Hochschule Konstanz
Technik, Wirtschaft und Gestaltung

Signale, Systeme und Sensoren

Aufbau eines einfachen Spracherkenners

Tobias Schoch, Luca Strattmann

Konstanz, 26. Mai 2019

Zusammenfassung (Abstract)

Thema:	Aufbau eines einfachen Spracherkenners	
Autoren:	Tobias Schoch	tobias.schoch@htwg-konstanz.de
	Luca Strattmann	luca.strattmann@htwg-konstanz.de
Betreuer:	Prof. Dr. Matthias O. Franz	mfranz@htwg-konstanz.de
	Jürgen Keppler	juergen.keppler@htwg-konstanz.de
	Christoph Kaiser	ch241kai@htwg-konstanz.de

In dem vierten Versuch der Versuchsreihe mit dem Titel *Aufbau eines einfachen Spracherkenners*, geht es um einen Spracherkenner und dessen Auswertung.
So werden wir einen einfachen Spracherkenner aufbauen, der beispielsweise zur Steuerung eines Staplers in einem Hochregallager dienen könnte.

Es reichen hierzu die Erkennung von vier einfachen Befehlen (Hoch, Tief, Links, Rechts). Der Aufbau des Spracherkenners folgt dem in der Vorlesung beschriebenen Prinzip des Prototyp Klassifikators.
Wir werden die Methoden des Windowing, der Triggerung, des Amplitudenspektrums und der Korrelationskoeffizienten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

1

Einleitung

In dem vierten Versuch der Versuchsreihe mit dem Titel *Aufbau eines einfachen Spracherkenners*, geht es um einen Spracherkenner und dessen Auswertung.
So werden wir einen einfachen Spracherkenner aufbauen, der beispielsweise zur Steuerung eines Staplers in einem Hochregallager dienen könnte.
Es reichen hierzu die Erkennung von vier einfachen Befehlen (Hoch, Tief, Links, Rechts). Der Aufbau des Spracherkenners folgt dem in der Vorlesung beschriebenen Prinzip des Prototyp Klassifikators.

Die zugehörigen Spektren werden mit der Windowing Methode berechnet. Zur Aufnahme der Befehle werden wir ein Mikrofon verwenden, dass mit der Soundkarte des Computers verbunden ist, über das Pythonpaket PyAudio.
Dieses ermöglicht eine bequeme Möglichkeit zum Auslesen der Soundkarte. Zudem werden wir eine Triggerfunktion dem Ausleseprogramm hinzufügen, dass die Signale alle an der selben Stelle anfangen.
Daraufhin werden wir das Signal in ein Amplitudenspektrum anwenden und die Methode des Windowing anwenden. Außerdem werden wir die Korrelationskoeffizienten anwenden.

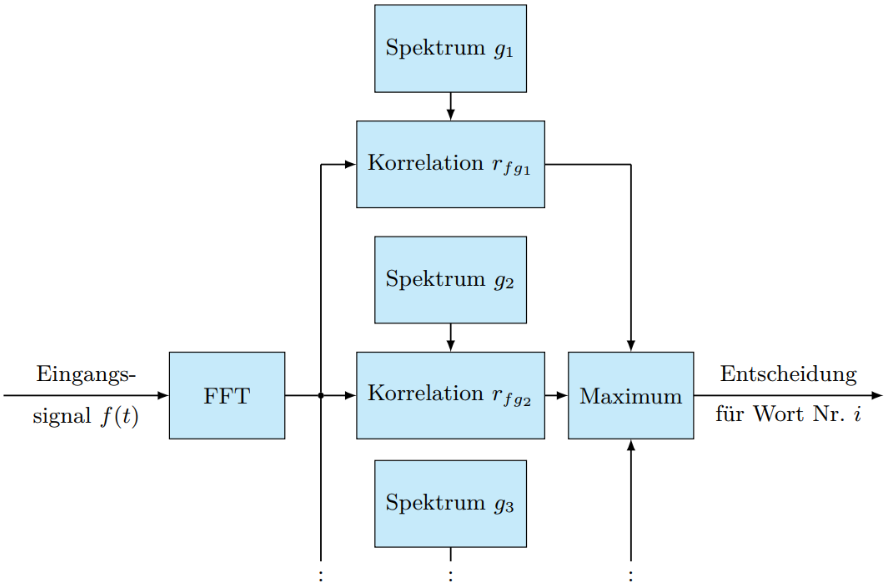


Abbildung 1.1: Architektur des Spracherkenners

2

Versuch 1

2.1 Fragestellung, Messprinzip, Aufbau, Messmittel

Fragestellung:

In dem ersten Teil des Versuchs *'Aufbau eines einfachen Spracherkenners'* werden wir einen Spracherkenner bauen. Zudem erweitern wir den Spracherkenner mit einer Triggerfunktion. Mit der Aufnahme bestimmen wir dessen Amplitudenspektrum. Im Anschluss werden die Methode des Windowing anwenden.
So werden wir eine Testaufnahme machen um diese anschließend auszuwerten.
Auf die entstandenen Numpy-Dateien wenden wir Windowing und das Amplitudenspektrum an.

Messprinzip:

Im ersten Versuch starten wir mit einem Pythonskript. Das Mikrofon wird mit einem Klinkenstecker direkt an die Soundkarte des Computers verbunden. Auf den Computern im Labor, ist das Paket PyAudio bereits in den IDE's integriert.
Mit dem geschriebenen Pythonskript können wir nun akustische Signale aufnehmen. Über das Objekt *audiorecorder* haben wir Zugriff auf die Aufnahmefunktion der Soundkarte.
Das Signal speichern wir anschließend mittels *numpy.save()*. Im Anschluss sollen wir eine beliebige Spracheingabe aufnehmen und diese in einem Diagramm darstellen.

Im Anschluss sollen wir das Aufnahmeprogramm um eine Triggerfunktion erweitern, welche die Aufnahme erst ab einem gewissen Lautstärkepegel starten lässt.
So können wir sicher stellen, dass alle Aufnahmen den selben Startpunkt besitzen.

Das Signal soll eine Dauer von einer Sekunde haben und die fehlenden Samples mit Nullen augefüllt werden.

Mit dem Code Aus dem dritten Versuch können wir mit der Aufnahme das Amplitudenspektrum bestimmen. Dies stellen wir grapisch dar. Danach implementieren die Methode des Windowing.
Diese werden wir jeweils in einer Länge von 512 Samples darstellen. Die einzelnen Windows werden wir mit der Gaußschen Fensterfunktion multiplizieren die eine Fensterbreite von der Standartabweichung 4 hat.
Den ersten Versuch werden wir mit dem Amplitudenspektrum erneut überprüfen und so das Spektrum aus der letzten Aufgabe auf Korrektheit überprüfen.

Aufbau:
Das Mikrofon wird durch einen Klinkenstecker direkt an die Soundkarte des Computers verbunden. Durch ein Pythonskript können wir nun Sprachaufnahmen machen.

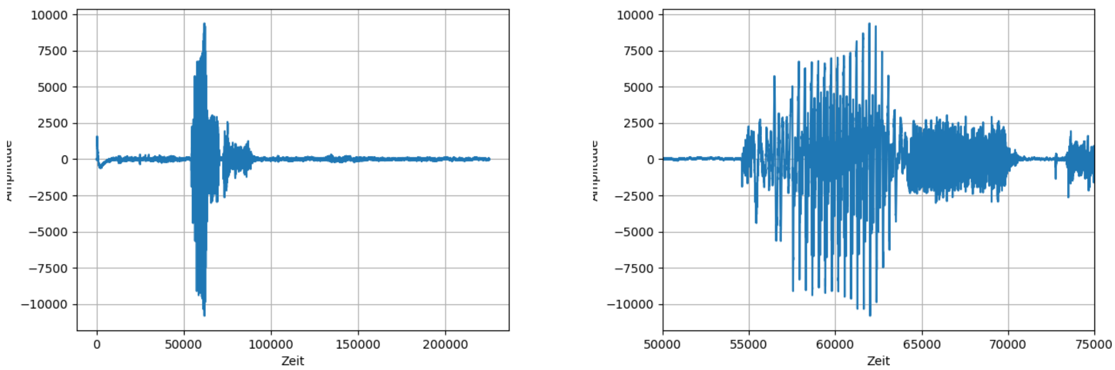
Abbildung 2.1: Aufnahme mit Triggerfunktion des Wortes rechts

Messmittel:

- Ein Mikrofon
- Ein Computer mit einer Python IDE

2.2 Messwerte

Mit einem Mikrofon das an den Computer angeschlossen ist, sehen wir die mit dem Pythonskript aufgenommene Sprachaufnahme des Wortes *'Test'*, welche mit Python visualisiert wurde.
In der linken Darstellung ist die gesamte Dauer des empfangenen Signals, welche sich durch einen einfachen Plot des Signals darstellen lässt.
Im rechten Bild ist eine verkürzte Darstellung der Sprachaufnahme um das Signal besser zu erkennen welches durch *plt.xlim(0, 35000)* auf eine Frequenz von 35000 limitiert wird.



(a) Sprachaufnahme des Wortes Test (b) Dieselbe Sprachaufnahme mit kurzer Zeitachse

Abbildung 2.2: Die aufgenommene Sprachaufnahme visualisiert mit Python

Die Aufnahme wurde durch die Funktion *p.open()* gestartet.
Im Gegensatz zum zweiten Teil der Aufgabe wird hier bereits durch das Starten des Skriptes aufgenommen.

Im folgenden sieht man die Sprachaufnahme des Wortes 'Rechts' mit der Triggerfunktion. Die Aufnahme läuft bereits durch das Starten des Programmes.
Die Stelle bei der die Überschreitung des Schwellenwertes geschieht wird gespeichert und beginnt ab dieser Stelle.
Durch die Triggerfunktion startet die Aufnahme dadurch erst ab dem Schwellenwert.

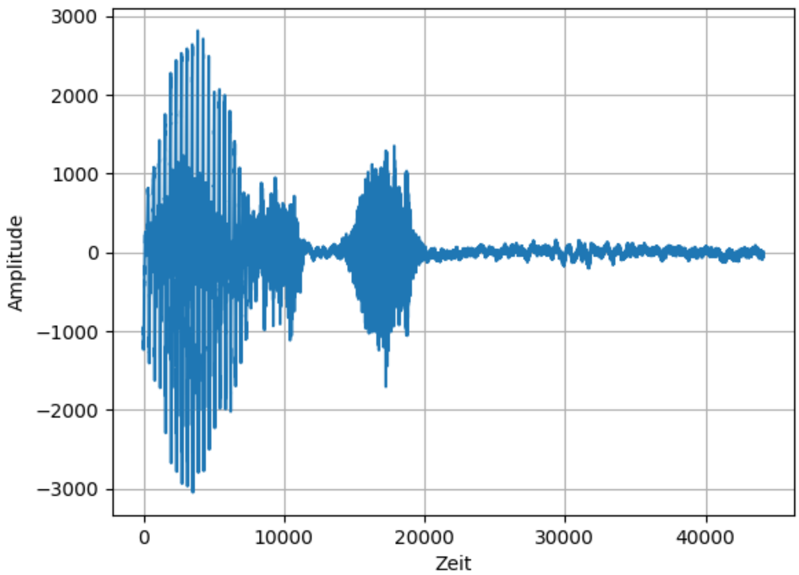
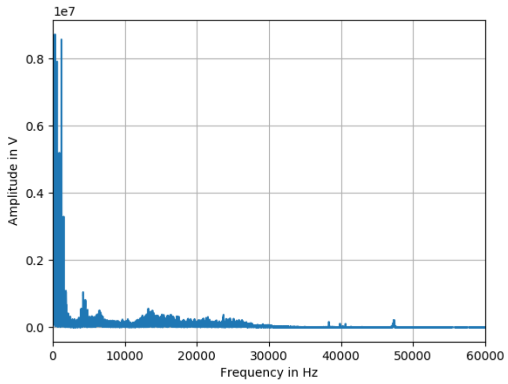


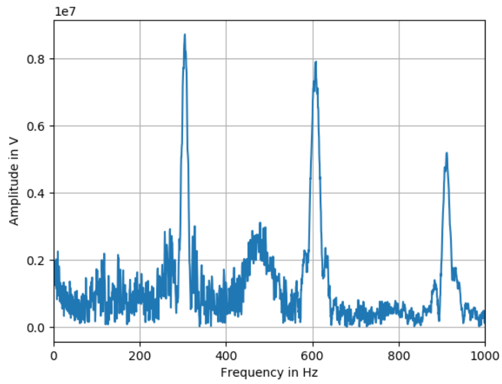
Abbildung 2.3: Aufnahme mit Triggerfunktion des Wortes rechts

2.3 Auswertung

Auf der linken Seite ist das Amplitudenspektrum und auf der rechten Seite das Amplitudenspektrum bis zur Frequenz von 1000.
Diese wurden mit der Formel berechnet. Mit der Anwendung der Formel auf jedes eingegangene Signal erhalten wir den folgenden Plot:



(a) Amplitudenspektrum der Sprachaufnahme



(b) Amplitudenspektrum mit geringerer Frequenz

Abbildung 2.4: Das Amplitudenspektrum mit der dazugehörigen Frequenz

Im folgenden sieht man eine Gaußsche Fensterfunktion mit der Fensterbreite von 4 Standardabweichungen.

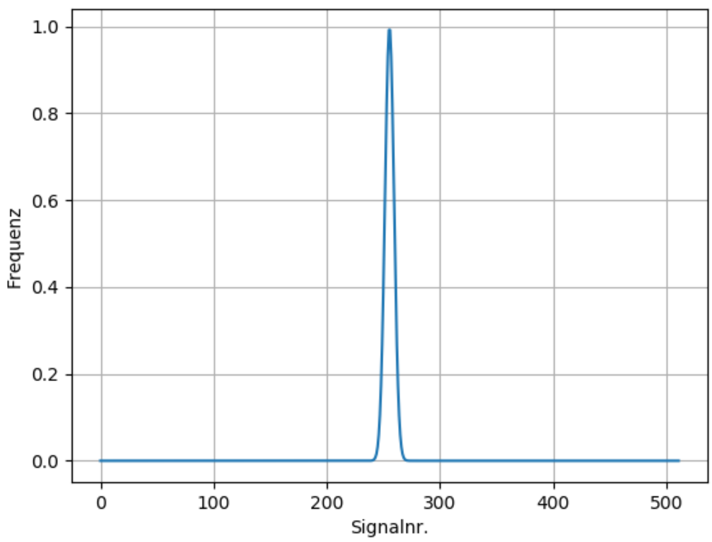


Abbildung 2.5: Gausssche Fensterfunktion mit Standardabweichung 4

Im Anschluss zerlegt man das Signal in Abschnitte mit der von Länge 512 Samples, die sich jeweils zur Hälfte überlappen sollen. Das Signal wird mit dem entsprechenden Signal der Gaußschen Fensterfunktion multipliziert und daraufhin eine lokale Fouriertransformation durchgeführt. Zu guter Letzt werden die Fouriertransformierten über alle Fenster gemittelt.

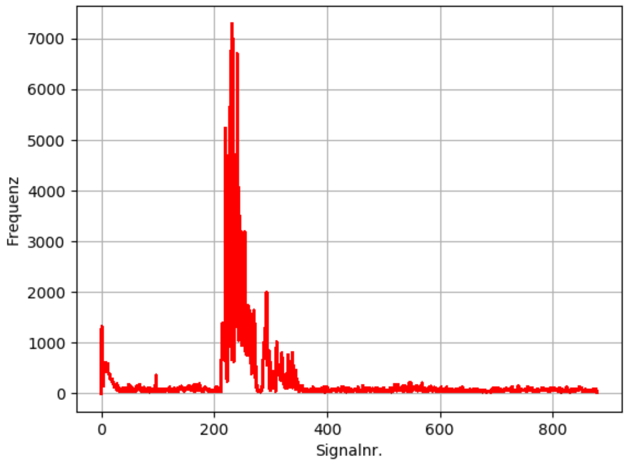


Abbildung 2.6: Aufnahme mit Triggerfunktion des Wortes rechts

Das Windowing wird auf das Signal angewandt. Nachdem zum Beispiel bei dem Wort 'Hoch' über 170 Windows ausgegeben wurden als Plot, sieht der Explorer folgendermaßen aus.

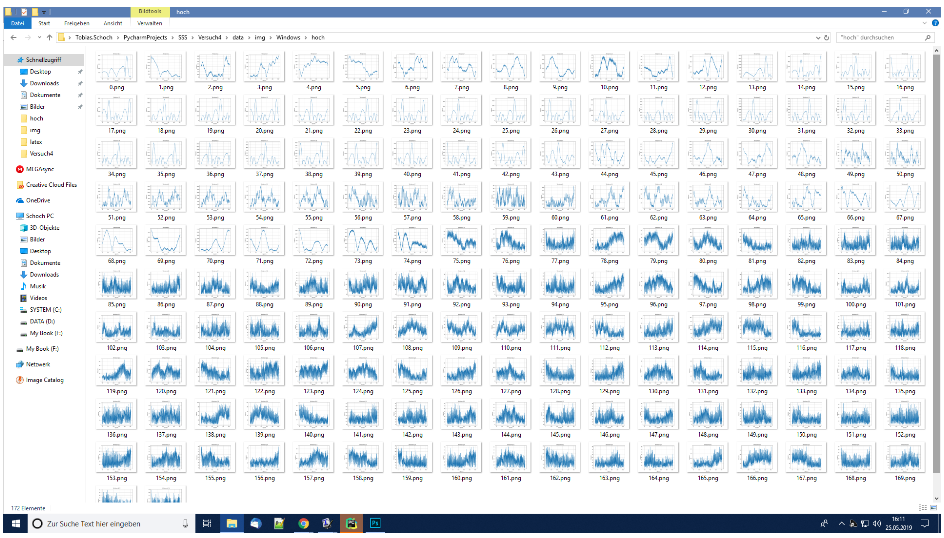


Abbildung 2.7: Aufnahme mit Triggerfunktion des Wortes rechts

2.4 Interpretation

Bei der Betrachtung der Sprachaufnahme sehen wir die übliche Amplituden und Frequenzen bei der Aussprache des Wortes 'Test'. In der Vergrößerung sind nochmals besser die Amplitudenausschläge zu sehen.

In der Abbildung 2.3 sieht man die Sprachaufnahme des Wortes 'Rechts'. Die Aufnahme wurde mit einem Pythonskript gemacht, dass eine Triggerfunktion implementiert hat. Ab einem bestimmten Schwellenwert beginnt die Aufnahme. Hier ist es schön zu sehen, dass die Aufnahme nicht mit Start des Skriptes beginnt, sondern erst wenn gesprochen wird.

Auf das Signal wurde daraufhin das Amplitudenspektrum berechnet. Durch dieses erfährt man die Frequenzen die in einem Wort stecken. Der größte Ausschlag bei der Frequenz mit eienr Amplitude von über 0.8 ist die maximalste Amplitude. Die Frequenz liegt bei knapp über 300Hz. Die x Achse wurde auf die Hälfte gekürzt, da bei ein Amplitudenspektrum sich nach der Hälfte wiederholt.

Wenn man diese mit dem Gaußschen Fenster multipliziert, erhält man die gemittelten fouriertransfomierten Fenster. Desto höher der absolute Ausschlag, desto höher ist die Amplitude im Window. Durch den senkrechten Ausschnitte sind steile Übergänge erzeugt worden, die im ursprünglichen Signal nicht vorhanden sind. Steile Übergänge erzeugen jedoch viele hohe Frequenzen im Spektrum.

3

Versuch 2

3.1 Fragestellung, Messprinzip, Aufbau, Messmittel

Im zweiten Versuch müssen wir vier verschiedene Befehle jeweils fünf mal aufnehmen.
Die Befehle die wir aufnehmen lauten:

- Links
- Rechts
- Hoch
- Tief

Anhand der aufgenommenen Numpy Funktionen berechnen wir die Spektren mit der Windowing Methode. Daraufhin berechnen wir noch die eigentlichen Referenzspektren.
Im Anschluss berechnen wir noch den Korrelationskoeffizienten nach Bravais-Pearson mit dem wir zwei verschiedene Spektren miteinander vergleichen können.
Dafür brauchen wir vom selben und einem anderen Sprecher erneut Sprachaufnahmen.
Mit diesen testen wir die Routine an den Referenzspektren:
Beim Vergleich identischer Spektren sollte die Korrelation 1 sein, bei verschiedenen Spektren nahe an 0.
Zudem sollen wir eine Fehler- und eine Detektionsrate angeben.

3.2 Messwerte

Die sämtlichen aufgenommenen Sprachaufnahmen in Form von Numpy-Dateien werden wir mit der Windowing Methode in Spektren berechnen. Dabei wird erstmal das Spektrum mit der Gaußschen Fensterfunktion berechnet und multipliziert mit den einzelnen Windows. Danach wird das Fenster absolut fouriertransformiert. Daraus wiederrum wird der Durchschnitt berechnet.

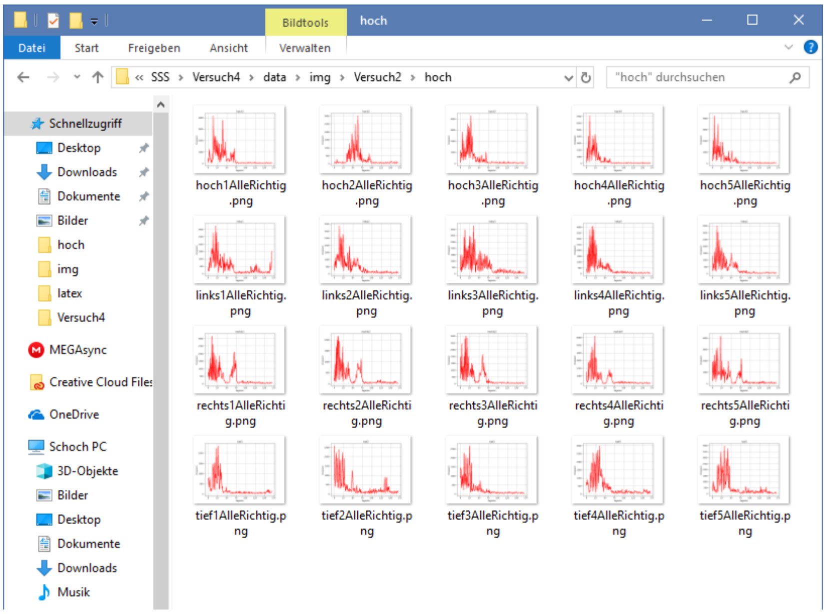
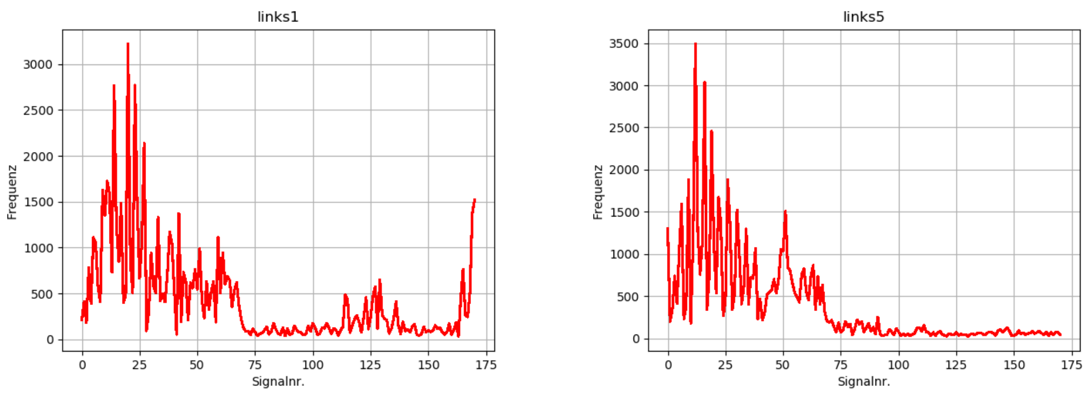


Abbildung 3.1: Sämtliche Signale mit Windowing berechnet



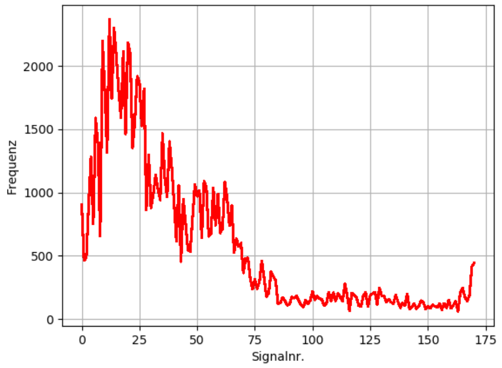
(a) Erste Sprachaufnahme von 'links'

(b) Fünfte Sprachaufnahme von 'links'

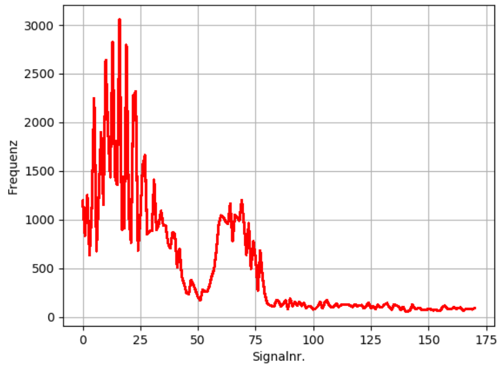
Abbildung 3.2: Zwei verschiedene Sprachaufnahmen im Vergleich.

3.3 Auswertung

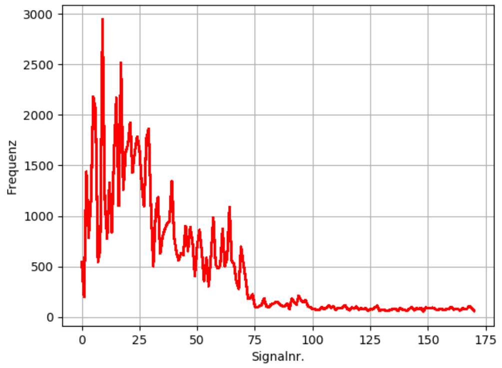
Die Daten aus den Numpydateien werden in einzelne Windows zerlegt mit einer Länge von 512 Samples die sich jeweils überlappen.
Die einzelnen Samples werden mit der Gaußschen Fensterfunktion multipliziert, welche eine Fensterbreite von 4 Standardabweichungen hat. In jedem Fenster wird eine lokale Fouriertransformation durchgeführt. Daraufhin werden die Fouriertransformierten über alle Fenster gemittelt.
Nachdem man sämtliche 4 Befehle und die jeweils fünf Beispiel dazu gemittelt hat, erhält man das Referenzspektrum.



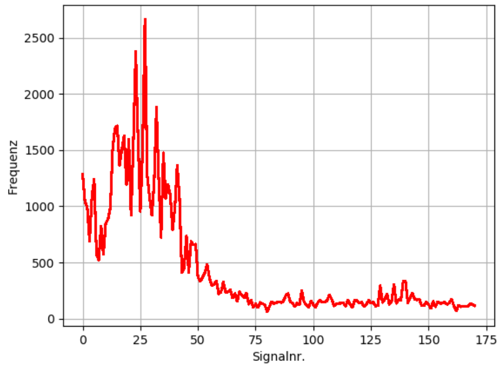
(a) Person 1: Referenzspektrum 'Links'



(b) Person 1: Referenzspektrum 'Rechts'



(c) Person 1: Referenzspektrum 'Hoch'



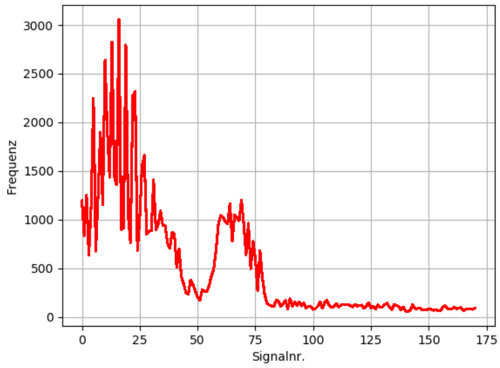
(d) Person 1: Referenzspektrum 'Tief'

Abbildung 3.3: Die vier Referenzspektren

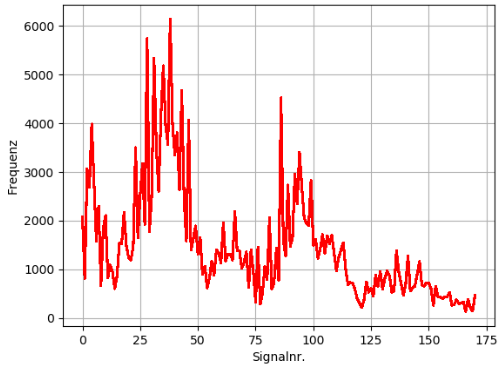
Hier werden zwei Eingabespektren beziehungsweise zwei Dateien mittels der Bravais-Pearson Methode verglichen auf ihre Korrelationskoeffizienten.
Wenn die Spektren identisch sind, sollte der Wert 1.0 ergeben. Bei ungleichen Spektren sollte dieser nahe 0 sein. Person 1 ist die Person, welche die Spracheeingabe gemacht hat.

Spracheingabe	Person 1	Person 2
Hoch	**0.5817**	**0.5496**
1 - Hoch	0.6856	0.6283
2 - Hoch	0.5001	0.6000
3 - Hoch	0.5829	0.5600
4 - Hoch	0.5754	0.4902
5 - Hoch	0.5644	0.4691
Tief	**0.7088**	**0.6172**
1 - Tief	0.6682	0.5976
2 - Tief	0.7210	0.6520
3 - Tief	0.6947	0.4947
4 - Tief	0.7069	0.6098
5 - Tief	0.7531	0.7318
Links	**0.6514**	**0.6084**
1 - Links	0.6708	0.5711
2 - Links	0.6451	0.5953
3 - Links	0.6273	0.5785
4 - Links	0.6472	0.6512
5 - Links	0.6666	0.6460
Rechts	**0.4219**	**0.1779**
1 - Rechts	0.5001	0.1401
2 - Rechts	0.5402	0.2892
3 - Rechts	0.5032	0.2790
4 - Rechts	0.3895	0.2849
5 - Rechts	0.1764	0.1764

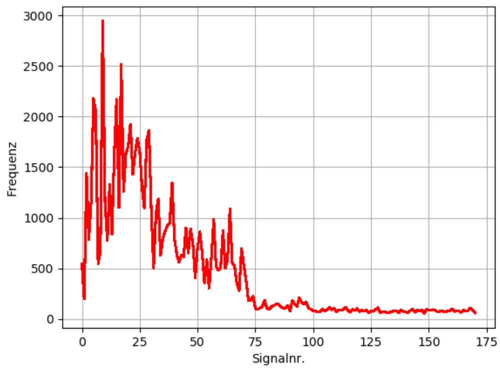
Tabelle 3.1: Vergleich der berechneten Referenzspektren



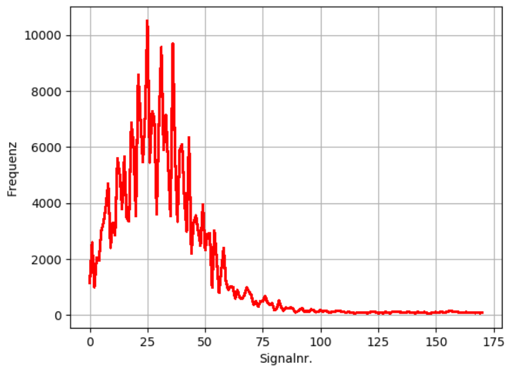
(a) Person 1: Referenzspektrum 'Rechts'



(b) Person 2: Referenzspektrum 'Rechts'



(c) Person 1: Referenzspektrum 'Hoch'



(d) Person 2: Referenzspektrum 'Hoch'

Abbildung 3.4: Die vier Referenzspektren mit den größten Unterschieden

3.4 Interpretation

In Abbildung [3.2] sieht man, wie sich Sprachaufnahmen von der selben Person mit dem selben Wort unterscheiden können. Im hinteren Teil von der linken Abbildung sieht man zum Schluß eine Steigung der Spannung.
Diese ist entstanden durch ungefähr 8 bis 9 andere Gruppen im Raum, durch welche die Hintergrundgeräusche sehr laut waren. Die anderen Geräusche unterscheiden sich nur geringfügig.

Nachdem wir die Signale ausgewertet haben, haben wir mit der Windowing Methode und der Bravais-Pearson den Korrelationskoeffizienten berechnet. In der Abbildung [3.3] sieht man die durchschnittlichen Spektren für eines der Worte (Hoch, Tief, Links, Rechts).
Dabei ähneln sich die Spektren der Wiederholungen der Worte sehr. Das Wort rechts hat ein sehr ausgeprägtes Spektrum.
Am Anfang geht das Signal sehr steil nach oben durch das Vokal 'e'. Bei den Konsonanten 'ch' gibt es eine sehr große Verringerung. Zum Schluss bei 's' geht das Signal wieder nach oben. So kann man gut die einzelnen Buchstaben des Wortes identifizieren.

Bei der Bravais-Pearson Methode mit den Referenzspektren und der erneuten Aufnahme der Worte erhält man eine Lange Tabelle mit einigen Werten. Am deutlichsten ist der Unterschied bei der Aufnahme mit dem Wort 'Rechts'.
Auch ist es gut sichtbar bei einzelnen Aufnahmen mit dem Wort 'Tief'. Der größte Unterschied liegt bei 0.35, was ziemlich viel ist, wenn man die Skala von 1.0 - 0.0 betrachtet.
Die Bravais-Pearson Korrelationskoeffizientenberechnung hat sich gelohnt und als richtig erwiesen, da die Werte immer bei den Sprachaufnahmen der selben Person höher waren, als bei der fremden Person.

Hier [3.4] kann nochmal gut einzelne Spektren begutachten, die das selbe Wort von unterschiedlichen Menschen darstellen. Man kann gut betrachten, dass der zweite Sprecher definitiv energischer und lauter gesprochen hat.
Die Amplituden liegen im zwei- bis sogar dreifachen Bereich und haben ein sehr viel unruhigeres Spektrum.

Anhang

A.1 Quellcode

A.1.1 Quellcode Versuch 1

```
# ---------------------------------------
# Task 1.1
# ---------------------------------------

import pyaudio
import numpy
import matplotlib.pyplot as plt

FORMAT = pyaudio.paInt16 # Voreinstellungen der Aufnahme
SAMPLEFREQ = 44100
FRAMESIZE = 1024
NOFFRAMES = 220
p = pyaudio.PyAudio()
print('running')
stream = p.open(format=FORMAT, # Aufnahmestart
                channels=1,
                rate=SAMPLEFREQ,
                input=True,
                frames_per_buffer=FRAMESIZE)
data = stream.read(NOFFRAMES * FRAMESIZE)
decoded = numpy.fromstring(data, 'Int16');
numpy.save('aufgabe4/test.npy', decoded)
stream.stop_stream() # Aufnahmestop
stream.close()
p.terminate()
```

Listing 4.1: Einlesen der Sprachaufnahme und ablegen des Signals in eine Numpy Datei

```python
# ----------------------------------------
# Task 1.2
# ----------------------------------------

import pyaudio
import numpy as np
import matplotlib.pyplot as plt
import time

def time_axis(arr):
    return np.array(range(len(arr)))/44100

# Voreinstellungen für die Aufnahme
FORMAT = pyaudio.paInt16
SAMPLEFREQ = 44100
FRAMESIZE = 44100
NOFFRAMES = 2

# Aufnahmefunktion der Soundkarte
p = pyaudio.PyAudio()
print('running')
# Aufnahmestart
stream = p.open(format=FORMAT, channels=1, rate=SAMPLEFREQ,
                input=True, frames_per_buffer=FRAMESIZE)
data = stream.read(NOFFRAMES * FRAMESIZE)
decoded = np.fromstring(data, 'Int16') / ((2**15)/2-1)
# Aufnahmestop
stream.stop_stream()
stream.close()
p.terminate()

# Triggerfunktion lässt die Funktion erst ab Schwellenwert starten
start = np.argmax(np.abs(decoded) > 0.05) - 1024
# Berechnung des Endwerts der Aufnahme
end = start + 44100
triggered = decoded[start:end]
triggered = np.concatenate((triggered, [0]*(44100 - end - start)))
# Aufnamespektrum ausgeben in eine Numpy Datei
np.savetxt("aufgabe4/tief_5_" + str(int(time.time())) + ".npy", triggered)
```

Listing 4.2: Einlesen einer Sprachaufnahme mit Aktivierung durch Triggerung

```python
# ---------------------------------------
# Task 1.3
# ---------------------------------------

import matplotlib.pyplot as plt
import numpy as np

# Einlesen der .npy Datei
data = np.load('data/test.npy')
freq = np.zeros(225280)

# Darstellung des Amplitudenspektrums
plt.plot(data)
plt.grid()
plt.xlabel('Zeit')
plt.ylabel('Amplitude')
plt.savefig('data/img/testamp.png')
plt.show()

# Einlesen der .csv Datei
data2 = np.load('data/rechts2.npy')

# Darstellung des Amplitudenspektrums
plt.plot(data2)
plt.grid()
plt.xlabel('Zeit')
plt.ylabel('Amplitude')
plt.savefig('data/img/rechtsamp.png')
plt.show()

# Darstellung des Amplitudenspektrums
plt.plot(data)
plt.grid()
plt.xlabel('Zeit')
plt.ylabel('Amplitude')
plt.xlim(50000, 75000)
plt.savefig('data/img/testamp2.png')
plt.show()

# Der zweite Wert wird absolut minus den ersten absoluten wert gerechnet um später den Wert
difference = 2 / 225280
```

```
# Die zweite Spalte der .csv Datei wird Fouriertransformiert
fourier = np.fft.fft(data[:225280])
# Die Fouriertransformierte Frequenz wird absolutiert, so dass kein negativer Wert mehr vorzufinden ist
spektrum = np.abs(fourier)
# Formel um die Anzahl der Schwingungen in die Freuquenz umzurechnen − f = n / (M ∗ t)
for x in range(0, 225280, 1):
    freq[x] = (x / (difference * 225280))

# Darstellung des Amplitudenspektrums
plt.plot(freq, spektrum)
plt.grid()
plt.xlabel('Frequency in Hz')
plt.ylabel('Amplitude in V')
plt.xlim(0, 60000)
plt.savefig('data/img/testspektrum1.png')
plt.show()

# Darstellung des Amplitudenspektrums in vergrößerter Darstellung
plt.plot(freq, spektrum)
plt.grid()
plt.xlabel('Frequency in Hz')
plt.ylabel('Amplitude in V')
plt.xlim(0, 35000)
plt.savefig('data/img/testspektrum2.png')
plt.show()

# Darstellung des Amplitudenspektrums in vergrößerter Darstellung
plt.plot(freq, spektrum)
plt.grid()
plt.xlabel('Frequency in Hz')
plt.ylabel('Amplitude in V')
plt.xlim(0, 1000)
plt.savefig('data/img/testspektrum3.png')
plt.show()
```

Listing 4.3: Amplitudenspektrum und Ausgabe von Plots

```python
# ----------------------------------------
# Task 1.4
# ----------------------------------------

from scipy import signal
import matplotlib.pyplot as plt
import numpy as np

# Einlesen der .npy Datei
data = np.load('data/test.npy')
window = np.zeros((879, 512))
z = 256
freq = np.zeros(225280)
gaussianwindow = signal.windows.gaussian(512, std=4)
for y in range(0, 879):
    z = z - 256
    for x in range(0, 512):
        window[y, x] = np.mean(np.abs(np.fft.fft(data[z] * gaussianwindow)))
        z = z + 1
    # plt.plot(window[y])
    # plt.title('Windownr' + str(y+1))
    # plt.xlabel('Signalnr.')
    # plt.ylabel('Frequenz')
    # plt.grid(True)
    # plt.savefig('data/img/' + str(y) + '.png')
    # plt.show()

# Darstellung des Amplitudenspektrums
plt.plot(gaussianwindow)
plt.grid(True)
plt.xlabel('Signalnr.')
plt.ylabel('Frequenz')
plt.savefig('data/img/gauss.png')
plt.show()

# Darstellung des Amplitudenspektrums
plt.plot(window)
plt.grid(True)
plt.xlabel('Signalnr.')
plt.ylabel('Frequenz')
plt.savefig('data/img/Alle.png')
```

```
plt.show()

for y in range(0, 879):
    for x in range(0, 512):
        window[y][x] = window[y][x] * gaussianwindow[x]
    window[y] = np.abs(np.fft.fft(window[y]))
    window[y] = np.mean(window[y])

plt.plot(window, 'r')
plt.grid(True)
plt.xlabel('Signalnr.')
plt.ylabel('Frequenz')
plt.savefig('data/img/AlleRichtig.png')
plt.show()

# Der zweite Wert wird absolut minus den ersten absoluten wert gerechnet um später den Wert
difference = 2 / 225280
# Die zweite Spalte der .csv Datei wird Fouriertransformiert
fourier = np.fft.fft(window)
# Die Fouriertransformierte Frequenz wird absolutiert, so dass kein negativer Wert mehr vorzufinden ist
spektrum = np.abs(fourier)
# Formel um die Anzahl der Schwingungen in die Freuquenz umzurechnen - f = n / (M * t)
for x in range(0, 225280, 1):
    freq[x] = (x / (difference * 225280))

plt.plot(freq, 'r')
plt.grid(True)
plt.xlabel('Signalnr.')
plt.ylabel('Frequenz')
plt.savefig('data/img/ampwin.png')
plt.show()
```

Listing 4.4: Windowing und Ausgabe von Plots bzw. Windows

A.1.2 Quellcode Versuch 2

```
# ------------------------------------
# Task 2.1
# ------------------------------------

from scipy import signal
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats

# Definieren der Dateinamen
num = ["hoch1", "hoch2", "hoch3", "hoch4", "hoch5", "tief1", "tief2", "tief3", "tief4", "tief5", "links1",
      "links2", "links3", "links4", "links5", "rechts1", "rechts2", "rechts3", "rechts4", "rechts5"]
num2 = ["ahoch1", "ahoch2", "ahoch3", "ahoch4", "ahoch5", "atief1", "atief2", "atief3", "atief4", "atief5",
       "alinks1", "alinks2", "alinks3", "alinks4", "alinks5", "arechts1", "arechts2", "arechts3", "arechts4",
       "arechts5"]
numm = ["hoch1", "hoch2", "hoch3", "hoch4", "hoch5", "tief1", "tief2", "tief3", "tief4", "tief5", "links1",
       "links2", "links3", "links4", "links5", "rechts1", "rechts2", "rechts3", "rechts4", "rechts5"]
nummm = ["hoch1", "hoch2", "hoch3", "hoch4", "hoch5", "tief1", "tief2", "tief3", "tief4", "tief5", "links1",
        "links2", "links3", "links4", "links5", "rechts1", "rechts2", "rechts3", "rechts4", "rechts5"]
capital = ["hoch", "tief", "links", "rechts"]
capital2 = ["hoch", "tief", "links", "rechts"]

# Gaußfenster definieren mit Fensterbreite Standardabweichung 4
gaussianwindow = signal.windows.gaussian(512, std=4)

# Darstellung des Gaußfensters
plt.plot(gaussianwindow)
plt.grid(True)
plt.xlabel('Signalnr.')
plt.ylabel('Frequenz')
plt.savefig('data/img/gauss.png')
plt.show()

# For loop um alle Dateien zu analysieren
for a in range(0, 20):
  # Einlesen der Numpy Dateien von Person 1 & 2
  data = np.load('data/' + str(num[a]) + '.npy')
  data2 = np.load('data/' + str(num2[a]) + '.npy')
  # Definieren eines leeren Vectors für Person 1 & 2
  num[a] = np.zeros((171, 512))
```

```
num2[a] = np.zeros((171, 512))
z = 256

# For loop um die einzelnen Windows zu erstellen
for y in range(0, 171):
    z = z − 256
    # For loop um die einzelnen Frames zu berechnen
    for x in range(0, 512):
        # Signale ∗ Gaußfenster, das wiederrum wird absolut fouriertransformiert.
        # Daraus der Durchschnitt ergibt den Windowingwert
        num[a][y, x] = np.mean(np.abs(np.fft.fft(data[z] ∗ gaussianwindow)))
        num2[a][y, x] = np.mean(np.abs(np.fft.fft(data2[z] ∗ gaussianwindow)))
        z = z + 1
    # plt.plot(num[a][y])
    # plt.title('Windownr' + str(y+1+(a∗171)))
    # plt.xlabel('Signalnr.')
    # plt.ylabel('Frequenz')
    # plt.grid(True)
    # plt.savefig('data/img/' + str(y+1+(a∗171)) + '.png')
    # plt.show()

# For loop um die einzelnen Windows zu erstellen
for y in range(0, 171):
    # For loop um die einzelnen Frames zu berechnen
    for x in range(0, 512):
        # Signale ∗ Gaußfenster, das wiederrum wird absolut fouriertransformiert.
        # Daraus der Durchschnitt ergibt den Windowingwert
        num[a][y, x] = num[a][y, x] ∗ gaussianwindow[x]
        num2[a][y, x] = num2[a][y, x] ∗ gaussianwindow[x]
    num[a][y] = np.abs(np.fft.fft(num[a][y]))
    num2[a][y] = np.abs(np.fft.fft(num2[a][y]))
    num[a][y] = np.mean(num[a][y])
    num2[a][y] = np.mean(num2[a][y])

plt.plot(num[a], 'r') # Plot zur Darstellung der Mittelung der Windows
plt.title(str(nummm[a]))
plt.grid(True)
plt.xlabel('Signalnr.')
plt.ylabel('Frequenz')
plt.savefig('data/img/' + numm[a] + 'AlleRichtig.png')
plt.show()
```

```
# For loop zur Ausgabe der endgültig berechneten Plots
for z in range(0, 4):
    # Vektoren zum Speichern der Plots
    capital[z] = np.zeros((171, 512))
    capital2[z] = np.zeros((171, 512))
    # For loop für die einzelnen Windows
    for y in range(0, 171):
        # For loop für die einzelnen Samples
        for x in range(0, 512):
            # Hoch Mittelung
            if (z == 0):
                capital[z][y, x] = (num[0][y, x] + num[1][y, x] + num[2][y, x] + num[3][y, x]
                                    + num[4][y, x]) / 4
                capital2[z][y, x] = (num2[0][y, x] + num2[1][y, x] + num2[2][y, x]
                                     + num2[3][y, x] + num2[4][y, x]) / 4
            # Tief Mittelung
            elif (z == 1):
                capital[z][y, x] = (num[5][y, x] + num[6][y, x] + num[7][y, x] + num[8][y, x]
                                    + num[9][y, x]) / 4
                capital2[z][y, x] = (num2[5][y, x] + num2[6][y, x] + num2[7][y, x] + num2[8][y, x]
                                     + num2[9][y, x]) / 4
            # Links Mittelung
            elif (z == 2):
                capital[z][y, x] = (num[10][y, x] + num[11][y, x] + num[12][y, x] + num[13][y, x]
                                    + num[14][y, x]) / 4
                capital2[z][y, x] = (num2[10][y, x] + num2[11][y, x] + num2[12][y, x] + num2[13][y, x]
                                     + num2[14][y, x]) / 4
            # Rechts Mittelung
            elif (z == 3):
                capital[z][y, x] = (num[15][y, x] + num[16][y, x] + num[17][y, x] + num[18][y, x]
                                    + num[19][y, x]) / 4
                capital2[z][y, x] = (num2[15][y, x] + num2[16][y, x] + num2[17][y, x] + num2[18][y, x]
                                     + num2[19][y, x]) / 4
    plt.plot(capital[z], 'r') # Geplotete Endwerte
    plt.grid(True)
    plt.xlabel('Signalnr.')
    plt.ylabel('Frequenz')
    plt.savefig('data/img/' + capital2[z] + 'Average.png')
    plt.show()
```

Listing 4.5: Windowing und Mittelung der Spektren

```python
# ----------------------------------------
# Task 2.2
# ----------------------------------------

from scipy import signal
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats

# Definieren der Dateinamen
num = ["hoch1", "hoch2", "hoch3", "hoch4", "hoch5", "tief1", "tief2", "tief3", "tief4", "tief5", "links1",
      "links2", "links3", "links4", "links5", "rechts1", "rechts2", "rechts3", "rechts4", "rechts5"]
numm = ["hoch1", "hoch2", "hoch3", "hoch4", "hoch5", "tief1", "tief2", "tief3", "tief4", "tief5", "links1",
       "links2", "links3", "links4", "links5", "rechts1", "rechts2", "rechts3", "rechts4", "rechts5"]
capital = ["hoch", "tief", "links", "rechts"]
capital2 = ["hoch", "tief", "links", "rechts"]

# Gaußfenster definieren mit Fensterbreite Standardabweichung 4
gaussianwindow = signal.windows.gaussian(512, std=4)

# For loop um alle Dateien zu analysieren
for a in range(0, 20):
    # Einlesen der Numpy Dateien von Person 1
    data = np.load('data/' + str(num[a]) + '.npy')
    # Definieren eines leeren Vectors für Person 1
    num[a] = np.zeros((171, 512))
    z = 256

    # For loop um die einzelnen Windows zu erstellen
    for y in range(0, 171):
        z = z - 256
        # For loop um die einzelnen Frames zu berechnen
        for x in range(0, 512):
            # Signale * Gaußfenster, das wiederrum wird absolut fouriertransformiert.
            # Daraus der Durchschnitt ergibt den Windowingwert
            num[a][y, x] = np.mean(np.abs(np.fft.fft(data[z] * gaussianwindow)))
            z = z + 1

    for y in range(0, 171): # For loop um die einzelnen Windows zu erstellen
        for x in range(0, 512): # For loop um die einzelnen Frames zu berechnen
            # Signale * Gaußfenster, das wiederrum wird absolut fouriertransformiert.
```

```
            # Daraus der Durchschnitt ergibt den Windowingwert
            num[a][y, x] = num[a][y, x] * gaussianwindow[x]
        num[a][y] = np.abs(np.fft.fft(num[a][y]))
        num[a][y] = np.mean(num[a][y])

# For loop zur Ausgabe der endgültig berechneten Plots
for z in range(0, 4):
    # Vektoren zum Speichern der Plots
    capital[z] = np.zeros((171, 512))
    for y in range(0, 171): # For loop für die einzelnen Windows
        for x in range(0, 512): # For loop für die einzelnen Samples
            # Hoch Mittelung
            if (z == 0):
                capital[z][y, x] = (num[0][y, x] + num[1][y, x] + num[2][y, x] + num[3][y, x]
                                    + num[4][y, x]) / 4
            # Tief Mittelung
            elif (z == 1):
                capital[z][y, x] = (num[5][y, x] + num[6][y, x] + num[7][y, x] + num[8][y, x]
                                    + num[9][y, x]) / 4
            # Links Mittelung
            elif (z == 2):
                capital[z][y, x] = (num[10][y, x] + num[11][y, x] + num[12][y, x] + num[13][y, x]
                                    + num[14][y, x]) / 4
            # Rechts Mittelung
            elif (z == 3):
                capital[z][y, x] = (num[15][y, x] + num[16][y, x] + num[17][y, x] + num[18][y, x]
                                    + num[19][y, x]) / 4
    capital[z] = capital[z].ravel()

# 2D Array in 1D Array für Bravais-Pearson Methode
for x in range(0, 20):
    num[x] = num[x].ravel()

# Korrelationskoeffizient berechnet
r, p = scipy.stats.pearsonr(num[0], num[0])
print("r:", r, "p:", p)
r, p = scipy.stats.pearsonr(num[0], capital[0])
print("r:", r, "p:", p)
r, p = scipy.stats.pearsonr(num[0], capital[0])
print("r:", r, "p:", p)
```

Listing 4.6: Windowing und Bravais-Pearson Methode

```python
# ----------------------------------------
# Task 2.3
# ----------------------------------------

from scipy import signal
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats

# Definieren der Dateinamen
num = ["hoch1", "hoch2", "hoch3", "hoch4", "hoch5", "tief1", "tief2", "tief3", "tief4", "tief5", "links1",
    "links2", "links3", "links4", "links5", "rechts1", "rechts2", "rechts3", "rechts4", "rechts5"]
anderer = ["ahoch1", "ahoch2", "ahoch3", "ahoch4", "ahoch5", "atief1", "atief2", "atief3", "atief4", "atief5",
        "alinks1", "alinks2", "alinks3", "alinks4", "alinks5", "arechts1", "arechts2", "arechts3", "arechts4",
        "arechts5"]
moi = ["mhoch1", "mhoch2", "mhoch3", "mhoch4", "mhoch5", "mtief1", "mtief2", "mtief3", "mtief4",
    "mtief5", "mlinks1", "mlinks2", "mlinks3", "mlinks4", "mlinks5", "mrechts1", "mrechts2", "mrechts3",
    "mrechts4", "arechts5"]
capital = ["hoch", "tief", "links", "rechts"]
capital2 = ["hoch", "tief", "links", "rechts"]

# Gaußfenster definieren mit Fensterbreite Standardabweichung 4
gaussianwindow = signal.windows.gaussian(512, std=4)

# For loop um alle Dateien zu analysieren
for a in range(0, 20):
    # Einlesen der Numpy Dateien von Person 1
    data = np.load('data/' + str(num[a]) + '.npy')
    # Einlesen der Numpy Dateien für Person 2
    data2 = np.load('data/' + str(anderer[a]) + '.npy')
    # Einlesen der zweiten Numpy Dateien für Person 1
    data3 = np.load('data/' + str(moi[a]) + '.npy')
    # Definieren eines leeren Vectors für Person 1
    num[a] = np.zeros((171, 512))
    # Definieren eines leeren Vectors für Person 1
    moi[a] = np.zeros((171, 512))
    # Definieren eines leeren Vectors für Person 2
    anderer[a] = np.zeros((171, 512))
    z = 256

    # For loop um die einzelnen Windows zu erstellen
```

```
    for y in range(0, 171):
        z = z − 256
        # For loop um die einzelnen Frames zu berechnen
        for x in range(0, 512):
            # Signale ∗ Gaußfenster, das wiederrum wird absolut fouriertransformiert.
            # Daraus der Durchschnitt ergibt den Windowingwert
            num[a][y, x] = np.mean(np.abs(np.fft.fft(data[z] ∗ gaussianwindow)))
            anderer[a][y, x] = np.mean(np.abs(np.fft.fft(data2[z] ∗ gaussianwindow)))
            moi[a][y, x] = np.mean(np.abs(np.fft.fft(data3[z] ∗ gaussianwindow)))
            z = z + 1

    # For loop um die einzelnen Windows zu erstellen
    for y in range(0, 171):
        # For loop um die einzelnen Frames zu berechnen
        for x in range(0, 512):
            # Signale ∗ Gaußfenster, das wiederrum wird absolut fouriertransformiert.
            # Daraus der Durchschnitt ergibt den Windowingwert
            num[a][y, x] = num[a][y, x] ∗ gaussianwindow[x]
            anderer[a][y, x] = anderer[a][y, x] ∗ gaussianwindow[x]
            moi[a][y, x] = moi[a][y, x] ∗ gaussianwindow[x]
        num[a][y] = np.abs(np.fft.fft(num[a][y]))
        anderer[a][y] = np.abs(np.fft.fft(anderer[a][y]))
        moi[a][y] = np.abs(np.fft.fft(moi[a][y]))
        num[a][y] = np.mean(num[a][y])
        anderer[a][y] = np.mean(anderer[a][y])
        moi[a][y] = np.mean(moi[a][y])

# For loop zur Ausgabe der endgültig berechneten Plots
for z in range(0, 4):
    # Vektoren zum Speichern der Plots
    capital[z] = np.zeros((171, 512))
    # For loop für die einzelnen Windows
    for y in range(0, 171):
        # For loop für die einzelnen Samples
        for x in range(0, 512):
            # Hoch Mittelung
            if (z == 0):
                capital[z][y, x] = (num[0][y, x] + num[1][y, x] + num[2][y, x] + num[3][y, x]
                                   + num[4][y, x]) / 4
            # Tief Mittelung
            elif (z == 1):
                capital[z][y, x] = (num[5][y, x] + num[6][y, x] + num[7][y, x] + num[8][y, x]
```

```
                                    + num[9][y, x]) / 4
            # Links Mittelung
            elif (z == 2):
                capital[z][y, x] = (num[10][y, x] + num[11][y, x] + num[12][y, x] + num[13][y, x]
                                    + num[14][y, x]) / 4
            # Rechts Mittelung
            elif (z == 3):
                capital[z][y, x] = (num[15][y, x] + num[16][y, x] + num[17][y, x] + num[18][y, x]
                                    + num[19][y, x]) / 4
    # 2D Array in 1D Array für Bravais-Pearson Methode
    capital[z] = capital[z].ravel()

# 2D Array in 1D Array für Bravais-Pearson Methode
for x in range(0, 20):
    num[x] = num[x].ravel()
    moi[x] = moi[x].ravel()
    anderer[x] = anderer[x].ravel()

# Korrelationskoeffizient berechnet
r1, p = scipy.stats.pearsonr(capital[0], moi[0])
print("capital-moi1 r:", r1, "p:", p)
r2, p = scipy.stats.pearsonr(capital[0], moi[1])
print("capital-moi2 r:", r2, "p:", p)
r3, p = scipy.stats.pearsonr(capital[0], moi[2])
print("capital-moi3 r:", r3, "p:", p)
r4, p = scipy.stats.pearsonr(capital[0], moi[3])
print("capital-moi4 r:", r4, "p:", p)
r5, p = scipy.stats.pearsonr(capital[0], moi[4])
print("capital-moi5 r:", r5, "p:", p)

s1, p = scipy.stats.pearsonr(capital[0], anderer[0])
print("capital-anderer1 r:", s1, "p:", p)
s2, p = scipy.stats.pearsonr(capital[0], anderer[1])
print("capital-anderer2 r:", s2, "p:", p)
s3, p = scipy.stats.pearsonr(capital[0], anderer[2])
print("capital-anderer3 r:", s3, "p:", p)
s4, p = scipy.stats.pearsonr(capital[0], anderer[3])
print("capital-anderer4 r:", s4, "p:", p)
s5, p = scipy.stats.pearsonr(capital[0], anderer[4])
print("capital-anderer5 r:", s5, "p:", p)

print()
```

```python
r1 = (r1 + r2 + r3 + r4 + r5) / 5
print(r1)
s1 = (s1 + s2 + s3 + s4 + s5) / 5
print(s1)
print()
# --------------------------------------------

r6, p = scipy.stats.pearsonr(capital[1], moi[5])
print("capital-moi1 r:", r6, "p:", p)
r7, p = scipy.stats.pearsonr(capital[1], moi[6])
print("capital-moi2 r:", r7, "p:", p)
r8, p = scipy.stats.pearsonr(capital[1], moi[7])
print("capital-moi3 r:", r8, "p:", p)
r9, p = scipy.stats.pearsonr(capital[1], moi[8])
print("capital-moi4 r:", r9, "p:", p)
r10, p = scipy.stats.pearsonr(capital[1], moi[9])
print("capital-moi5 r:", r10, "p:", p)

s6, p = scipy.stats.pearsonr(capital[1], anderer[5])
print("capital-anderer1 r:", s6, "p:", p)
s7, p = scipy.stats.pearsonr(capital[1], anderer[6])
print("capital-anderer2 r:", s7, "p:", p)
s8, p = scipy.stats.pearsonr(capital[1], anderer[7])
print("capital-anderer3 r:", s8, "p:", p)
s9, p = scipy.stats.pearsonr(capital[1], anderer[8])
print("capital-anderer4 r:", s9, "p:", p)
s10, p = scipy.stats.pearsonr(capital[1], anderer[9])
print("capital-anderer5 r:", s10, "p:", p)

print()
r1 = (r6 + r7 + r8 + r9 + r10) / 5
print(r1)
s1 = (s6 + s7 + s8 + s9 + s10) / 5
print(s1)
print()
# --------------------------------------------

r11, p = scipy.stats.pearsonr(capital[2], moi[10])
print("capital-moi1 r:", r11, "p:", p)
r12, p = scipy.stats.pearsonr(capital[2], moi[11])
print("capital-moi2 r:", r12, "p:", p)
r13, p = scipy.stats.pearsonr(capital[2], moi[12])
```

```
print("capital−moi3 r:", r13, "p:", p)
r14, p = scipy.stats.pearsonr(capital[2], moi[13])
print("capital−moi4 r:", r14, "p:", p)
r15, p = scipy.stats.pearsonr(capital[2], moi[14])
print("capital−moi5 r:", r15, "p:", p)

s11, p = scipy.stats.pearsonr(capital[2], anderer[10])
print("capital−anderer1 r:", s11, "p:", p)
s12, p = scipy.stats.pearsonr(capital[2], anderer[11])
print("capital−anderer2 r:", s12, "p:", p)
s13, p = scipy.stats.pearsonr(capital[2], anderer[12])
print("capital−anderer3 r:", s13, "p:", p)
s14, p = scipy.stats.pearsonr(capital[2], anderer[13])
print("capital−anderer4 r:", s14, "p:", p)
s15, p = scipy.stats.pearsonr(capital[2], anderer[14])
print("capital−anderer5 r:", s15, "p:", p)

print()
r1 = (r11 + r12 + r13 + r14 + r15) / 5
print(r1)
s1 = (s11 + s12 + s13 + s14 + s15) / 5
print(s1)
print()
# −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

r16, p = scipy.stats.pearsonr(capital[3], moi[15])
print("capital−moi1 r:", r16, "p:", p)
r17, p = scipy.stats.pearsonr(capital[3], moi[16])
print("capital−moi2 r:", r17, "p:", p)
r18, p = scipy.stats.pearsonr(capital[3], moi[17])
print("capital−moi3 r:", r18, "p:", p)
r19, p = scipy.stats.pearsonr(capital[3], moi[18])
print("capital−moi4 r:", r19, "p:", p)
r20, p = scipy.stats.pearsonr(capital[3], moi[19])
print("capital−moi5 r:", r20, "p:", p)

s16, p = scipy.stats.pearsonr(capital[3], anderer[15])
print("capital−anderer1 r:", s16, "p:", p)
s17, p = scipy.stats.pearsonr(capital[3], anderer[16])
print("capital−anderer2 r:", s17, "p:", p)
s18, p = scipy.stats.pearsonr(capital[3], anderer[17])
print("capital−anderer3 r:", s18, "p:", p)
```

```
s19, p = scipy.stats.pearsonr(capital[3], anderer[18])
print("capital−anderer4 r:", s19, "p:", p)
s20, p = scipy.stats.pearsonr(capital[3], anderer[19])
print("capital−anderer5 r:", s20, "p:", p)

print()
r1 = (r16 + r17 + r18 + r19 + r20) / 5
print(r1)
s1 = (s16 + s17 + s18 + s19 + s20) / 5
print(s1)
print()
```

Listing 4.7: Bravais-Pearson Methode mit Ausgabe der Korrelation