Hochschule Konstanz
Technik, Wirtschaft und Gestaltung

Signale, Systeme und Sensoren

Aufbau eines einfachen Spracherkenners

Gina Kokoska, Simon Winter

Konstanz, 9. Januar 2022

Zusammenfassung (Abstract)

Thema:	Aufbau eines einfachen Spracherkenners	
Autoren:	Gina Kokoska	gi161kok@htwg-konstanz.de
	Simon Winter	si411win@htwg-konstanz.de
Betreuer:	Prof. Dr. Matthias O. Franz	mfranz@htwg-konstanz.de
	Jürgen Keppler	juergen.keppler@htwg-konstanz.de
	Daniel Castelo	dacastel@htwg-konstanz.de

In diesem Versuch wird ein einfacher Spracherkenner erstellt, der z.B. zur Steuerung eines Staplers in einem Hochregallager dienen könnte. Hierfür werden die Steurungsbefehle "Hoch", "Tief", "Links" und "Rechts" aufgenommen. Zur Erkennung der Befehlswörter werden die jeweiligen Spektren mit der Windowing-Methode berechnet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Kapitel 1

Einleitung

Diese Versuchsreihe stellt mit Hilfe der Windowing-Methode Referenzspektren der aufgenommenen Steuerbefehle dar. Anschliessend wird der Datensatz mit Steurbefehlen eines anderen Sprechers verglichen. Zur Vergleichsauswertung wird die Korrelation beider Spektren berrechnet. Eine starke Korrelation ist als erfolgreiche Spracherkennung zu deuten.

[1] [2]

Kapitel 2

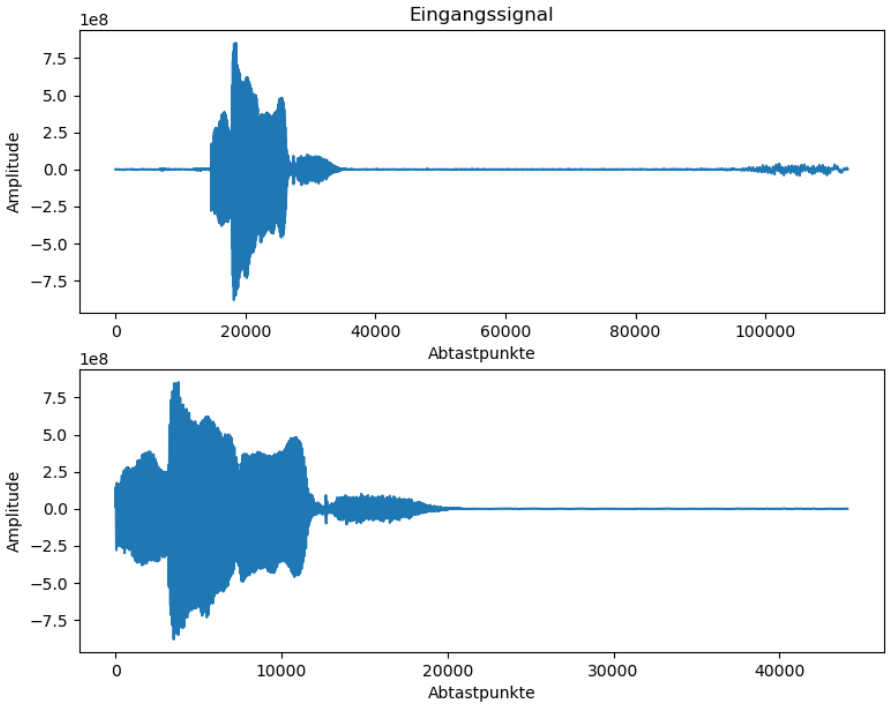
Versuch 1

2.1 Fragestellung, Messprinzip, Aufbau, Messmittel

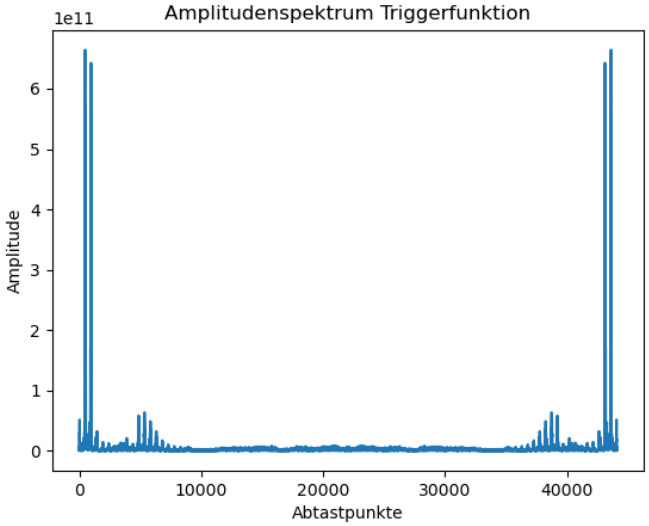
Es wird ein Python-Programm zur Aufnahme akustischer Signale verfasst, welches die aufgenommene Sequenz als wave-Datei und als Plot speichert. Damit die Aufnahmen im Folge versuch verglichen werden können, wird die Aufnahmezeit auf eine Sekunde beschränkt, mit Bedingungen einer Aplitudenüberschreitung von 150000000. Das getriggerte Signal wird auch als wave-Datei gespeichert und graphisch dargestellt. Anschliessend wird mit Hilfe der Fouriertransformation das Amplitudenspektrum des getriggerten Signals berechnet und dargestellt. Zuletzt wird das Signal in Samples der Länge 512 zerlegt, die sich zur Hälfte überlappen sollen. Jedes Fenster wird mit einer gaußschen Fensterfunktion gewichtet, mit entsprechender Fensterbreite von 4 Standardabweichungen. Erst wird eine lokale Fouriertransformation auf die einzelnen Fenster angewendet. Anschliessend wird die gemittelte Fouriertransformation aller Fenster berechnet. Das Amplitudenspektrum wird berechnet und mit dem getriggertem Spektrum verglichen. Das verwendete Kleinmembran-Kondensatormikrofon ist in ein Callcenter-Headset eingebaut, welches Noise-cancelling verwendet. Es verwendet einen elektrostatischen Wandler, um Luftschall in eine elektrische Spannung um zu wandeln. Es nutzt das Prinzip der Kapazität, in welchem der Abstand einer leitfähigen Membran zur Gegenelektrode gemessen wird. Hierbei wird eine polarisierte Kapsel bestehend aus einer elastischen Membran und einer Gegenelektrode verwendet. Diese bilden die Platten eines Kondensators der Gleichspannung als elektrische Kapazität speichern kann. Auftreffender Schall bringt die Membran zum Schwingen, wodurch sich der Abstand zwischen Membran und Gegenelektrode und damit auch die Kapazität des Kondensators verändert. Je geringer der Abstand desto grösser der eintreffende Schall (antiproportionales Signal). Als Messmittel wird ein Plantronics EncorePro HW725 Headset verwendet.

2.2 Messwerte

Das Programm lieferte folgendes original (oben) und getriggertes (unten) Signal:

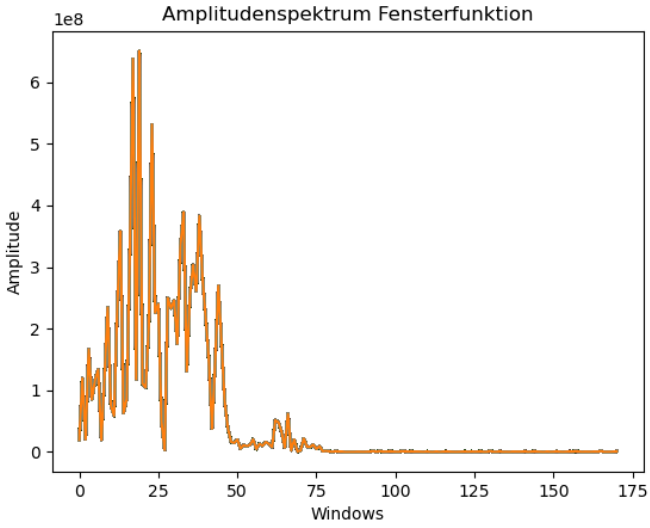


Das Programm lieferte folgende Amplitudenspektrum:



2.3 Auswertung

Das Windowing lieferte folgendes Amplitudenspektrum:



2.4 Interpretation

Schaut man sich die Plots des Eingangssignals sowie des getriggerten Eingangsignals an, so erkennt man die erfolgreiche Zuschneidung des Signals. Anhand der x-Achse kann man die Abtastpunkte auslesen, welche für das getriggerte Signal genau eine Sekunde ergeben. Die Amplitudenüberschreitung ist so eingestellt, das ein eingehendes Signal frühzeitig erkannt wird, ohne das etwas vom gewollten Signal abgeschnitten wird. Vergleicht man das Amplitudenspektrum der Fensterfunktion mit dem graphischen Plot des zugeschnittenen Eingangssignals, so lässt sich eindeutig erkennen, das sich beide Plots in ihrer Form sehr ähnlich sehen. Somit kann man das Amplitudenspektrum der Fensterfunktion eindeutig dem Signal des Eingangssignals zuordnen.

Kapitel 3

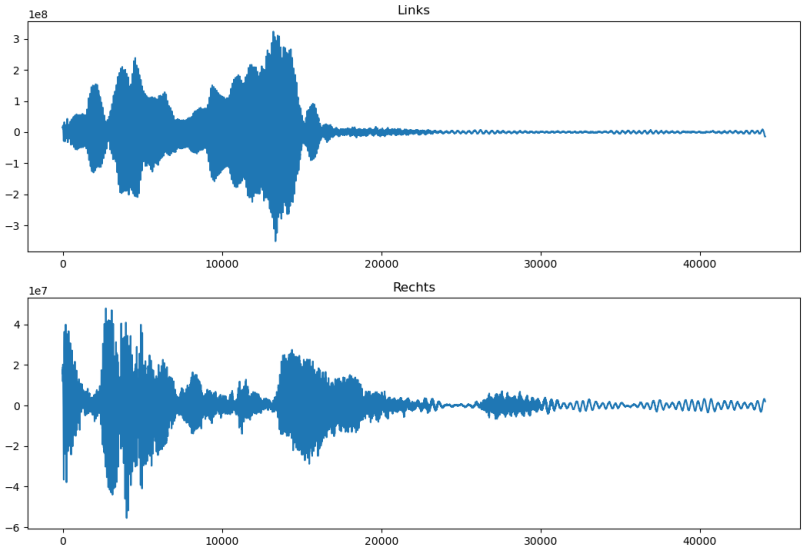
Versuch 2

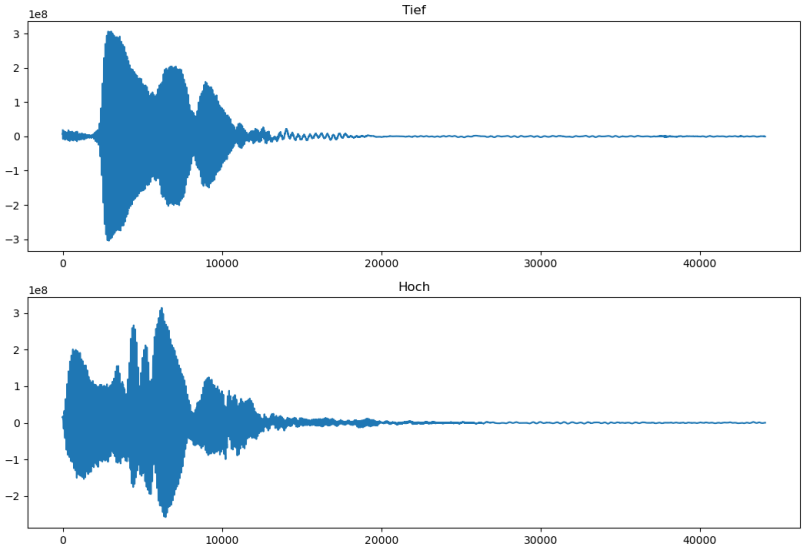
3.1 Fragestellung, Messprinzip, Aufbau, Messmittel

Für den Spracherkenner werden jeweils 5 Referenzspektren der Steuerungsbeefehle ”Hoch”, ”Tief”, ”Links” und ”Rechts” mit Hilfe des Programms aus Aufgabe 1 aufgenommen, gespeichert und verglichen. Der Sprecher ist für alle Aufnahmen der selbe. Das gemittelte Referenzspektrum jedes Befehls wird graphisch dargestellt. Anschliessend wird ein Testdatensatz eines anderen Sprechers und des original Sprechers erstellt. Beide Datensätze werden durch Berechnung der Korrelationskoeffizienten nach Bravais-Pearson mit einander verglichen. Es ist eine Korrelation nahe 1 bei den selben Befehlswörter zu erwarten, da die identischer Wörter Spektren sehr ähnlich sein sollten. Abweichungen sollten durch Messfehler wie unterschiedliche Sprechweisen erklärbar sein. Zuletzt wird der Spracherkenner anhand der in der Vorlesung behandelten Architektur implementiert. Beide Datensätze werden nochmals mit einander verglichen, um die prozentual erfolgreiche Erkennungsrate und falsch erkannte Fehlerrate zu berechnen. Als Messmittel wird das Headset aus Versuch 1 verwendet.

3.2 Messwerte

Das Programm lieferte folgende Referenzspektren:





3.3 Auswertung

Die Vergleiche zwischen unseren Referenzspektren und unseren Eingabespektren ergeben folgende Werte:

Korrelation Datensatz(Ref) mit original Sprecher G

korrkoefLLG = sst.pearsonr(LRef, LG)[0] = 0.024579713287452655
korrkoefLRG = sst.pearsonr(LRef, RG)[0] = 0.015589028822441266
korrkoefLTG = sst.pearsonr(LRef, TG)[0] = 0.011521206726410545
korrkoefLHG = sst.pearsonr(LRef, HG)[0] = -0.027171988057656 72

Erkanntes Wort: Links

Korrelation Datensatz(Ref) mit zweitem Sprecher E

korrkoefLLE = sst.pearsonr(LRef, LE)[0] = 0.017434312307556363
korrkoefLRE = sst.pearsonr(LRef, RE)[0] = 0.00036481785038843954
korrkoefLTE = sst.pearsonr(LRef, TE)[0] = -0.003291010877873985
korrkoefLHE = sst.pearsonr(LRef, HE)[0] = 0.0004718242888293782

Erkanntes Wort: Links

3.4 Interpretation

Die positive, aber geringe Korrelation lässt auf einen funktionierenden Vergleich zwischen Referenzspektrum und Eingabespektrum schliessen. Der höchste, wenn auch immer noch schwach korrellierende Wert, wurde bei dem Vergleich Links - Links erkannt.

Eine weitere positive Korrelation zu einem unterschiedlichen Wort wurde festgestellt, diese ist jedoch ca um die hälfte geringer. Zudem wurde auch eine leicht negative Korrelation errechnet. Der höchste gemessene Korrelationswert wurde bei dem Originalsprecher erreicht, was aufgrund des Referenzspektrums natürlich Sinn macht. Der zweithöchste Wert wurde bei dem Wortvergleich Links - Links beim zweiten Sprecher errechnet, demnach bezeichnen wir unseren Spracherkenner als funktionierend. Die allgemein geringeren Korrelationswerte erklären wir uns mit der Aufnahmequaltität des Mikrofons sowie dem eingebau-

ten, automatischen Noise-Cancelling. Wir gehen davon aus, dass bei anderen Mikrofonen stärker positve Korrelationskoeffizienten errechnet werden würden. Es ist anzunehmen, dass der Spracherkenner funktioniert, da der Originalsprecher sowie der Fremdsprecher weiblich waren.

Anhang

A.1 Quellcode

A.1.1 Quellcode Versuch 1a und 1b

```
import pyaudio
import numpy as np


FORMAT = pyaudio.paInt16
SAMPLEFREQ = 44100 # 44 kbps
FRAMESIZE = 1024 # each buffer contains 1024 samples
NOFFRAMES = 220
PATH = "eingangssignal_raw"
PATH_T = "referenzdatensatz"
PATH_W = "eingangssignal_win"
PATH_G = "sprechergina"
PATH_L = "sprecherluca"

p = pyaudio.PyAudio()
print('running')
# 1a open stream
stream = p.open(format=FORMAT,channels=1,rate=SAMPLEFREQ,input=True,
    frames_per_buffer=FRAMESIZE)
data = stream.read(NOFFRAMES*FRAMESIZE)
decoded = np.fromstring(data, np.int)
stream.stop_stream()
stream.close()
p.terminate()

# 1b Triggerfunktion auf decoded angewendet
for i in range(len(decoded)):
    if decoded[i] > 15000000:
```

```
        start = i
        break

decodedTriggered = decoded[start:start+SAMPLEFREQ]

def safeTriggeredAsCSV(eingangssignal):
    csvTrigArray = np.savetxt(PATH_T + "/decodedtrigL1.csv",
    eingangssignal, delimiter=",")
    return csvTrigArray


safeTriggeredAsCSV(decodedTriggered)

print('done')
```

A.1.2 Quellcode Versuch 1c und 1d

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

FORMAT = pyaudio.paInt16
SAMPLEFREQ = 44100 # 44 kbps
FRAMESIZE = 1024 # each buffer contains 1024 samples
NOFFRAMES = 220
PATH = "eingangssignal_raw"
PATH_T = "referenzdatensatz"
PATH_W = "eingangssignal_win"


# oeffne stream
p = pyaudio.PyAudio()
print('running')

stream = p.open(format=FORMAT,channels=1,rate=SAMPLEFREQ,input=True,
    frames_per_buffer=FRAMESIZE)
data = stream.read(NOFFRAMES*FRAMESIZE)
decoded = np.fromstring(data, np.int)
stream.stop_stream()
stream.close()
p.terminate()

# plot decoded und decodedTriggered in einer Abb
# plt decodedTriggered in line 75
plt.figure(figsize=(9, 7))
plt.subplot(211)
xvalue = np.arange(0, len(decoded))
plt.plot(decoded)
plt.title("Grundperiode")
plt.xlabel("Zeit in s")
plt.ylabel("Amplitude")


print('done')


```

```
# 1b Triggerfunktion auf decoded angewendet
for i in range(len(decoded)):
    if decoded[i] > 15000000:
        start = i
        break

decodedTriggered = decoded[start:start+SAMPLEFREQ]


# window ueber alle referenz daten
windownmr = 171

windows = np.zeros((windownmr, 512))

schnitt = 256

gauss = signal.windows.gaussian(512, 4)


for y in range(0, windownmr):
    schnitt = schnitt - 256
    for x in range(0, 512):
        windows[y, x] = np.mean(np.abs(np.fft.fft(decodedTriggered[
   schnitt] * gauss)))
        schnitt = schnitt + 1

for y in range(0, windownmr):
    for x in range(0, 512):
        windows[y] = np.mean(windows[y, x])




# plot decodedTriggerd
# save untriggerd vgl mit triggerd
deltaT = 1 / SAMPLEFREQ
xvalue = np.arange(start, start+SAMPLEFREQ) * deltaT
plt.subplot(212)
plt.plot(decoded[start:start+SAMPLEFREQ])
plt.xlabel("Abtastpunkte")
plt.ylabel("Amplitude")
plt.savefig('untriggeredANDtriggered.png')
```

```
81
82
83 #1c fft auf triggered Signal fuer amplitudenspektrum
84 ffttriggerd = np.fft.fft(decodedTriggered)
85 specTriggered = np.abs(ffttriggerd)
86 plt.figure()
87 plt.plot(specTriggered)
88 plt.title('Amplitudenspektrum Triggerfunktion')
89 plt.xlabel('Abtastpunkte')
90 plt.ylabel('Amplitude')
91 plt.savefig('triggeredAmplitudenspektrum.png')
92
93 # fft auf eingangsignal fuer amplitudenspektrum
94 fft = np.fft.fft(decoded)
95 spec = np.abs(fft)
96 plt.figure()
97 plt.plot(spec)
98 plt.title('Amplitudenspektrum')
99 plt.xlabel('Abtastpunkte')
100 plt.ylabel('Amplitude')
101 plt.savefig('ungetriggeredAmplitudenspektrum.png')
102
103
104 # plot fuer windowed funktion
105 plt.figure()
106 plt.plot(windows)
107 plt.title('Amplitudenspektrum Fensterfunktion')
108 plt.xlabel('Abtastpunkte')
109 plt.ylabel('Amplitude')
110 plt.savefig('fensterAmplitudenspektrum.png')
```

A.1.3 Quellcode Versuch 2

```
import numpy as np
from scipy import signal

PATH_W = "windowreferenzdatensatz"
PATH_R = "refspektren"


# addition aller 5 aufnahmen, mit anschliessender mittelung
LRef = (np.genfromtxt(PATH_R + "RefWinL1.csv")
+ np.genfromtxt(PATH_R + "RefWinL2.csv")
+ np.genfromtxt(PATH_R + "RefWinL3.csv")
+ np.genfromtxt(PATH_R + "RefWinL4.csv")
+ np.genfromtxt(PATH_R + "RefWinL5.csv")) / 5

# addition aller 5 aufnahmen, mit anschliessender mittelung
RRef = (np.genfromtxt(PATH_R + "RefWinR1.csv")
+ np.genfromtxt(PATH_R + "RefWinR2.csv")
+ np.genfromtxt(PATH_R + "RefWinR3.csv")
+ np.genfromtxt(PATH_R + "RefWinR4.csv")
+ np.genfromtxt(PATH_R + "RefWinR5.csv")) / 5

# addition aller 5 aufnahmen, mit anschliessender mittelung
TRef = (np.genfromtxt(PATH_R + "RefWinT1.csv")
+ np.genfromtxt(PATH_R + "RefWinT2.csv")
+ np.genfromtxt(PATH_R + "RefWinT3.csv")
+ np.genfromtxt(PATH_R + "RefWinT4.csv")
+ np.genfromtxt(PATH_R + "RefWinT5.csv")) / 5

# addition aller 5 aufnahmen, mit anschliessender mittelung
HRef = (np.genfromtxt(PATH_R + "RefWinH1.csv")
+ np.genfromtxt(PATH_R + "RefWinH2.csv")
+ np.genfromtxt(PATH_R + "RefWinH3.csv")
+ np.genfromtxt(PATH_R + "RefWinH4.csv")
+ np.genfromtxt(PATH_R + "RefWinH5.csv")) / 5

# speichern der gemittelten referenzspektren
def safeRefWAsCSV(eingangssignal):
    csvRefArray = np.savetxt(PATH_R + "/decodedtrigR1.csv",
    eingangssignal, delimiter=",")
    return csvRefArray
```

```
40
41 safeRefWAsCSV(LRef)
```

```
import matplotlib.pyplot as plt
import numpy as np



# plotten der referenz spektren
PATH_R = "refspektren"

LRef =  np.genfromtxt(PATH_R + "/LRef.csv")
RRef =  np.genfromtxt(PATH_R + "/RRef.csv")
HRef =  np.genfromtxt(PATH_R + "/HRef.csv")
TRef =  np.genfromtxt(PATH_R + "/TRef.csv")

fig1, axs = plt.subplots(2)
axs[0].plot(LRef)
axs[0].set_title("Links")
axs[1].plot(RRef)
axs[1].set_title("Rechts")


fig2, axs = plt.subplots(2)
axs[0].plot(HRef)
axs[0].set_title("Hoch")
axs[1].plot(TRef)
axs[1].set_title("Tief")

plt.show()
```

```
import numpy as np
import scipy.stats as sst


PATH_R = "refspektren"
PATH_G = "sprechergina"
PATH_E = "sprecherelena"

# Referenzdatensatz
LRef = np.genfromtxt(PATH_R + "/LRef.csv")
RRef = np.genfromtxt(PATH_R + "/RRef.csv")
TRef = np.genfromtxt(PATH_R + "/TRef.csv")
HRef = np.genfromtxt(PATH_R + "/HRef.csv")

# selber sprecher wie referenz
LG = np.genfromtxt(PATH_G + "/winL.csv")
RG = np.genfromtxt(PATH_G + "/winR.csv")
TG = np.genfromtxt(PATH_G + "/winT.csv")
HG = np.genfromtxt(PATH_G + "/winH.csv")

# anderer sprecher wie referenz
LE = np.genfromtxt(PATH_E + "/winL.csv")
RE = np.genfromtxt(PATH_E + "/winR.csv")
TE = np.genfromtxt(PATH_E + "/winT.csv")
HE = np.genfromtxt(PATH_E + "/winH.csv")


# korrelation datensatz mit selben sprecher
korrkoefLLG = sst.pearsonr(LRef, LG)[0]
korrkoefLRG = sst.pearsonr(LRef, RG)[0]
korrkoefLTG = sst.pearsonr(LRef, TG)[0]
korrkoefLHG = sst.pearsonr(LRef, HG)[0]

# korrelation datensatz mit anderem sprecher
korrkoefLLE = sst.pearsonr(LRef, LE)[0]
korrkoefLRE = sst.pearsonr(LRef, RE)[0]
korrkoefLTE = sst.pearsonr(LRef, TE)[0]
korrkoefLHE = sst.pearsonr(LRef, HE)[0]


# berechne welches wort am besten erkannt wird original sprecher
maxValG = max(max(korrkoefLLG, korrkoefLRG),max(korrkoefLTG, korrkoefLHG)
```

```
    )
43
44
45 # berechne welches wort am besten erkannt wird anderer sprecher
46 maxValE = max(max(korrkoefLLE, korrkoefLRE),max(korrkoefLTE, korrkoefLHE)
    )
47
48
49 print(korrkoefLTG)
50 print(korrkoefLLG)
51 print(korrkoefLHG)
52 print(korrkoefLRG)
53 print(maxValG)
54
55 print(korrkoefLTE)
56 print(korrkoefLLE)
57 print(korrkoefLHE)
58 print(korrkoefLRE)
59 print(maxValE)
```

Literaturverzeichnis

[1] Franz, Prof. Dr. Matthias O.: *Vorlesung 10 - Sprache und Spracherkennung: Kurzzeit-Fouriertransformation, Erzeugung und Wahrnehmung von Sprache, Mustererkennung durch Korrelation*. In: *Vorlesung Technische Grundlagen der angewandten Informatik*, 2016.

[2] Franz, Prof. Dr. Matthias O.: *Vorlesung 1 - Einführung*. In: *Vorlesung Technische Grundlagen der angewandten Informatik*, 2016.