HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES

**Signale, Systeme und Sensoren**

# Aufbau eines einfachen Spracherkenners

**M. Kieser, J. Altmeyer**

**Konstanz, 12. Dezember 2015**

# Zusammenfassung (Abstract)


| | | |
|---|---|---|
| Thema: | Aufbau eines einfachen Spracherkenners | |
| Autoren: | M. Kieser | makieser@htwg-konstanz.de |
| | J. Altmeyer | jualtmey@htwg-konstanz.de |
| Betreuer: | Prof. Dr. Matthias O. Franz | mfranz@htwg-konstanz.de |
| | Jürgen Keppler | juergen.keppler@htwg-konstanz.de |
| | Martin Miller | martin.miller@htwg-konstanz.de |



Erstellung eines einfachen Spracherkenners mit dem Prinzip des Prototyp-Klassifikators. Weiterhin wird das Windowing Verfahren behandelt und implementiert. Dieses wird für die Ermittlung der Spektren für den Spracherkenner benötigt.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Listingverzeichnis

# 1

# Einleitung

Im nachfolgenden Versuch geht es um die Erstellung eines einfachen Spracherkenners. Dieser wird nach dem Prinzip des Prototyp-Klassifikators umgesetzt. Die zu unterscheidenden Prototypen sind dabei "Hoch", "Tief", "Links", und "Rechts". Der Spracherkenner vergleicht diese im voraus aufgenommenen Prototypen anhand ihres Spektrums mit dem gesprochenen Wort des Benutzers. Bei jedem gesprochenen Wort entscheidet der Spracherkenner wie stark das Wort mit einem der Prototypen korreliert. Der Spracherkenner wählt den Prototypen, wo es zu der größten Übereinstimmung kommt.
Für den Vergleich der Spektren benötigt man die Windowing Methode, welche in Kapitel 2 vorbereitend behandelt wird. Der eigentliche Spracherkenner wird in Kapitel 3 erstellt.

# 2

# Versuch 1- Fourieranalyse lang andauernder Signale

Lang andauernde Signale sind für die Fourieranalyse sehr rechenintensiv. Deshalb bedient man sich der Methode des Windowing, welches nach Franz2015 [1] das Signal in oft überlappende Fenster zerlegt. Diese Zerlegung ermöglicht eine lokale Fourieranalyse eines Fensters. Diese Analyse ist wesentlich schneller durchgeführt, da man nur einen Teil des Signals betrachtet. In dem nachfolgenden Versuch wird die Windowing Methode implementiert und anschließend auf ihre Korrektheit anhand eines Beispiel Signals geprüft.

## 2.1 Fragestellung, Aufbau, Messprinzip, Messmittel

**Fragestellung:** Ist das Amplitudenspektrum eines Audiosignals durch die Windowing Methode richtig dargestellt.
**Aufbau:** Der Antwort auf diese Frage wird sich in 5 Schritten genähert:

a) automatisierte Aufnahme eines Audiosignals und dessen Speicherung
b) Aufbereitung des Audiosignals durch Triggerung und Zeitangleichung
c) Erstellung des Amplitudenspektrums des Audiosignals
d) Anwendung des Windowing-Verfahrens auf das Audiosignal
e) Vergleich der Beiden Amplitudenspektren in Kapitel 2.4

**Messprinzip und Messmittel:** Es wird ein Mikrofon als Messmittel eingesetzt. Das Mikrofon erzeugt je nach empfangenen Ton einen entsprechenden Spannungswert. Mithilfe dieses Messprinzips lassen sich die Informationen weiterverarbeiten.

## 2.2 Messwerte

Nachfolgend die Messwerte zu den Schritten a) und b):

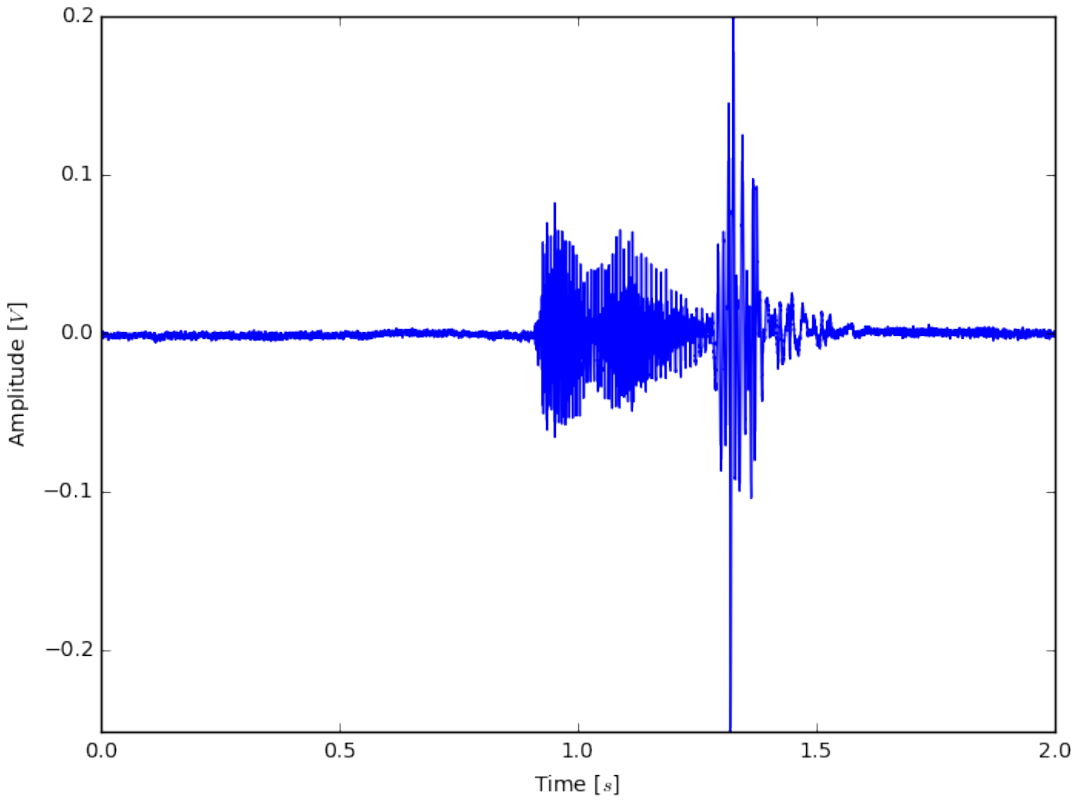


Abbildung 2.1: **a)** unbearbeitetes Audiosignal des Wortes «Aua»

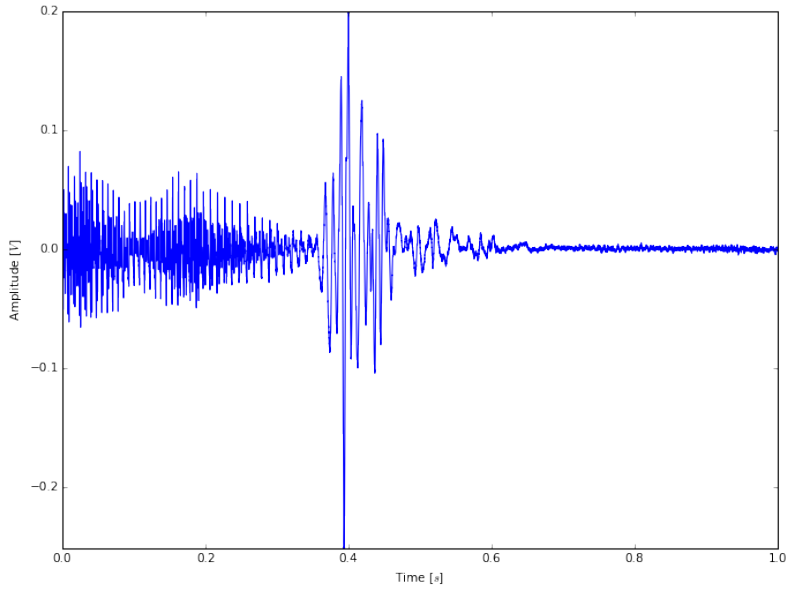


Abbildung 2.2: **b)** aufbereitetes Audiosignal aus a), getriggert und eine Sekunde lang

## 2.3 Auswertung

**Schritt c)**

Aus dem aufbereiteten Audiosignal aus Abbildung 2.2 wird mittels der Fouriertransformation das Amplitudenspektrum in Abbildung 2.3 erzeugt. Dabei wird das ganze Signal komplett betrachtet. Die Berechnung der Ergebnisse sind dem Pythoncode in Listing 4.1 zu entnehmen.

**Schritt d)**

Verglichen zu dem in Schritt c) berechneten Amplitudenspektrum, wird nun das Amplitudenspektrum aus mehreren Teilbereichen mittels der Windowing Methode zusammengestellt. Die Zusammenführung des Signals entsteht durch Mittlung der einzelnen Teilbereiche. Das Ergebnis ist in Abbildung 2.4 dargestellt. Die Berechnung des Windowing sind speziell der Methode «windowing(rec)» aus Listing 4.1 zu entnehmen.

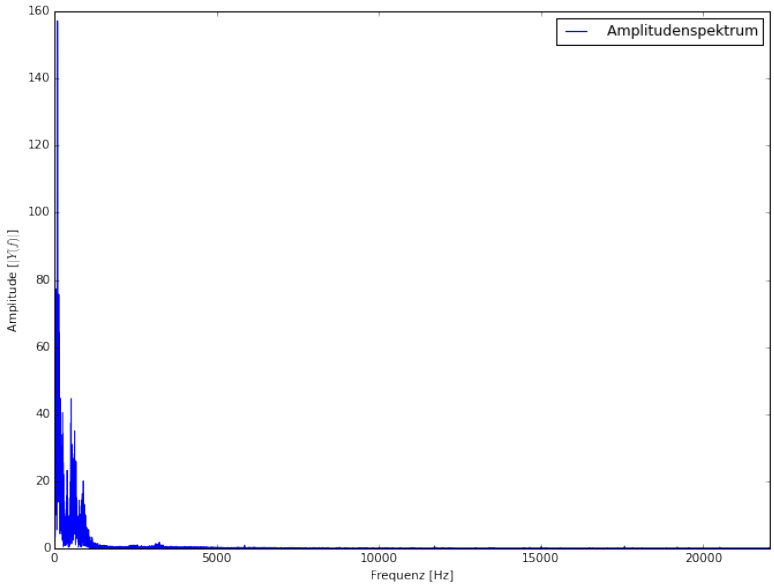


Abbildung 2.3: **c)** Amplitudenspektrum des aufbereiteten Audiosignals

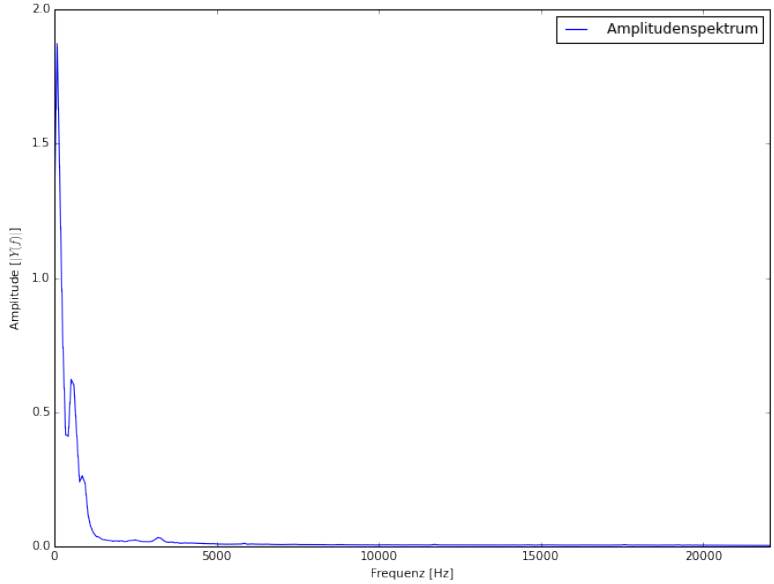


Abbildung 2.4: **d)** Amplitudenspektrum nach Windowing des aufbereiteten Audiosignals

## 2.4 Interpretation

In Abbildung 2.4 sieht man einen geglätteten Verlauf des in Abbildung 2.3 gezeigten Amplitudenspektrums. Vergleicht man nun Abbildung 2.3 und Abbildung 2.4 so erkennt man, dass die Amplitudenspektren vom Profil einander entsprechen. Dies zeigt die korrekte Berechnung der Windowing Methode.

# 3

# Versuch 2 - Spracherkennung

Dieser Versuch zeigt die Funktionsweise eines einfachen Spracherkenners. Dazu wird ein Spracherkenner so aufgebaut, dass er die Worte ”Hoch”, ”Tief”, ”Links” und ”Rechts” erkennt. Mit diesen Befehlen könnte beispielsweise ein Gabelstapler in einem Hochregallager sprachgesteuert werden. Der Spracherkenner basiert auf dem Prinzip des Prototyp-Klassifikators, wie es in der Vorlesung 10 von Herrn Franz [1] beschrieben ist.

## 3.1 Fragestellung, Messprinzip, Aufbau, Messmittel

Wie ist ein Spracherkenner aufgebaut und wie funktioniert er? Antworten darauf liefern die folgenden Kapitel, insbesondere Kapitel 3.3.

Für den Spracherkenner wird ein Referenzspektrum bzw. einen Prototyp für jedes Wort, welches er später erkennen soll, benötigt. Zudem ist es sinnvoll einen Testdatensatz aufzunehmen. Dazu wird der selbe Versuchsaufbau wie in Versuch 1 verwendet. Mit dem Mikrofon wird diesmal ein Referenzdatensatz und ein Testdatensatz von zwei Sprechern A und B aufgenommen. Der Referenzdatensatz besteht aus den Worten ”Hoch”, ”Tief”, ”Links” und ”Rechts”, jeweils fünf mal, von Sprecher A gesprochen. Dieser dient dem Spracherkenner später als Vergleichswert. Der Testdatensatz besteht ebenfalls aus den oben genannten Worten, jeweils fünf mal, einmal von Sprecher A und einmal von Sprecher B gesprochen.

## 3.2 Messwerte

Alle in Kapitel 3.1 gemachten Aufnahmen (Referenzsignale und Testsignale) sind getriggert und auf die Länge von einer Sekunde zu geschnitten. Für die Testsignale ist an dieser Stelle keine weitere Bearbeitung notwendig. Das Referenzspektrum für später soll jedoch schon hier berechnet werden. Dazu wird auf jedes Referenzsignal die Windowing-Methode aus Versuch 1 angewandt. Das Referenzspektrum für jedes Wort erhalten wir durch zusätzliche Mittelung der jeweils fünf Spektren. In den Abbildungen 3.1, 3.2, 3.3 und 3.4 ist jeweils das Referenzspektrum der Worte ”Hoch”, ”Tief”, ”Links” und ”Rechts” abgebildet.

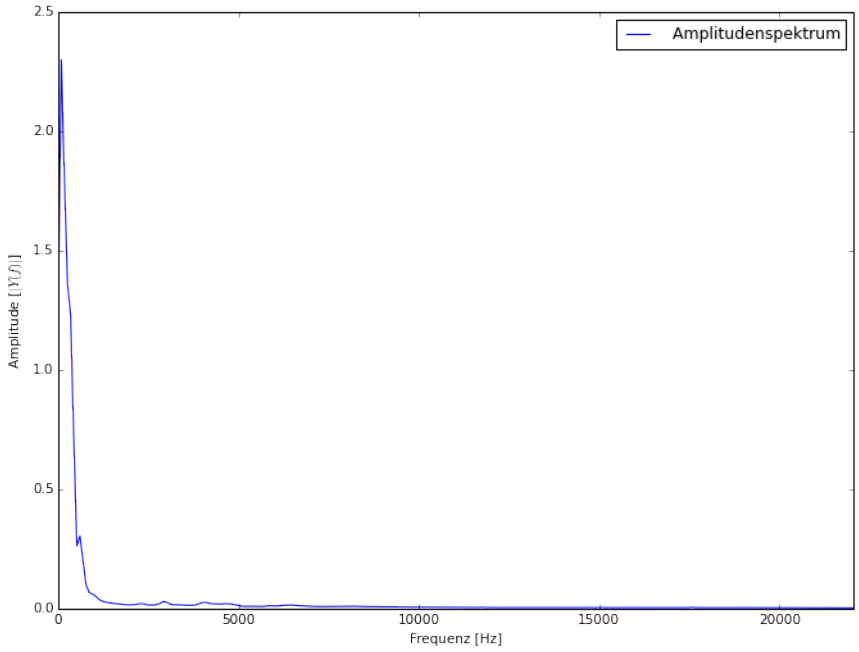


Abbildung 3.1: Referenzspektrum des Wortes "Hoch"

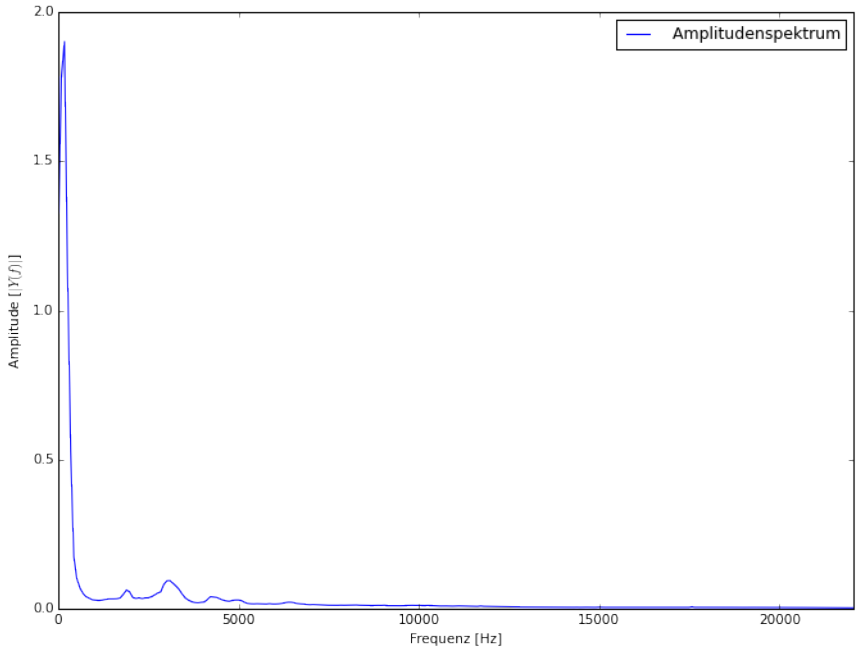


Abbildung 3.2: Referenzspektrum des Wortes "Tief"

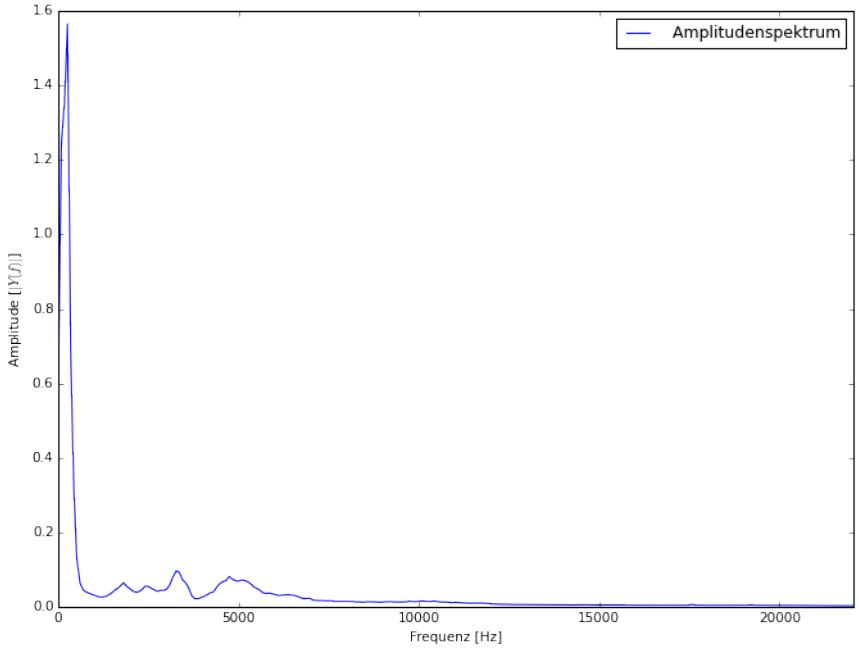


Abbildung 3.3: Referenzspektrum des Wortes "Links"

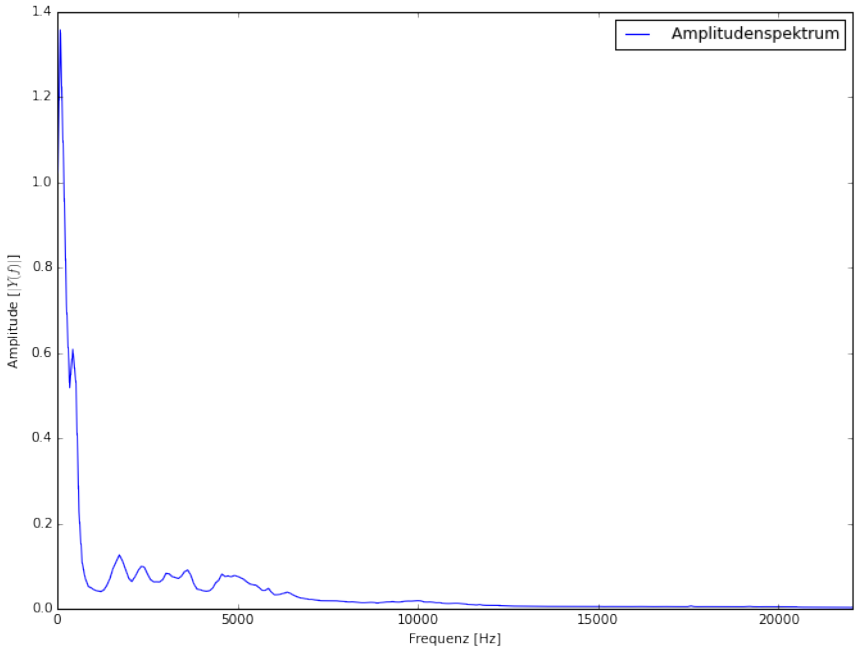


Abbildung 3.4: Referenzspektrum des Wortes "Rechts"

## 3.3 Auswertung

Mit den Messwerten aus Kapitel 3.2 kann nun ein Spracherkenner aufgebaut und getestet werden. Unser Spracherkenner funktioniert folgendermaßen:

Ein beliebiges Signal (also ein Wort) aus dem Testdatensatz wird dem Spracherkenner übergeben. Dieser berechnet zunächst das Spektrum des Signals mit der Windowing-Methode. Danach vergleicht bzw. korreliert er jedes Referenzspektrum mit dem Eingabespektrum. Dazu wird jeweils der Korrelationskoeffizient nach Bravais-Pearson berechnet. Ein Koeffizient nahe an 1 bedeutet eine hohe Ähnlichkeit der Signale, nahe an 0 das Gegenteil. Der Spracherkenner entscheidet sich nun für das Wort, welches die höchste Übereinstimmung mit einem der Referenzspektren hat, d.h. das Wort bei dem der Korrelationskoeffizient insgesamt betrachtet am größten ist. In Python wurde dafür eine Methode speechRecognition() erstellt, siehe Listing A.1.

Um den oben gezeigten Spracherkenner zu testen, werden ihm alle Testspektren übergeben. Das bedeutet jeweils fünf mal die Worte ”Hoch”, ”Tief”, ”Links” und ”Rechts” einmal von Sprecher A und einmal von Sprecher B gesprochen. Die Tabelle 3.1 zeigt nun die Detektionsrate unseres Spracherkenners für das jeweilige Wort pro Sprecher. Insgesamt ergibt sich eine Detektionsrate von 1.0 für Sprecher A und 0.5 für Sprecher B.

| **Wort** | **Sprecher A** | **Sprecher B** |
|---|---|---|
| Hoch | 1.0 | 0.6 |
| Tief | 1.0 | 0.4 |
| Links | 1.0 | 0.0 |
| Rechts | 1.0 | 1.0 |

Tabelle 3.1: Detektionsraten des Spracherkenners für das jeweilige Wort

## 3.4 Interpretation

Wie die Tabelle 3.1 zeigt, funktioniert der Spracherkenner hervorragend für Sprecher A. Alle Wörter wurden richtig erkannt. Von Sprecher A stammen jedoch auch die Referenzspektren, wodurch dieses Ergebnis nicht besonders überrascht. Bei Sprecher B schneidet der Spracherkenner eher schlecht ab. Lediglich das Wort ”Rechts” wurde immer korrekt erfasst, das Wort ”Links” wurde sogar nie erkannt. Der Spracherkenner arbeitet nur für Sprecher A zuverlässig.

# Anhang

## A.1 Quellcode für Versuche 1 - 2

```
1
2  #-*-coding:utf-8-*-
3  """
4  CreatedonMonDec714:09:282015
5
6  @author:edc10
7  """
8
9  #import pyaudio
10 import numpy as np
11 import scipy.signal as win
12 import scipy.stats as stats
13 import matplotlib.pyplot as plt
14
15 #FORMAT = pyaudio.paInt16
16 SAMPLEFREQ = 44100
17 SEC=2
18 READBYTES = int(44100*SEC)
19 TRIGGER = 800
20 WINDOWSIZE = 512
21 WINDOWTIME = WINDOWSIZE / SAMPLEFREQ
22
23
24 def plotRecord(rec, filename=''):
25     myDpi = 75
26     fig, ax = plt.subplots(figsize=(800/myDpi, 600/myDpi), dpi=myDpi)
27     ax.autoscale(enable=True, axis='x', tight=True)
28     time = np.linspace(0, len(rec)/SAMPLEFREQ, len(rec))
29     amp = rec/((2**15)-1)
30     ax.plot(time, amp)
```

```
31      ax.set_xlabel('Time [$s$]')
32      ax.set_ylabel('Amplitude [$V$]')
33      if filename is not '':
34          fig.savefig(filename, transparent=True, dpi=myDpi)
35      return
36
37
38  def trigger(rec):
39      for i in range(len(rec)):
40          if rec[i] > TRIGGER:
41              return rec[i:]
42      return []
43
44
45  def cut(rec):
46      if len(rec) < SAMPLEFREQ:
47          size = SAMPLEFREQ - len(rec)
48          rec = np.append(rec, np.zeros(size))
49
50      return rec[:SAMPLEFREQ]
51
52
53  def getInputDataPyAudio():
54      p = pyaudio.PyAudio()
55      print('running')
56
57      stream = p.open(format=FORMAT, channels=1, rate=SAMPLEFREQ,
58                      input=True, frames_per_buffer=READBYTES)
59      data = stream.read(READBYTES)
60      decoded = np.fromstring(data, 'Int16')
61
62      stream.stop_stream()
63      stream.close()
64      p.terminate()
65      print('done')
66      return decoded
67
68
69  def plotFFT(rec, filename=''):
70
71      #fft
72      # n = Anzahl der Schwingungen innerhalb der gesamten Signaldauer
```

```
73      amp = rec/((2**15)-1)
74      c = np.fft.fft(amp)
75      n = np.abs(c)
76
77      # spiegelung eleminieren
78      half = len(n)/2+1
79      #print(half)
80      count = np.arange(0, int(half))
81      #print(count)
82
83      # Anzahl der Schwingungen innerhalb der gesamten Signaldauer dargestellt
84      dpi=75
85      fig, axN = plt.subplots(figsize=(800/dpi,600/dpi), dpi=dpi)
86      axN.plot(count[:],n[:half], color = "blue", label=" Amplitudenspektrum ")
87      # lässt X-Achse bei 0 beginnen
88      axN.autoscale(enable=True, axis='x', tight=True)
89      axN.legend(loc='upper right');
90      axN.set_xlabel("Frequenz [Hz]")
91      axN.set_ylabel("Amplitude [$|Y(f)|$]")
92
93      # als png abspeichern
94      if filename is not '':
95          fig.savefig(filename, transparent=True, dpi=dpi)
96      return
97
98
99  def plotFFT2(rec, filename=''):
100     # spiegelung eleminieren
101     half = len(rec)/2+1
102     count = np.arange(0, int(half)) / WINDOWTIME
103
104     # Anzahl der Schwingungen innerhalb der gesamten Signaldauer dargestellt
105     dpi=75
106     fig, axN = plt.subplots(figsize=(800/dpi,600/dpi), dpi=dpi)
107     axN.plot(count[:],rec[:half], color = "blue", label=" Amplitudenspektrum ")
108     # lässt X-Achse bei 0 beginnen
109     axN.autoscale(enable=True, axis='x', tight=True)
110     axN.legend(loc='upper right');
111     axN.set_xlabel("Frequenz [Hz]")
112     axN.set_ylabel("Amplitude [$|Y(f)|$]")
113
114     # als png abspeichern
```

```
115     if filename is not '':
116         fig.savefig(filename, transparent=True, dpi=dpi)
117     return
118 
119 
120 def record5():
121     for i in range(5):
122         data = getInputDataPyAudio()
123         triggered = trigger(data)
124         cutted = cut(triggered)
125         plotRecord(cutted)
126         np.savetxt("TestMarcelRechts" + str(i) + ".csv", cutted, delimiter=",")
127         print("fertig")
128         input()
129     return
130 
131 
132 def getInputData(filename):
133     return np.genfromtxt(filename, delimiter=',')
134 
135 
136 def windowing(rec):
137     # Inrervalle für die Windows berechnen
138     steps = []
139     for i in range(0, len(rec), int(WINDOWSIZE / 2)):
140         steps.append(i)
141     steps.append(len(rec))
142 
143     # Windows erstellen
144     windows = []
145     for i in range(len(steps) - 3):
146         windows.append(rec[steps[i]:steps[i + 2]])
147 
148     #------------------------------------------------
149 
150     # Gaus anwenden
151     for i in range(len(windows)):
152         windows[i] = windows[i] * win.gaussian(WINDOWSIZE, std=WINDOWSIZE/4)
153 
154     # lokale Fouriertransformation
155     result = np.zeros(WINDOWSIZE)
156     for i in range(len(windows)):
```

```
157         windows[i] = np.fft.fft(windows[i]/((2**15)-1))
158         windows[i] = np.abs(windows[i])
159         result = result + windows[i]
160 
161     result = result / len(windows)
162     return result
163 
164 
165 def getMeans(pathname):
166     files = ['Hoch', 'Tief', 'Links', 'Rechts']
167     dic = {}
168     for name in files:
169         result = []
170         for i in range(5):
171             data = getInputData(pathname + name + str(i) + ".csv")
172             result.append(windowing(data))
173 
174         summ = np.zeros(WINDOWSIZE)
175         for res in result:
176             summ = summ + res
177 
178         mean = summ / len(result)
179         dic[name] = mean
180     return dic
181 
182 
183 def speechRecognition(referenceDict, data):
184     data = windowing(data)
185     maximum = 0
186     for key in referenceDict:
187         coefficient = stats.pearsonr(referenceDict[key], data)[0]
188         if coefficient > maximum:
189             maximum = coefficient
190             bestKey = key
191     return bestKey
192 
193 
194 def versuch1a():
195     decoded = getInputDataPyAudio()
196     #np.savetxt("Aufnahme1.csv", decoded, delimiter=",")
197     plotRecord(decoded)
198     #plotRecord(decoded, 'v1.png')
```

```
199
200
201 def versuch1b():
202     decoded = getInputDataPyAudio()
203     triggered = trigger(decoded)
204     cutted = cut(triggered)
205     plotRecord(cutted)
206
207
208 def versuch1bc():
209     data = np.genfromtxt('Aufnahme1.csv', delimiter=',')
210     triggered = trigger(data)
211     cutted = cut(triggered)
212     plotRecord(cutted, 'Aufnahme1TriggerCut.png')
213     np.savetxt("Aufnahme1TriggerCut.csv", cutted, delimiter=",")
214     plotFFT(cutted, 'Aufnahme1TriggerCutFFT.png')
215
216
217 def versuch1d():
218     data = getInputData('Aufnahme1TriggerCut.csv')
219     data = windowing(data)
220     plotFFT2(data, "Aufnahme1TriggerCutWindow")
221
222
223 def versuch2():
224     # Referenzspektrum einlesen (Prototypen)
225     reference = getMeans("Referenz/Referenz")
226
227     # Testdaten in Dictionaries abspeichern
228     testJulian = {}
229     testMarcel = {}
230     for name in ['Hoch', 'Tief', 'Links', 'Rechts']:
231         testJulian[name] = []
232         testMarcel[name] = []
233         for i in range(5):
234             dataJulian = getInputData("TestJulian/TestJulian" + name + str(i) + ".csv")
235             dataMarcel = getInputData("TestMarcel/TestMarcel" + name + str(i) + ".csv")
236             testJulian[name].append(dataJulian)
237             testMarcel[name].append(dataMarcel)
238
239     #Referenzspektren plotten
240     for key in reference:
```

```
241         plotFFT2(reference[key], 'Referenzspektrum' + key + ".png")
242 
243     # Spracherkennung
244     print("---Julian---")
245     for key in testJulian:
246         detections = 0
247         for i in range(5):
248             word = speechRecognition(reference, testJulian[key][i])
249             print("Wort:" + key)
250             print("Erkannt: " + word)
251             if word == key:
252                 detections += 1
253         detectionRate = detections / 5
254         print("Detektionsrate " + key + ": " + str(detectionRate))
255 
256     print()
257 
258     print("---Marcel---")
259     for key in testMarcel:
260         detections = 0
261         for i in range(5):
262             word = speechRecognition(reference, testMarcel[key][i])
263             print("Wort:" + key)
264             print("Erkannt: " + word)
265             if word == key:
266                 detections += 1
267         detectionRate = detections / 5
268         print("Detektionsrate " + key + ": " + str(detectionRate))
269 
270 
271 def main():
272     #versuch1a()
273     #versuch1bc()
274     #record5()
275     #versuch1d()
276     versuch2()
277 
278 
279 if __name__ == "__main__":
280     main()
```

Listing 4.1: QuellCodeV1 bis V2

# Literaturverzeichnis

[1] Prof. Dr. Matthias O. Franz. Vorlesung 10 - sprache und spracherkennung: Kurzzeitfouriertransformation, erzeugung und wahrnehmung von sprache, mustererkennung durch korrelation. In *Vorlesung Technische Grundlagen der angewandten Informatik*, 2015.