von

$$\frac{\mathsf{win\_size}}{\mathsf{sr}} = \frac{2048}{22500} = 91.02\ \mathbf{ms}$$

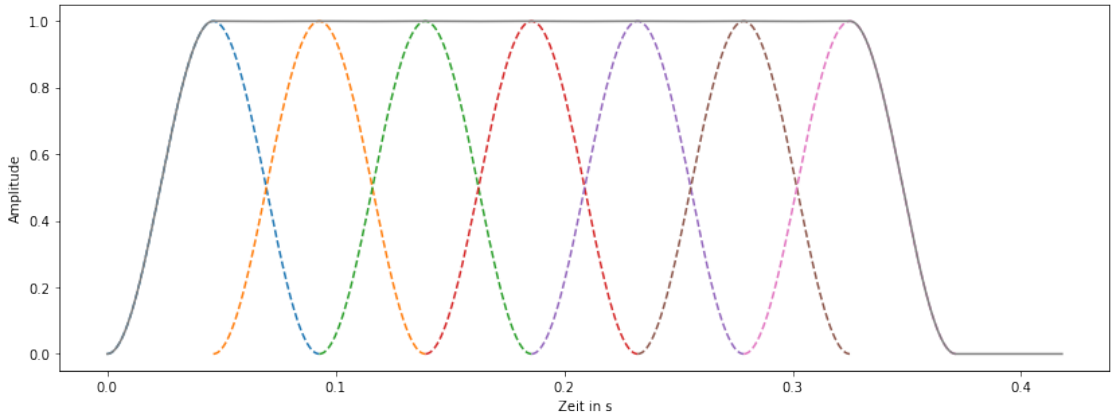
entspricht und somit alle Kriterien ausreichend erfüllt.

```
%matplotlib inline
import matplotlib.pyplot as plt

win_size = 2048
stride = win_size >> 1

window = numpy.hanning(win_size)

def signal_to_frames(signal):
    frames = librosa.util.frame(signal, frame_length=win_size,␣
 ↪hop_length=stride)
    return numpy.swapaxes(frames, 0, 1)

def frames_to_signal(frames):
    count = len(frames) * stride + win_size
    ret = numpy.ndarray(count)
```

```
    ret.fill(0)
    offset = 0
    for frame in frames:
        ret[offset:offset+len(frame)] += frame * window
        offset += stride
    return ret[:int(sr*T)]
```

Die Abschitte überlappen sich zu 50%. Die nachfolgende Grafik zeigt, wie die überlappenden Abschnitte mit Hilfe der Hanning-Fensterfunktion rekombiniert werden.

[ ]:
```
frames = signal_to_frames(numpy.ones(stride*8))
plt.figure(figsize=(14, 5))
plt.xlabel("Zeit in s")
plt.ylabel("Amplitude")
for i, frame in enumerate(frames):
    plt.plot(t[i*stride:i*stride+win_size], frame * window, '--')
plt.plot(t[0:stride*9], frames_to_signal(frames));
```



1.2.2 Methoden der Rauschkategorisierung

Die Amplitude des Rauschens im Eingangssignal ist im betrachteten Zeitraum nicht konstant. Jedoch ist die Energie des Rauschens im gesamten Frequenzspektrum des Signals verteilt und kann damit von dem Nutzsignal unterschieden werden. Wenn der Grenzwert zwischen Störsignal und Nutzsignal bekannt ist, dann kommen unabhängig von der Domäne zwei Arten von Grenzwertfunktionen in Betracht: - Ein harter Filter $f1$, bei dem alle Werte unter einem Grenzwert auf Null gesetzt werden, und - Ein weicher Filter $f2$, bei dem der Bereich außerhalb des Grenzwertes in Richtung Null skaliert wird.

Diese Filter sind folgend für Werte arr mit dem Grenzwert g mit $0 < g <= max(arr)$ definiert:

[ ]:
```
# Harter Filter
def f1(arr, g):
    return (abs(arr) >= g) * arr
```

```
# Weicher Filter
def f2(arr, g, maximum=1):
    return numpy.sign(arr) * numpy.clip(abs(arr) - g, 0, maximum - g) * (1 /␣
 ↪(maximum-g))
```

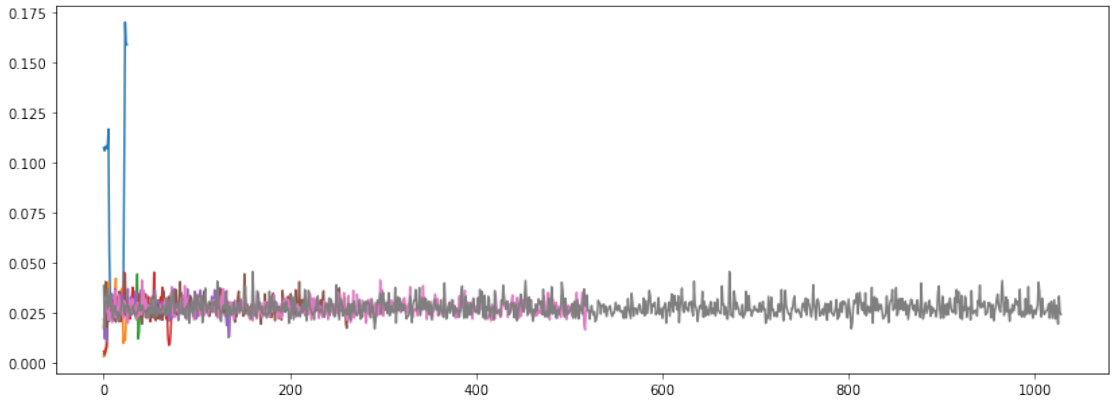
1.2.3 Wahl des Waveletalgorithmus

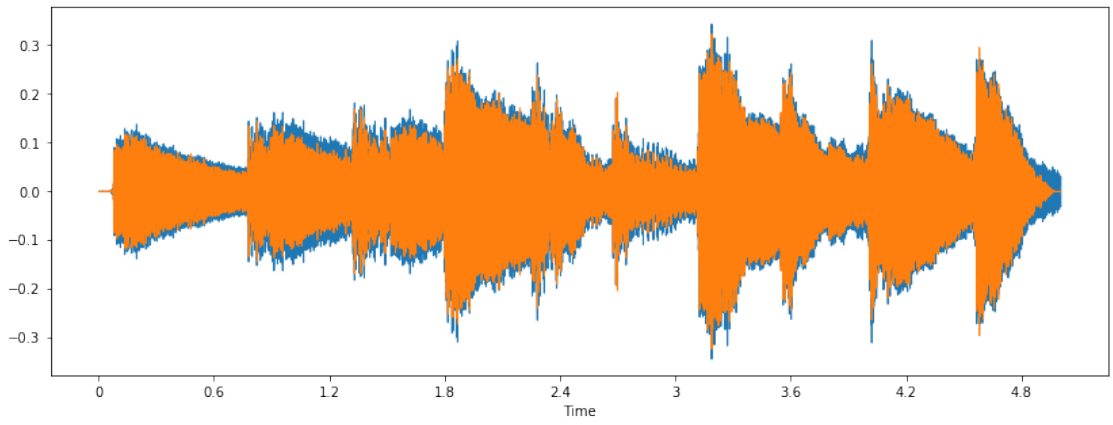
[ ]:
```
import pywt

print(pywt.wavelist(kind='discrete'))
algo = pywt.Wavelet('coif2')
r_frames = signal_to_frames(r)
x_frames = signal_to_frames(x)

def frames_to_coeffs(frames):
    maxlevel = pywt.dwt_max_level(win_size, algo)
    return pywt.wavedec(frames, algo, level=maxlevel)

def coeffs_to_frames(coeffs):
    return pywt.waverec(coeffs, algo)

x_coeffs = frames_to_coeffs(x_frames)
r_coeffs = frames_to_coeffs(r_frames)

plt.figure(figsize=(14, 5))
for i, coeff in enumerate(r_coeffs):
    r_coeffs[i] = numpy.max(numpy.abs(r_coeffs[i]), axis=0)
    plt.plot(r_coeffs[i])

for i, coeff in enumerate(x_coeffs):
    x_coeffs[i] = f2(coeff, r_coeffs[i])

y_frames = coeffs_to_frames(x_coeffs)

y = frames_to_signal(y_frames)
plt.figure(figsize=(14, 5))
librosa.display.waveshow(x, sr=sr)
librosa.display.waveshow(y, sr=sr)
Audio(y, rate=sr)
```

```
['bior1.1', 'bior1.3', 'bior1.5', 'bior2.2', 'bior2.4', 'bior2.6', 'bior2.8',
'bior3.1', 'bior3.3', 'bior3.5', 'bior3.7', 'bior3.9', 'bior4.4', 'bior5.5',
'bior6.8', 'coif1', 'coif2', 'coif3', 'coif4', 'coif5', 'coif6', 'coif7',
'coif8', 'coif9', 'coif10', 'coif11', 'coif12', 'coif13', 'coif14', 'coif15',
'coif16', 'coif17', 'db1', 'db2', 'db3', 'db4', 'db5', 'db6', 'db7', 'db8',
'db9', 'db10', 'db11', 'db12', 'db13', 'db14', 'db15', 'db16', 'db17', 'db18',
```

```
'db19', 'db20', 'db21', 'db22', 'db23', 'db24', 'db25', 'db26', 'db27', 'db28',
'db29', 'db30', 'db31', 'db32', 'db33', 'db34', 'db35', 'db36', 'db37', 'db38',
'dmey', 'haar', 'rbio1.1', 'rbio1.3', 'rbio1.5', 'rbio2.2', 'rbio2.4',
'rbio2.6', 'rbio2.8', 'rbio3.1', 'rbio3.3', 'rbio3.5', 'rbio3.7', 'rbio3.9',
'rbio4.4', 'rbio5.5', 'rbio6.8', 'sym2', 'sym3', 'sym4', 'sym5', 'sym6', 'sym7',
'sym8', 'sym9', 'sym10', 'sym11', 'sym12', 'sym13', 'sym14', 'sym15', 'sym16',
'sym17', 'sym18', 'sym19', 'sym20']
```

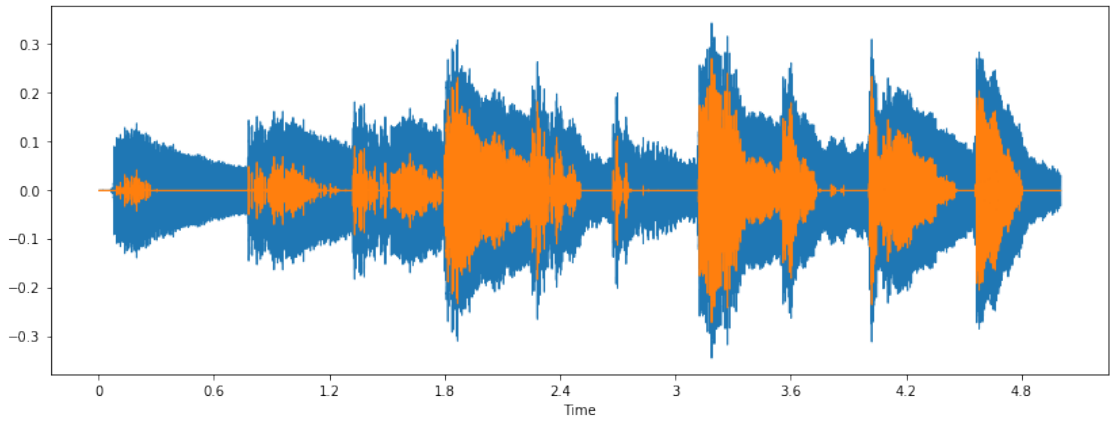
[ ]:
```
<IPython.lib.display.Audio object>
```



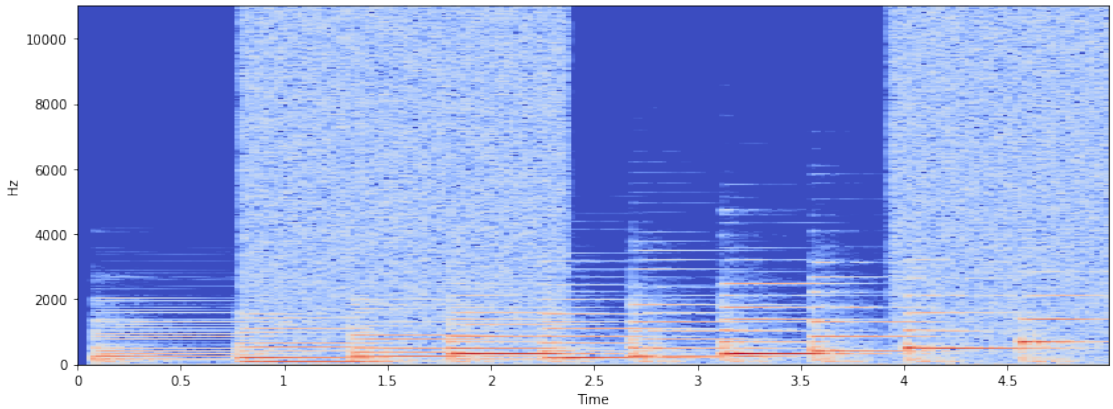


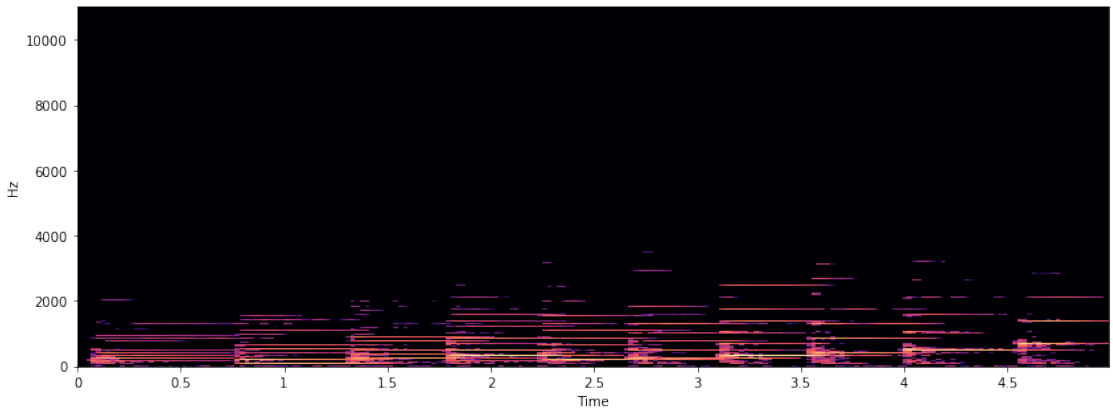
[ ]:
```
yt = f2(x, 0.1)
Audio(yt, rate=sr)

plt.figure(figsize=(14, 5))
librosa.display.waveshow(x, sr=sr)

librosa.display.waveshow(yt, sr=sr)
```

[ ]: <librosa.display.AdaptiveWaveplot at 0x7fb8fc380460>

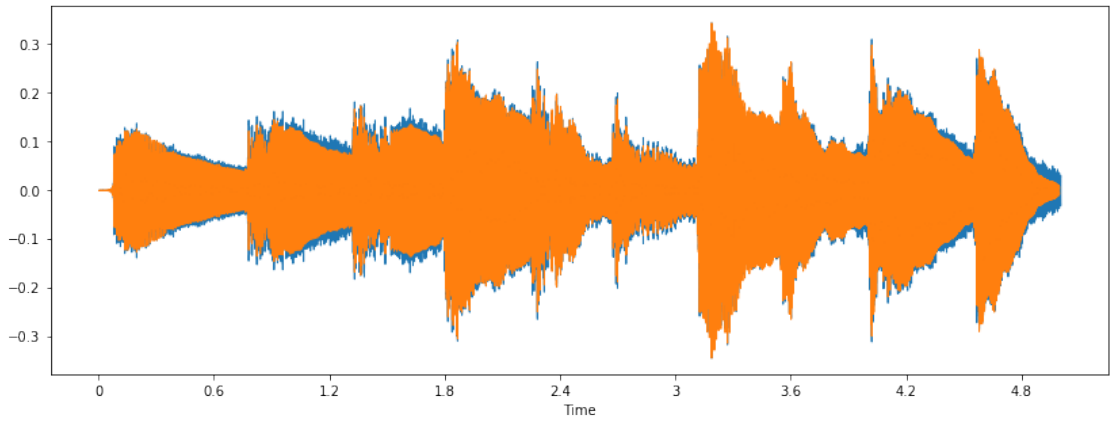


[ ]:
```
X = librosa.stft(x, n_fft=2**11)
Y = f2(abs(X), 0.9, numpy.max(abs(X))) * numpy.exp(1j * numpy.angle(X))

y = librosa.istft(Y)
y = y * max(x) / max(y)


Xdb = librosa.amplitude_to_db(abs(X))
Ydb = librosa.amplitude_to_db(abs(Y))

plt.figure(figsize=(14, 5))
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
plt.figure(figsize=(14, 5))

librosa.display.specshow(Ydb, sr=sr, x_axis='time', y_axis='hz');
```





[ ]:
```
plt.figure(figsize=(14, 5))
librosa.display.waveshow(x, sr=sr)
librosa.display.waveshow(y, sr=sr);
```



[ ]:
```
Audio(y, rate=sr)
```

[ ]:
```
<IPython.lib.display.Audio object>
```

[ ]:

[ ]: