Ostfalia
Hochschule für angewandte Wissenschaften

Fakultät Elektrotechnik

Bachelorarbeit

Entwicklung einer Spracherkennung für den Forschungshubschrauber ACT/FHS

von

Rilana Rohde

70455436

Wirtschaftsingenieurwesen Elektro- und Informationstechnik (WEIT)

für das

Deutsche Zentrum für Luft- und Raumfahrt e.V. Braunschweig

Dipl.-Inform. Matthias Bodenstein

betreut durch

Ostfalia Hochschule für angewandte Wissenschaften

Prof. Dr. rer. nat. Claus Wilhelm Turtur

Bearbeitungszeitraum: 06. Oktober 2020 – 27. November 2020

DLR Deutsches Zentrum für Luft- und Raumfahrt

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Braunschweig, den 27. November 2020 , Rilana Rohde

(Unterschrift Rilana Rohde)

Institut für Flugsystemtechnik

Deutsches Zentrum für Luft- und Raumfahrt

DLR e. V. Institut für Flugsystemtechnik
Lilienthalplatz 7, 38108 Braunschweig

Rilana Rohde

Fischerstraße 5

38300 Wolfenbüttel

Ihr Zeichen
Ihr Schreiben
Unser Zeichen

Ihr Gesprächspartner Matthias Bodenstein

Telefon 0531 2953251
Telefax 0531 2952877
E-Mail matthias.bodenstein@dlr.de

24. September 2020

Aufgabenstellung Bachelorarbeit

Thema:

Entwicklung einer Spracherkennung für den Forschungshubschrauber ACT / FHS

Aufgabenstellung:

Hubschrauber werden aufgrund ihrer umfassenden Einsatzmöglichkeiten auch an schwer zugänglichen Orten häufig als Rettungsmittel eingesetzt. Dabei stellen die Flüge besonders bei erschwerten Außenbedingungen wie Nebel oder Dunkelheit eine große Herausforderung für die Piloten dar. Im Rahmen des DLR-Projektes „SALVARE“ sollen mithilfe von unterschiedlichen Assistenzfunktionen Piloten bei Rettungseinsätzen unter erschwerten Bedingungen unterstützt werden. Dabei ist es unter anderem vorgesehen, im Bereich des multimedialen Cockpits eine Sprachsteuerung zu entwickeln. Vorhandene offene Spracherkennungssysteme sind bisher kaum in der Lage, eine gute Spracherkennung in speziellen Einsatzbereichen wie dem eines Hubschraubers zu leisten. Ziel dieser Arbeit ist die Analyse der Einsatzmöglichkeiten und Optimierung einer Sprachsteuerung in der Umgebung eines Forschungshubschraubers.

Im Einzelnen sollen im Rahmen der Abschlussarbeit folgende Punkte bearbeitet werden:

- Literaturrecherche zu den relevanten Themengebieten
- Vergleich auf dem Markt verfügbarer Spracherkennungssysteme insbesondere im Hinblick auf Eignung für das Projekt
- Optimierung der Erkennung eines beschränkten Sprachbefehlssatzes für den ACT/FHS
- Untersuchung des Einflusses von Störgeräuschen wie sie in Hubschraubern auftreten auf die Qualität der Spracherkennung.
- Bewertung der Ergebnisse für den Einsatz im ACT/FHS
- Dokumentation der Arbeiten in einer Abschlussarbeit
- Präsentation der Arbeit in einem kurzen Vortrag im DLR

Lilienthalplatz 7
38108 Braunschweig
Telefon 0531 295-0
Internet DLR.de/ft

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACT ..... *Active Control Technology*

AI ..... *artificial intelligence*

API ..... *application programming interface*

ASR ..... automatic speech recognition

AVES ..... *Air Vehicle Simulator*

CNN ..... Convolutional Neural Networks

DL ..... *Deep Learning*

DLR ..... *Deutsches Zentrum für Luft- und Raumfahrt*

EC ..... *Eurocopter*

FHS ..... *Flying Helicopter Simulator / Fliegender Hubschrauber Simulator*

FT ..... *Institut für Flugsystemtechnik*

IT ..... *Informationstechnik*

KI ..... *künstliche Intelligenz*

KNN ..... *Künstliche Neuronale Netzwerke*

LM ..... *language model*

MFCC ..... *Mel Frequency Cepstral Coefficients*

ML ..... *Machine Learning*

MTTA ..... *mobile Telemetrie- und Telekommandoanlage*

NLP ..... *natural language processing*

RNN ..... *Recurrent Neuronal Network*

SALVARE ..... *Safe Landing and Takeoff in low Visibility for advanced Rescue Operations*

SNR ..... *Signal-Noise-Ratio*

STT ..... *Speech-to-text*

WER ..... *word error rate*

1 Einleitung und Aufgabenstellung

Der vorliegende Bericht ist im Rahmen einer Bachelorarbeit am Deutschen Zentrum für Luft- und Raumfahrt (DLR) entstanden. Betreut wurde diese von Prof. Dr. rer. nat. Claus Wilhelm Turtur von der Ostfalia Hochschule für angewandte Wissenschaften. Von Seiten des DLR, welches im Kapitel 2 vorgestellt wird, wurde das Projekt von Dipl.-Inform. Matthias Bodenstein begleitet.

Hubschrauber werden aufgrund Ihrer Vorteile in der Notfallrettung immer beliebter. Um besonders in schwierigen Situationen die Piloten zu unterstützen, wird an zahlreichen Assistenzsystemen geforscht und entwickelt. Im Rahmen des Leitkonzepts „Rettungshubschrauber 2030“, an dem auch das DLR mitarbeitet, soll unter anderem eine Sprachsteuerung für Helikopter entwickelt werden. Dies ist jedoch aufgrund der akustischen Umgebung im Hubschrauber mit Herausforderungen verknüpft. In dieser Arbeit soll die Erkennung menschlicher Sprache im Forschungshubschrauber ACT/FHS des DLR untersucht und verbessert werden. [DLR20a]

Dazu werden nach einer Vorstellung des DLR zunächst die wissenschaftlichen Grundlagen und der Stand der Technik beschrieben (Kapitel 3 und 4). Im Kapitel 5 werden die durchgeführten Messungen, Anpassungen und deren Auswertung beschrieben. Schließlich folgt im Kapitel 6 eine kritische Betrachtung und ein Ausblick auf ein mögliches weiteres Vorgehen. Das Kapitel 7 fasst die Arbeit zusammen.

2 Das Deutsche Zentrum für Luft- und Raumfahrt (DLR)

Das DLR ist das Forschungszentrum für die Luft- und Raumfahrt der Bundesrepublik Deutschland. Die über 9000 Mitarbeiter[1] sind an 27 Standorten in Deutschland verteilt. In 51 Instituten und Einrichtungen beschäftigt sich das DLR neben der Luft- und Raumfahrt mit Forschung und Entwicklung in den Themenbereichen Energie, Verkehr, Sicherheit und Digitalisierung. Diese Arbeit wird im Institut für Flugsystemtechnik geschrieben, welches im nächsten Abschnitt kurz vorgestellt werden soll. [Deu20a]; [Deu20b]

2.1 Institut für Flugsystemtechnik

Am Standort Braunschweig befindet sich unter anderem das Institut für Flugsystemtechnik (FT). Hier beschäftigen sich ca. 170 Mitarbeiter mit der Flugmechanik oder untersuchen und erforschen fliegende Systeme aller Art mess- und systemtechnisch. [Nag20]

In der Abteilung Flugversuchstechnik und Informationstechnik (IT) wird besonders an Systemen für den Einsatz in der Flugerprobung geforscht. Dazu gehört unter anderem die Versuchsanlage des Forschungshubschraubers. Diese wird im folgenden Abschnitt 2.2 erklärt. [Nag20]

2.2 Der Forschungshubschrauber ACT/FHS

Der Forschungshubschrauber ACT/FHS, zu sehen in der Abbildung 2.1, basiert auf einem Serienhubschrauber vom Typ Eurocopter EC 135 und wurde für den Einsatz als Forschungsobjekt erheblich modifiziert [Deu20d]. Dieser Leichthubschraubertyp wird oft auch im Bereich der Luftrettung, von der Polizei und dem Militär eingesetzt [rth20].

[1] Aus Gründen der besseren Lesbarkeit wird innerhalb der vorliegenden Ausarbeitung auf geschlechterspezifische Formulierungen verzichtet.

Abbildung 2.1: Der Forschungshubschrauber ACT/FHS [DLR20b]

Der Name ACT/FHS steht für die Abkürzungen Active Control Technology (ACT) / Flying Helicopter Simulator (FHS), also einen fliegenden Hubschraubersimulator mit aktiver Regelungstechnik. Diese Art der Flugregelung basiert auf der Verarbeitung von aktuellen Flugmesswerten und verbessert die Steuermöglichkeit des Helikopters. [Bit05]

Es gibt im Forschungshubschrauber zwei Arten von Steuerleitungen, eine elektrisch kabelgebundene Steuerung (fly by wire) sowie eine optische Datenübertragung mit Lichtwellenleitern (fly by light) [Deu20c]

Der Forschungshubschrauber ACT/FHS ist eine Versuchsumgebung, die für Übungen, Forschungen und Experimente im Flugbetrieb genutzt werden kann. Dazu beinhaltet er ein verbautes Experimentalsystem sowie eine Bodenstation, eine mobile Telemetrie- und Telekommandoanlage (MTTA). Außerdem gibt es ein Experimentalsystem in einem Simulator (engl. Air Vehicle Simulator, AVES), das funktional kompatibel zu dem Forschungshubschrauber ist. Die Flugversuche werden immer von einer Bodenstation aus überwacht. Der Forschungshubschrauber hat vielfältige Einsatzmöglichkeiten, so können beispielsweise neue Systeme erprobt werden, Piloten ausgebildet werden, neue Steuerungs-

und Regelungssysteme entwickelt und die Flugeigenschaften von Hubschrauberkonfigurationen simuliert werden. [Deu20c]; [Deu20d]; [Boo20]; [Wie20]

2.3 Rettungshubschrauber 2030 und das Projekt SALVARE

Da Rettungshubschrauber einige Vorteile in der präklinischen Versorgung von Notfallpatienten haben, wird angestrebt, diese immer mehr auch unter schwierigeren Bedingungen fliegen zu lassen. Auch wird angenommen, dass die Anzahl der Rettungsflüge zukünftig weiter steigen wird. Daher arbeitet unter anderem das Leitkonzept „Rettungshubschrauber 2030“ daran, die Fluggeräte zu verbessern, neue Konzepte für Rettungseinsätze zu entwickeln und mehr Assistenzsysteme einzusetzen. [DLR20a]

Am DLR wird für das Leitkonzept Rettungshubschrauber 2030 ein Projekt bearbeitet, dass einige Teilfunktionen dafür realisieren soll. Der Projektname SALVARE ist ein Akronym für Safe Landing and Takeoff in low Visibility for advanced Rescue Operations. [DLR20a]; [Dre19]

Ziel des Projektes ist die Erweiterung des Einsatzeitraumes von Rettungshubschraubern, diese sollen rund um die Uhr (24 Stunden am Tag, 7 Tage pro Woche) fliegen. Dazu werden Assistenzsysteme zur Unterstützung der Piloten entwickelt und erprobt, beispielsweise ein multimediales Cockpit. Dazu gehört unter anderem eine Sprachsteuerung im Helikopter. Im Rahmen dieser Arbeit soll zunächst die Spracherkennung betrachtet werden. Die erkannten Sprachbefehle müssen dann verarbeitet und in eine Sprachsteuerung für den Forschungshubschrauber umgesetzt werden. [Dre19]

3 Wissenschaftliche Grundlagen

Um den Hintergrund dieses Projektes besser einordnen zu können, werden in den folgenden Abschnitten die wissenschaftlichen und technischen Grundlagen kurz beschrieben. Im Abschnitt 3.1 werden die Begriffe Schall und Rauschen eingeführt und das Vorkommen besonders im Helikopter beschrieben. Mit der menschlichen Sprache und Sprachsignalen sowie Sprachsignalverarbeitung setzt sich der Abschnitt 3.2 auseinander. Schließlich werden im Abschnitt 3.3 Neuronale Netze und deren Aufbau und Abgrenzung beschrieben.

3.1 Schall und Rauschen

Als Schall werden mechanische Schwingungen in einem elastischen Medium bezeichnet. Dieser kann beispielsweise in Frequenzbereiche unterschieden werden. Der für Menschen wahrnehmbare Frequenzbereich liegt grob zwischen etwa $16\ \mathrm{Hz}$ und $20\ \mathrm{kHz}$, Schall in dem Bereich wird auch als Hörschall bezeichnet. [MS05]; [Spe14]

Rauschen ist ein zufälliges Schallsignal, welches nicht vorhersagbar ist. Es lassen sich keine Abhängigkeiten und statistisches Verhalten darin finden, damit wird es auch als stochastisches Signal oder Zufallssignal bezeichnet. [Ham16]; [OL05]; [ZBH87]

Die unterschiedlichen Arten des Rauschens sollen im nächsten Unterabschnitt 3.1.1 erklärt werden. Der Unterabschnitt 3.1.2 geht auf das Auftreten und die Beschreibung von Schall und Rauschen in einem Helikopter ein.

3.1.1 Arten von Rauschen

In Analogie zu den Wellenlängen der Optik werden die unterschiedlichen Rauscharten mit Farben bezeichnet. Die Abbildung 3.1 zeigt schematisch eine Übersicht über die unterschiedlichen Rauscharten. Dabei ist jeweils der Schalldruckpegel über der logarithmischen Frequenz aufgetragen. [Sem16]

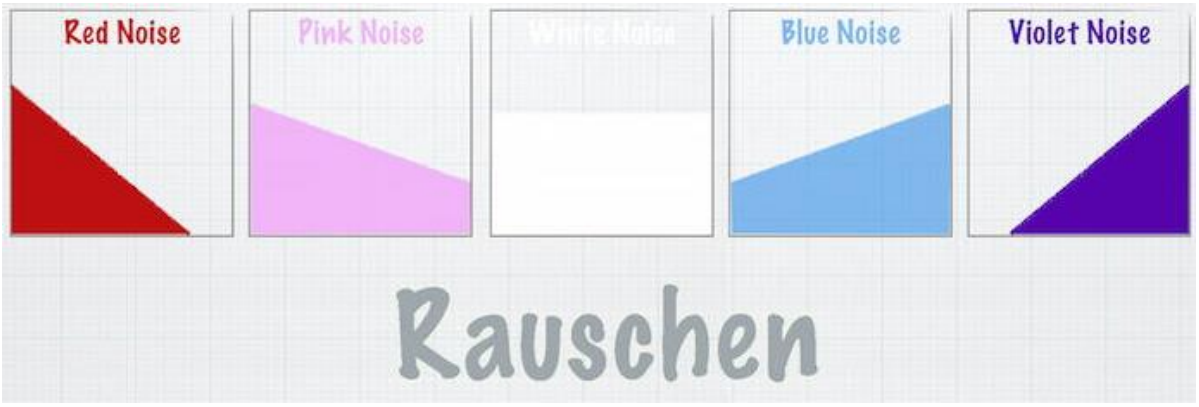

Abbildung 3.1: Schematische Darstellung der unterschiedlichen Rauscharten und derer Farbbezeichnungen [Sem16]

Das am meisten gebrauchte Rauschsignal ist das weiße Rauschen. Es hat in einem betrachteten Frequenzbereich ein konstantes Leistungsdichtespektrum. In der Akustik wird weißes Rauschen oft als Standardmesssignal genutzt, da es gut reproduzierbar ist. [SSK14]; [OL05]; [HRL81]; [Hei20]

Das rosa Rauschen (engl. pink noise) wird aufgrund der Eigenschaften des menschlichen Hörsystems subjektiv als gleich über alle Frequenzen wahrgenommen. Tatsächlich enthält es gegenüber dem weißen Rauschen eine Pegeldämpfung von 3 dB pro Oktave. Dies entspricht einer Dämpfung von 10 dB pro Dekade. [SSK14]; [Lip10]; [Ham16]; [Hei20]

Eine stärkere Dämpfung mit steigender Frequenz tritt bei rotem oder braunem Rauschen (engl. brown noise) auf. Außerdem gibt es noch blaues Rauschen sowie violettes Rauschen. Dabei nimmt der Schalldruckpegel mit steigender Frequenz zu. [Sem16]; [Lip15]

Die Schallarten und das Rauschverhalten im Helikopter werden im folgenden Unterabschnitt beschrieben.

3.1.2 Schall und Rauschverhalten im Helikopter

Schall wird unter anderem nach dem Medium, in dem er sich ausbreitet, eingeordnet. So werden die Begriffe Luftschall, Körperschall und Wasserschall unterschieden. Für die Geräusche im Innenraum eines Hubschraubers sind im Wesentlichen der Luftschall und der Körperschall wichtig. [Spe14]; [Zei06]

Der Luftschall zeigt in Untersuchungen ein nicht-statistisches und damit nicht vorhersagbares Verhalten, welches einem weißen Rauschen ähnelt. Der Körperschall hingegen wird an einem Helikopter durch verschiedene Komponenten beeinflusst. Die Abbildung 3.2 zeigt den Schalldruckpegel im Inneren eines Helikopters. Dabei werden zwei unterschiedliche Flugzustände des Helikopters, der Schwebeflug und der Bodenlauf, gemessen und jeweils einer vorherigen Simulation gegenübergestellt.

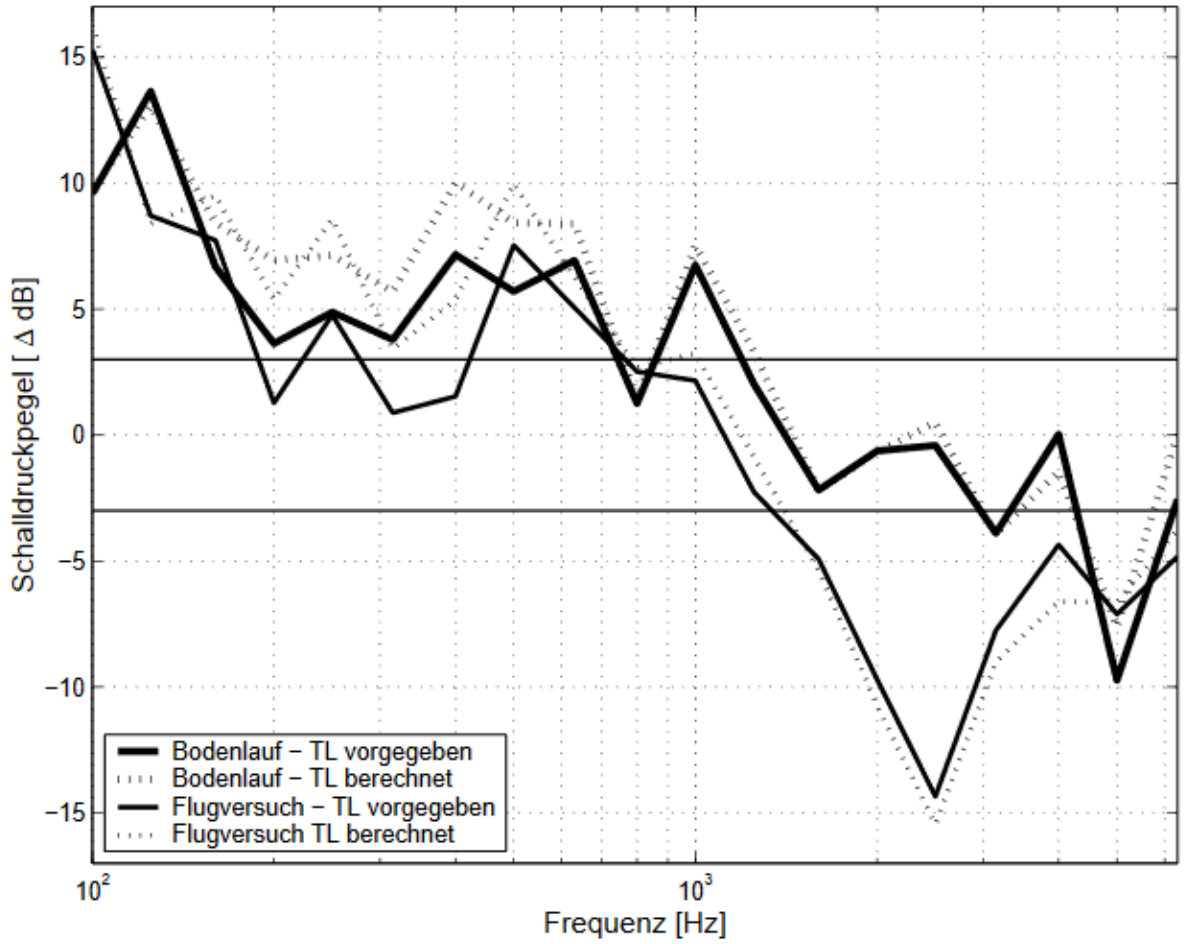

Abbildung 3.2 Innenlärmpegelvergleich bei Bodenlauf und im Schwebeflug [Zei06]

Die Schalldruckpegel zeigen keinen streng monotonen Verlauf. In dem betrachteten Frequenzbereich zwischen 0,1 kHz und ca. 6 kHz nimmt der Schalldruckpegel tendenziell mit steigender Frequenz ab. Das Zustandekommen des Verlaufes ist genauer bei Zeitler [Zei06] nachlesbar.

3.1.3 Kenngröße Signal-zu-Rausch-Abstand

Der Zusammenhang zwischen einem Signal zu dem Rauschen (engl. noise) wird auch als Signal-zu-Rausch-Verhältnis (engl. Signal-to-Noise-Ratio (SNR)) bezeichnet. Dieser ist definiert über die Leistungen:

$$\frac{S}{N} = \frac{P_{\text{Signal}}}{P_{\text{Rauschen}}}. \tag{3.1}$$

Dies lässt sich auch als Pegelmaß angeben:

$$SNR = 10 \cdot \log\left(\frac{P_{\text{Signal}}}{P_{\text{Rauschen}}}\right). \tag{3.2}$$

Der SNR kann sich auf die Verständlichkeit eines akustischen Signals auswirken. Er wird beispielsweise beeinflusst durch die Quantisierung eines Signals. [FB08]; [Hei20]

3.2 Sprache und Sprachsignale

Im folgenden Abschnitt soll auf einige Grundlagen der menschlichen Sprache und deren Verarbeitung eingegangen werden. Dazu werden zunächst die Ebenen der menschlichen Sprache beschrieben in Unterabschnitt 3.2.1. Wie die Sprache produziert wird, zeigt der Unterabschnitt 3.2.2. Sprachsignale können sehr unterschiedlich sein. Welche Variationen es bei Sprachsignalen gibt, wird in Unterabschnitt 3.2.3 beschrieben. Mit der Transformation und Verarbeitung von den Sprachsignalen befasst sich der Unterabschnitt 3.2.4.

3.2.1 Ebenen der menschlichen Sprache

Die menschliche Sprache besteht aus verschiedenen Einheiten. Die Abbildung 3.3 stellt eine Aufteilung der Struktur schematisch dar. Es gibt auch andere Aufteilungen wie beispielsweise linguistische Ebenen. [PK17]; [FS07]

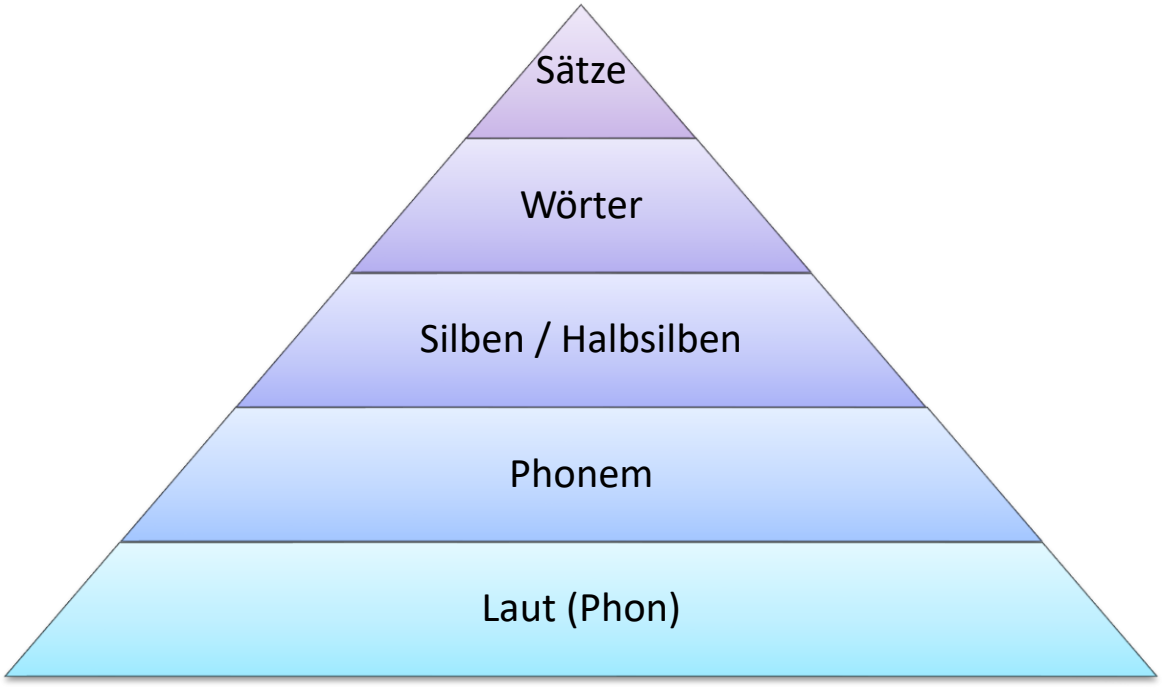

Abbildung 3.3: Aufbau der Sprache nach [FS07]

Einzelne Buchstaben der menschlichen Sprache können zu Lauten zusammengefasst werden. Diese werden auch Phone genannt und können wiederum zu Phonemen zusammengesetzt werden. Phoneme sind die kleinsten, akustisch voneinander zu unterscheidende Einheiten [Mey20]. Aus den Phonemen werden Silben und Halbsilben gebildet. Wörter bestehen dann aus einer oder mehreren Silben und können zu Sätzen formuliert werden. [FS07]; [PK17]

3.2.2 Sprachproduktion

Die Produktion der menschlichen Sprache ist ein komplexer physiologischer Prozess. Dieser kann modellhaft für die Betrachtung der Signalverarbeitung vereinfacht werden. Die Abbildung 3.4 zeigt ein Modell der Sprachproduktion. [Nie03]; [PK17]

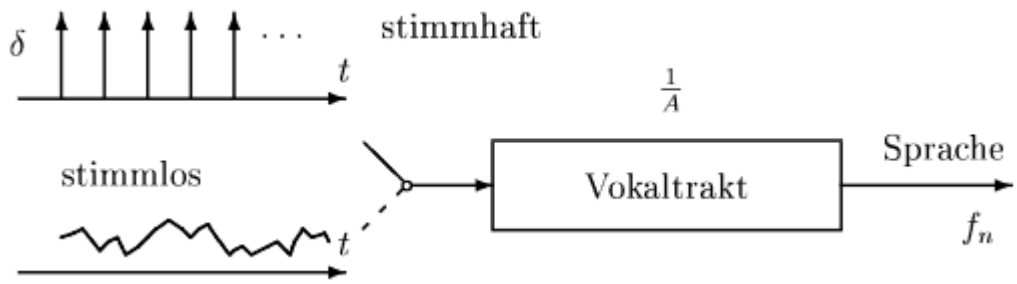

Abbildung 3.4: Modell der Sprachproduktion als lineares System [Nie03]

Man unterscheidet dabei zwischen stimmlosen und stimmhaften Lauten. Diese werden vom sog. Vokaltrakt geformt zur menschlichen Sprache. Signaltechnisch lässt sich dies in der Form des Sprachsignals über der Zeit sehen. Die stimmhaften Vokale bewirken stärkere Ausschläge des Sprachsignals als stimmlose. Vereinfacht wird der stimmhafte Anteil als Deltakamm modelliert. Ein Deltakamm ist eine periodische Folge von Dirac-Impulsen. Mit einem Dircac-Impuls wird ein idealer Puls unendlicher Höhe und unendlich schmaler Breite mit dem Flächeninhalt 1 beschrieben. [Nie03]; [PK17]; [OL05]

Weitere Informationen zur menschlichen Sprache und der Produktion dieser lässt sich in zahlreicher weiterführender Literatur finden [PK17]; [Nie03]; [FS07].

3.2.3 Variationen von Sprachsignalen

Ein Sprachsignal ist sehr individuell und hängt von einigen Einflussfaktoren ab. Die Abbildung 3.5 gibt einen Überblick, welche Einflüsse ein Sprachsignal ausmachen.

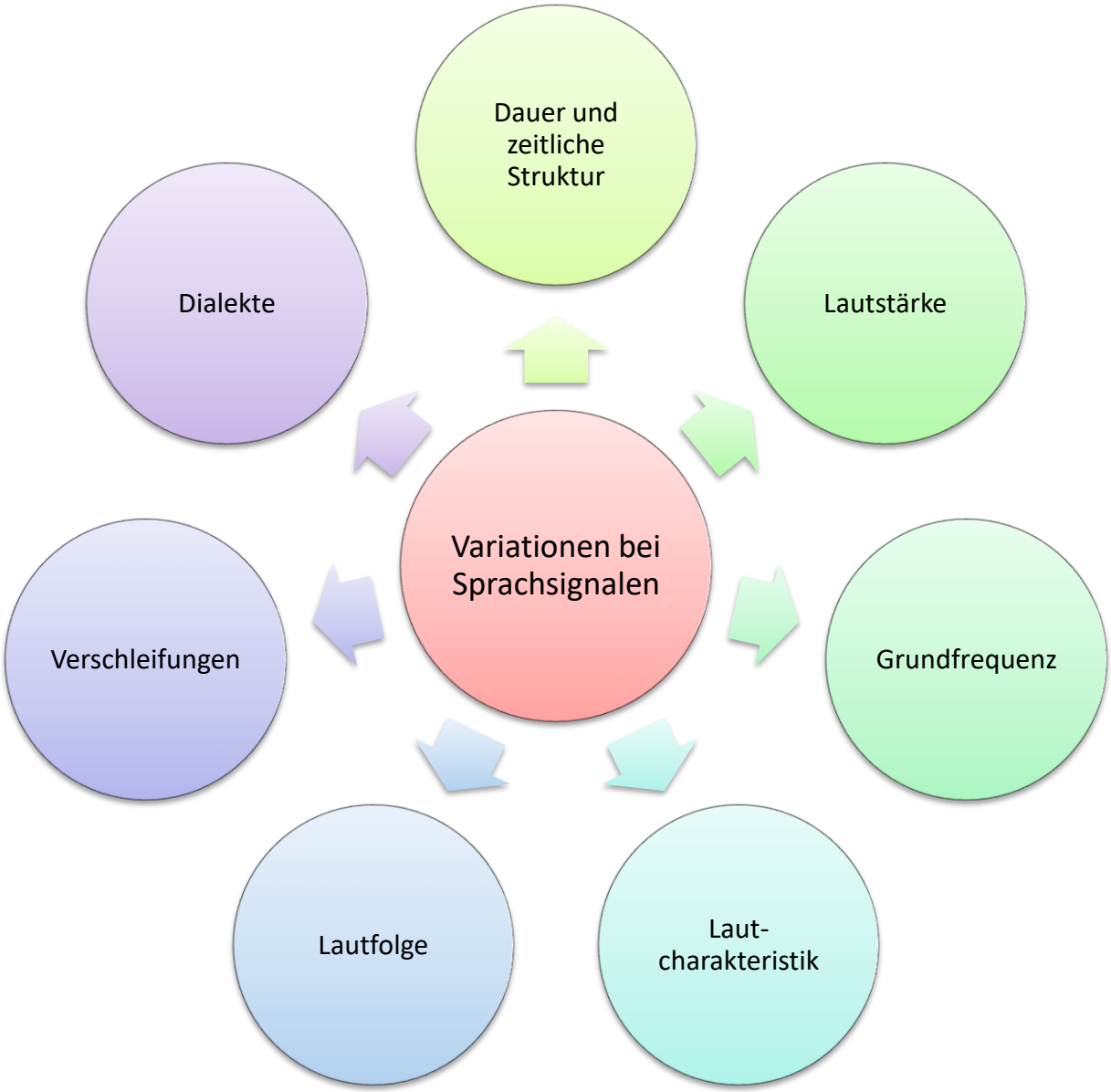

Abbildung 3.5: Variationen bei Sprachsignalen

Ein Sprachsignal kann unterschiedlich lang und verschieden strukturiert sein. Die Dauer liegt meist zwischen 50 % und 200 % der mittleren Dauer. Auch die Lautstärke kann situativ variieren. Je nach Sprecher ist die Grundfrequenz des Signals unterschiedlich, sie liegt meist zwischen 50 Hz und 400 Hz. Die Lautcharakteristik kann durch verschiedene Dinge beeinflusst werden, zum einen ist die Physiologie des Sprechers entscheidend, beispielsweise das Alter und das Geschlecht. Zum anderen hat auch der Übertragungsweg Einfluss auf die Charakteristik. Aufgrund unterschiedlicher Aussprachevarianten kommt es zu

verschiedenen Lautfolgen. Beim schnellen Sprechen kann es durch Ungenauigkeiten zu sogenannten „Verschleifungen“ kommen, die beispielsweise zwei einzelne Wörter als eines klingen lassen können. Auch Dialekte des Sprechers bewirken Unterschiede bei den Sprachsignalen. [PK17]

3.2.4 Transformationen von Sprachsignalen

Um das Verhalten eines Signals zu beschreiben und bei Übertragung durch ein System zu analysieren, können Transformationen wie beispielsweise die Laplace-Transformation genutzt werden. Diese sind in zahlreicher Literatur ausführlich beschrieben. [OL05]; [HRL81]; [LL19]; [PJ19]; [Mey17]; [Kar17]

Sprachsignale weisen eine zeitlich veränderliche Struktur auf. Da für die Verarbeitung eine Periodizität notwendig ist, werden Sprachsignale in kurze Zeitfenster aufgeteilt und diese als quasistationäre Intervalle betrachtet. Als Kompromiss zwischen einer guten zeitlichen und spektralen Auflösung sind die Zeitfenster einige $10\ \mathrm{ms}$ lang. [Nie03]; [OL05]

Für die Transformation menschlicher Sprachsignale gibt es eine eigene Vorgehensweise, das Mel Frequency Cepstrum. Dies orientiert sich an dem Modell der Sprachproduktion (vgl. Unterabschnitt 3.2.2) und der Physiologie des Menschen. Daher ist dies für die Spracherkennung besonders gut geeignet. Mel ist die Einheit der Tonheit, einer psychoakustischen Empfindungsgröße, die die menschliche Wahrnehmung / Audiologie wiedergeben soll und auf der Grundlage von physiologischen Gegebenheiten des menschlichen Ohres eingeführt wurde [SSK14]; [LSW09]. Das Cepstrum ist die Transformation eines transformierten Signals. Das Wort ist dabei ein Anagramm aus dem Wort Spectrum, der englischen Schreibweise des Wortes Spektrum. Dabei werden dann die Mel Frequency Cepstrum Coefficients (MFCC) bestimmt und verwendet. [Nie03]; [Mey20]; [PK17]

3.3 Deep Learning und künstliche Intelligenz

Das Thema der Künstlichen Intelligenz ist ein aktueller Forschungsbereich. Um die unterschiedlichen Begriffe des Themas voneinander abzugrenzen, werden diese zunächst im Unterabschnitt 3.3.1 eingeführt. In diesem Zusammenhang sind besonders die Neuronalen Netze wichtig. Der Unterabschnitt 3.3.2 geht auf den Aufbau und die Unterscheidung dieser ein. Mit unterschiedlichen Lernstrategien befasst sich der Unterabschnitt 3.3.3. Dabei soll im Folgenden nur kurz auf einige ausgewählte Bereiche des Themenkomplexes der Künstlichen Intelligenz eingegangen werden, da es nicht den Kernbereich dieser Arbeit darstellt.

3.3.1 Begriffsabgrenzungen

Um die Begriffe im Zusammenhang mit Künstlicher Intelligenz voneinander abzugrenzen und zu definieren, sollen diese im folgenden einmal beschrieben werden. Die Abbildung 3.6 bringt die Bezeichnungen Künstliche Intelligenz (KI), auf Englisch Artificial Intelligence (AI), Machine Learning (ML), Künstliche Neuronale Netzwerke (KNN) und Deep Learning (DL) in einen groben Zusammenhang.

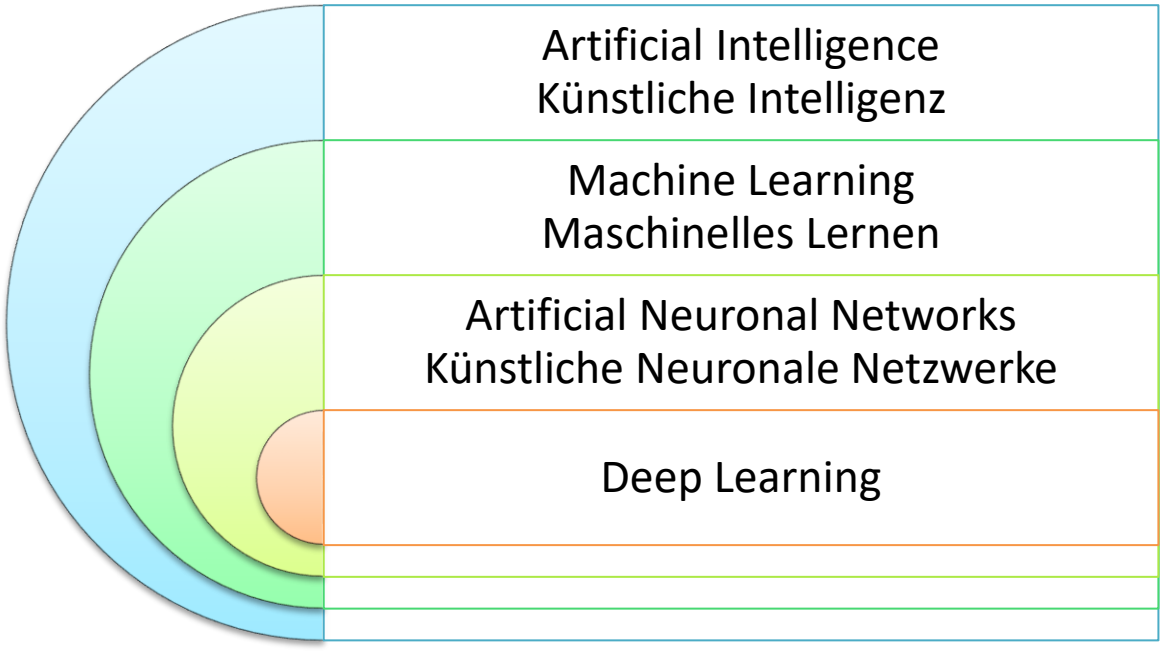

Abbildung 3.6: Begriffseinordnungen Künstliche Intelligenz, Machine Learning, Künstliche Neuronale Netze und Deep Learning nach [MM20]; [SS19]

Der Begriff der Künstliche Intelligenz ist der Oberbegriff des Themenbereiches. Für ein intelligentes System gibt es unterschiedliche Definitionen. Einige oft genannten Eigenschaften für Intelligenz sind beispielsweise das Lernen, das Schlussfolgern und das Vorausschauen. Ein intelligentes System soll die menschliche Intelligenz auf einigen Gebieten nachahmen. Beim maschinellen Lernen geht es darum, aus Datensätzen Erfahrung und daraus Wissen zu extrahieren. Dies kann beispielsweise durch Erkennung von Mustern geschehen. Mithilfe von künstlichen neuronalen Netzen und etwas spezieller und tiefgehender im Deep Learning wird die Struktur des menschlichen Gehirns als neuronales Netz nachempfunden Damit sollen Daten abstrahiert und mit numerischen Methoden das Lernen optimiert werden. Dabei kommen verschiedene Regler, auch Hyperparameter genannt, zum Einsatz, die das Ergebnis verbessern sollen. [SS19]; [MM20]; [RM20]; [Wut19]

3.3.2 Aufbau und Arten von Neuronalen Netzen

Um den Aufbau des menschlichen Gehirns nachzuahmen, werden oft Neuronale Netze genutzt. Diese orientieren sich modellhaft an menschlichen Neuronen und deren Verschaltung. Auf den generellen Aufbau von Neuronen wird in diverser Literatur näher eingegangen [MM20]; [Ras17]; [SS19]; [Wut19]. Im Unterunterabschnitt 3.3.2.1 wird zunächst grob der allgemeine Aufbau von Neuronalen Netzen thematisiert, bevor im Unterunterabschnitt 3.3.2.2 unterschiedliche Arten vorgestellt werden.

3.3.2.1 Aufbau eines Neuronalen Netzes

Die Neuronen sind im menschlichen Körper miteinander verbunden. Dies wird im Computer oft dadurch simuliert, dass mehrere Schichten Neuronen zu einem neuronalen Netz zusammengefasst werden. Die Verbindungen zwischen den Schichten werden unterschiedlich gewichtet. Diese Gewichte sind zunächst zufällig und können durch entsprechendes Training an die benötigten Aufgaben angepasst werden. Oft werden diese auch als Künstliche Neuronale Netze bezeichnet. Die [Ras17]; [RM20]

Neuronen reagieren nicht sofort auf Signale, sondern warten, bis ein gewisser Schwellwert überschritten wird. Erst dann wird das Signal weitergeleitet. Dadurch werden Rauschen und ungewollte, kleine Signale unterdrückt, damit nur starke, gewollte Signale weitergegeben werden. Dies wird auch als „Alles-oder-nichts-Gesetz“ bezeichnet. In einem neuronalen Netz wird dies mithilfe einer Aktivierungsfunktion nachgebildet. [Ras17]; [RM20]; [SS19]; [Tra19]

3.3.2.2 Arten von Neuronalen Netzen

Es gibt unterschiedliche Arten von Neuronalen Netzen. Diese unterscheiden sich je nach Aufbau und Struktur voneinander. Die Abbildung 3.7 gibt einen Überblick über eine Auswahl von Arten.

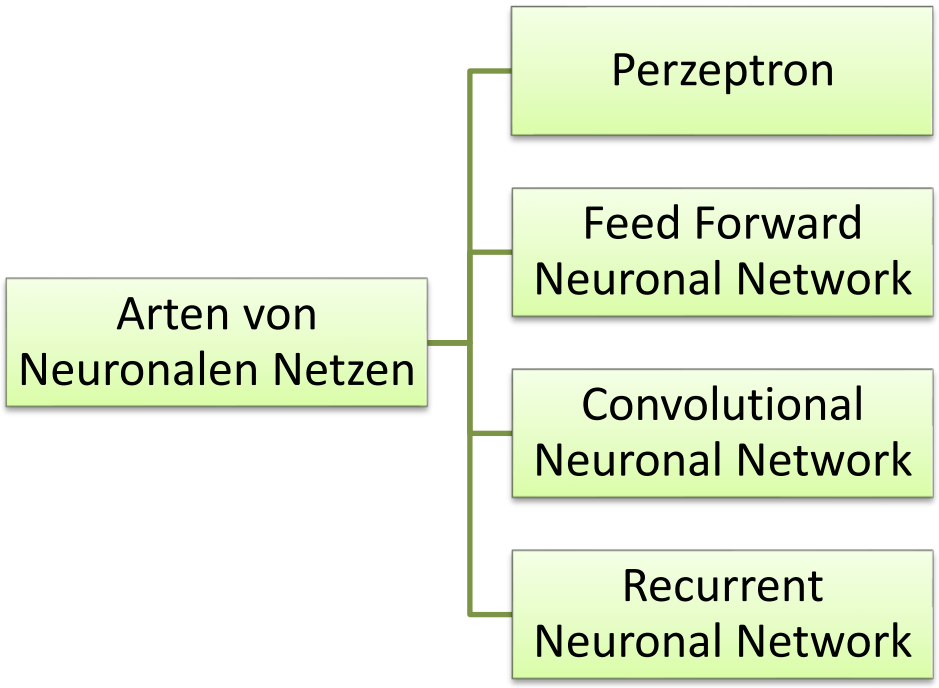

Abbildung 3.7: Arten von Neuronalen Netzen

Eine sehr einfache Struktur eines Neuronalen Netzes ist das Perzeptron. Es wird mithilfe eines Schwellwertes eine Entscheidung zwischen 0 oder 1 getroffen. Dies ist eher für simple Anwendungsfälle gedacht. Komplexer dagegen sind die vorwärtsgerichteten Neuronalen Netze, auch Feed-Forward Neuronal Network. Diese bestehen aus mehreren Schichten von Neuronen. Dabei sind die Neuronen immer mit welchen aus der direkt darüber liegenden

Schicht verbunden. Bei den rückgekoppelten Neuronalen Netzen (engl. Recurrent Neuronal Network, RNN) kommt zusätzlich noch ein „Gedächtnis“ dazu. Durch wiederkehrende Zellen wird der vorherige Zustand mit berücksichtigt und erlaubt eine Entscheidung in Abhängigkeit der Vorgeschichte. Convolutional Neural Networks (CNN) arbeiten wiederum ohne Rückkopplung. Mithilfe von mathematischen Operationen werden diese besonders für zweidimensionale und dreidimensionale Daten trainiert. [SS19]; [Wut19]

3.3.3 Lernstrategien

Es gibt verschiedene Möglichkeiten, wie ein neuronales Netz Daten verarbeiten und daraus Informationen extrahieren kann. Diese werden auch in unterschiedlichen Lernstrategien zusammengefasst. Die Abbildung 3.8 zeigt eine Auswahl einiger Lernstrategien. [MM20]; [SS19]

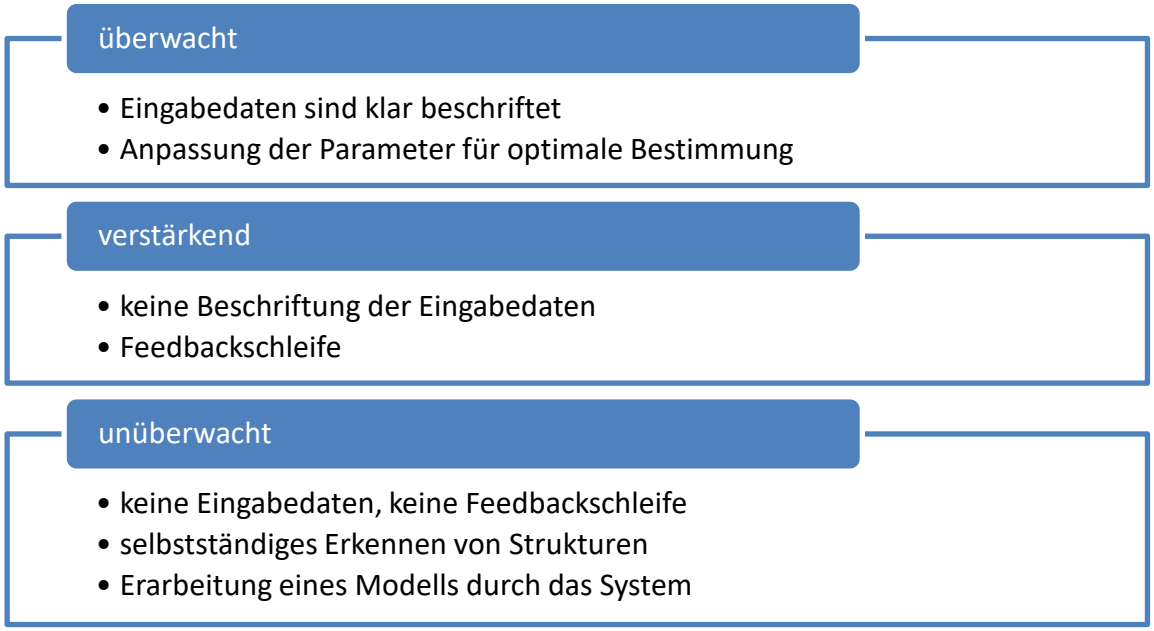

Abbildung 3.8: Überblick über drei verschiedene Lernstrategien nach [MM20]

Beim überwachten Lernen bekommt das System Eingabedaten, die mit Informationen beschriftet sind. Beispielsweise sind bei der Spracherkennung die gesprochenen, zu erkennenden Wörter mit dem richtigen Text versehen, dies wird auch als beschriftete Eingabedaten bezeichnet. Es wird also anhand von Beispielen gelernt. Dabei wird die Ausgabe mit dem richtigen Ergebnis verglichen und so das Neuronale Netz trainiert, um die

Ausgabe richtig anzupassen. Beim verstärkenden Lernen werden die Eingabedaten nicht beschriftet, das System muss selbst Korrelationen finden. Zu den Entscheidungen bekommt das System ein Feedback. Richtige Entscheidungen werden belohnt, das Ziel ist eine Maximierung der Belohnung. Anders funktioniert das unüberwachte Lernen. Es werden keine beschrifteten Eingabedaten vorgegeben, auch ein Feedback gibt es nicht. Aus dem Input, den das System bekommt, muss es selbstständig Strukturen erkennen und soll daraus ein Modell entwickeln. Dies wird beispielsweise zur Segmentierung von Mustern und Daten eingesetzt. [SS19]; [MM20]

4 Stand der Technik

In diesem Kapitel wird der aktuelle Stand der Technik von den für diese Arbeit relevanten Themengebieten dargestellt. Zunächst wird das Thema der Sprachverarbeitung im Abschnitt 4.1 beschrieben. Im Abschnitt 4.2 folgt eine kurze Vorstellung des Spracherkennungsprogramms Mozilla DeepSpech.

4.1 Sprachverarbeitung

Die Sprachverarbeitung ist ein aktuelles und komplexes Thema. Meist werden dabei statistische Methoden angewandt, um Aufgaben zu lösen, wie beispielsweise Texte zu „verstehen“. Dies kommt in unterschiedlichen Anwendungsgebieten vor und es gibt verschiedene Techniken. In den folgenden Unterabschnitten soll auf einige relevante Unterscheidungen und wichtige Grundlagen kurz eingegangen werden. Zunächst werden die unterschiedlichen Arten der Sprachverarbeitung beschrieben im Unterabschnitt 4.1.1. Das Saarbrückener Pipelinemodell (4.1.2) ist eine Darstellung der Vorgehensweise bei der Sprachverarbeitung. Der Unterabschnitt 4.1.3 zeigt eine Einteilung von Spracherkennungssystemen. Für die Spracherkennung kann Vorwissen sehr nützlich sein. Dies wird mit der Sprachmodellierung im Unterabschnitt 4.1.4 erklärt. Wie die Erkennungsleistung eines Spracherkenners beurteilt werden kann, wird im Unterabschnitt 4.1.5 beschrieben.

4.1.1 Arten der Sprachverarbeitung

Für das deutsche Wort „Sprache“ gibt es im Englischen zwei verschiedene Übersetzungen, speech und language. So werden auch die unterschiedlichen Arten der Sprachverarbeitung bezeichnet. Daher kann in der Sprachverarbeitung unterschieden werden in „speech processing“ und „(natural) language processing“, wie die Abbildung 4.1 zeigt.

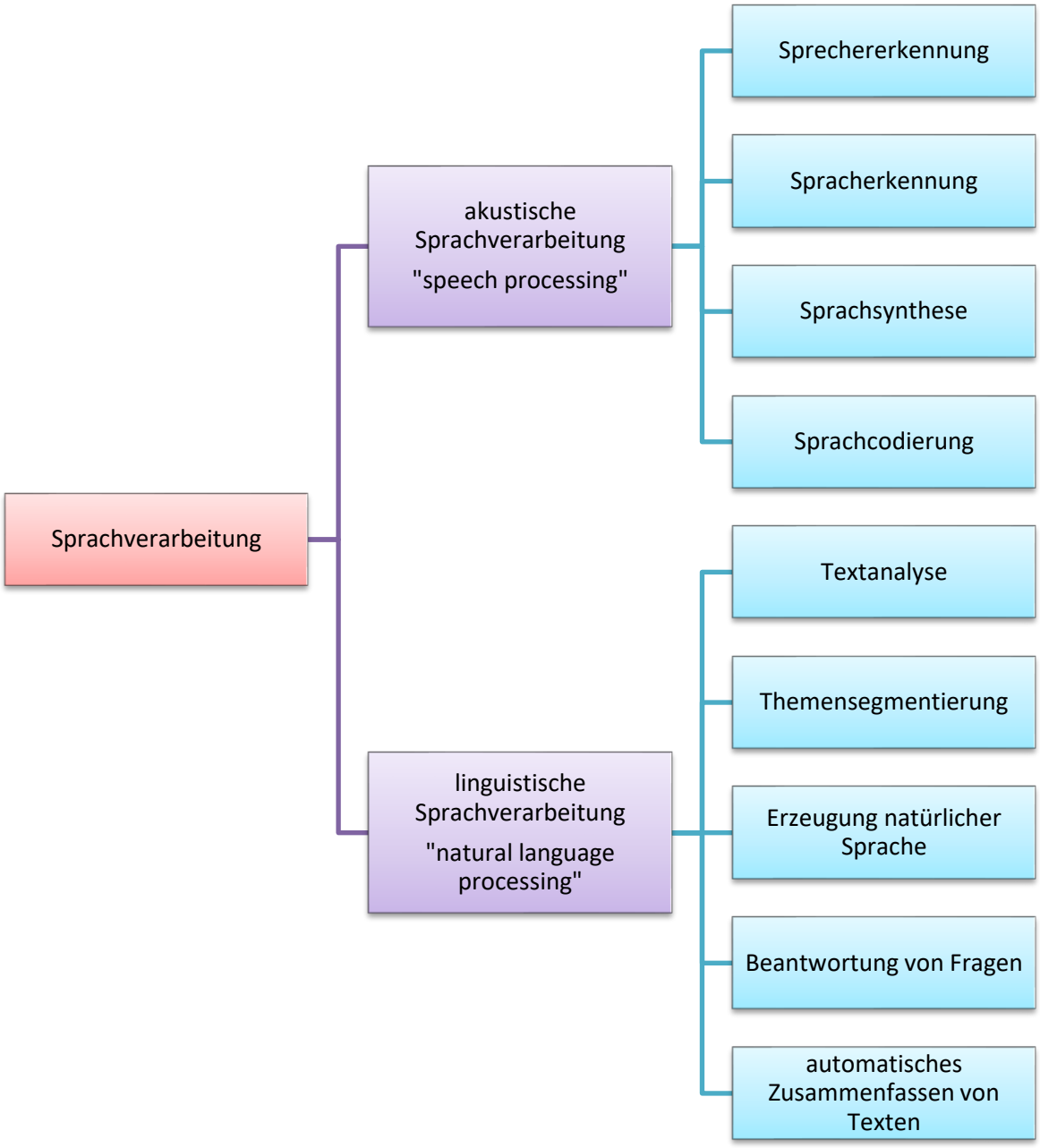

Abbildung 4.1: Unterscheidungen der Sprachverarbeitung mit ausgewählten Beispielen

Das englische Wort „speech“ beschreibt die Audioebene, hierbei geht es um die physikalischen Sprachsignale. Hingegen bezieht „language“ die Wörter und ihren Zusammenhang auf Textebene mit ein. Bei der akustischen Sprachverarbeitung (engl. speech processing) wird die Erkennung und Verarbeitung der Audiosignale betrachtet, dazu gehört

beispielsweise auch die Sprechererkennung. Bei der linguistischen Sprachverarbeitung (engl. natural language processing, (NLP) werden Texte analysiert, es sollen natürliche[1] Sprachen erkannt und inhaltlich verstanden werden. Dazu gehört auch die Analyse von Diskussionen und die automatische Beantwortung von Fragen. [RM20]; [Mey20]; [Sch95]

Im Rahmen dieser Arbeit wird vor allem das speech processing betrachtet. Daher wird im Folgenden der Begriff der Sprachverarbeitung dafür verwendet.

4.1.2 Saarbrückener Pipelinemodell

Es gibt unterschiedliche Modelle, die die Schritte der Sprachverarbeitung darstellen. Beispielhaft wird im Folgenden das Saarbrückener Pipelinemodell vorgestellt. Die Abbildung 4.2 gibt einen Überblick über den Ablauf bei diesem Modell.

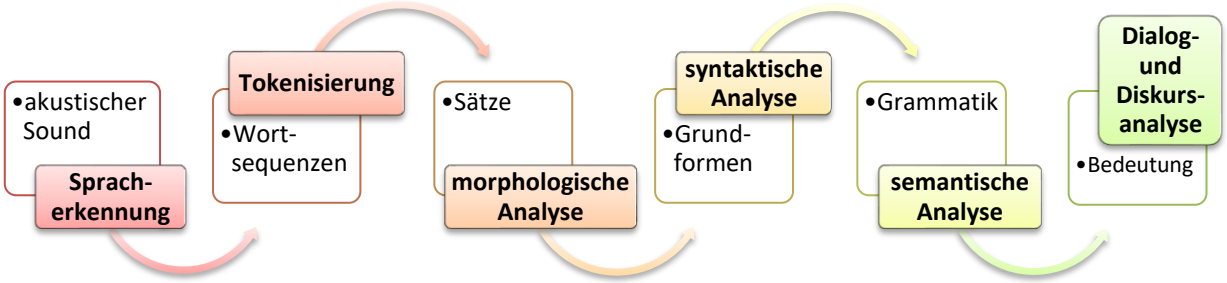

Abbildung 4.2: Saarbrückener Pipelinemodell, Darstellung nach [Mey20]

Zunächst ist eine akustische Aufnahme vorhanden, diese wird mithilfe einer Spracherkennung in Text umgesetzt. Diese Wortsequenzen werden mithilfe von Tokenisierung in Wörter unterteilt und zu Sätzen zusammengefasst. Mit der morphologischen Analyse der Sätze werden die Grundformen der Wörter ermittelt. Die Wörter werden dann auf ihre Funktion in der Satzstruktur untersucht, dies geschieht in der syntaktischen Analyse. Ist die Grammatik vollständig erkannt, so wird mithilfe der

[1] Als natürliche Sprache wird in diesem Zusammenhang eine von Menschen gesprochene Sprache bezeichnet, die im Gegensatz zu formalen Sprachen wie Programmiersprachen nicht so kompakt und genau definiert ist [Lew85]; [PK17].

semantischen Analyse dem Text eine Bedeutung zugeordnet. Die Dialog- und Diskursanalyse kommt bei mehreren Sprechern zum Einsatz, dabei werden dann die aufeinanderfolgenden Sätze in ihrer Bedeutung und Beziehung zueinander erfasst. [Art19]; [Mey20]; [RM20]

Im Rahmen dieser Arbeit wird sich mit der Spracherkennung. Die darauffolgenden Analysen des Modells sind für diese Anwendung im Helikopter nicht weiter relevant, da bei der Spracherkennung nur die definierten Befehle erkannt werden müssen. Daher werden diese hier nicht näher betrachtet.

Im Englischen wird dies auch als automatic speech recognition (ASR) bezeichnet. Ein Fall der Spracherkennung ist die Umwandlung von Sprache in Text, auf Englisch speech-to-text (STT). [Mor19]; [Mey20] Dies ist der Schwerpunkt dieser Ausarbeitung.

4.1.3 Einteilung der Spracherkennungssysteme

Spracherkennungssysteme können je nach Einsatzgebiet und Anforderungen in verschiedene Kategorien aufgeteilt werden. Die Abbildung 4.3 zeigt eine Übersicht der verschiedenen Spracherkennungssysteme.

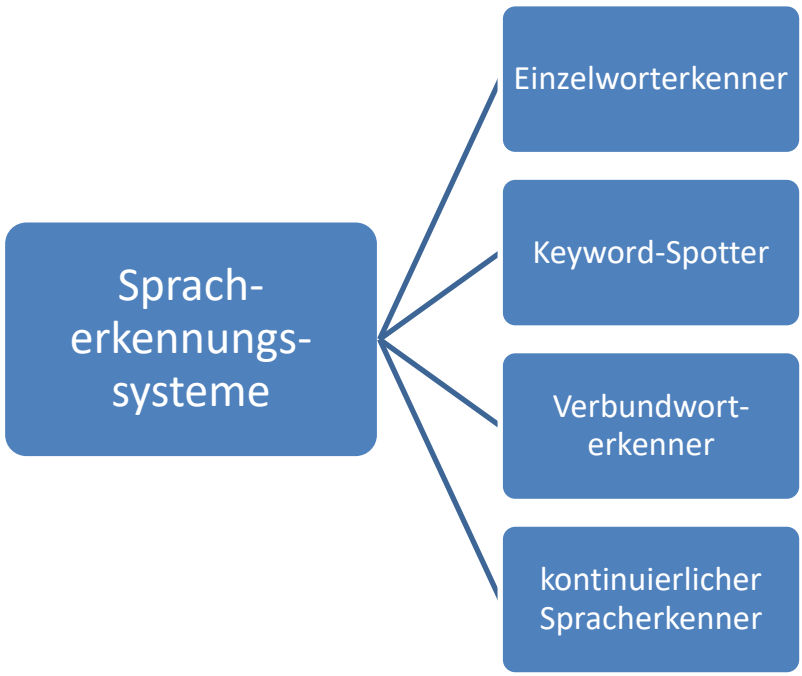

Abbildung 4.3: Spracherkennungssysteme unterteilt nach zu verarbeitenden Äußerungen

Ein Einzelworterkenner erkennt einzelne Wörter oder kurze Kommandos. Diese müssen isoliert gesprochen werden mit Pausen dazwischen. Auch ein Keyword-Spotter ist für die Erkennung einzelner Wörter oder kurzer Kommandos gedacht. Diese müssen jedoch vom System in einer fließenden Aussprache erkannt werden. Bei der Verbundworterkennung werden fließend gesprochene Wörter erkannt. Diese beinhalten jedoch nur ein begrenztes, kleines Vokabular, beispielsweise Zahlen für Telefonnummern. Soll eine natürliche Sprache fließend erkannt werden, so ist ein kontinuierlicher Spracherkenner notwendig. [PK17]

4.1.4 Sprachmodellierung

Bei der menschlichen Sprache gibt es bestimmte Vorgaben und ähnliche Abläufe, beispielsweise in der Grammatik. Durch Vorwissen und Erwartungen kann daher die Erkennung von Sprache verbessert werden. Dies wird mithilfe von Sprachmodellen (engl. language model, LM) modelliert. [PK17]

Hierfür können beispielsweise Hidden-Markov-Modelle angewandt werden [PK17]; [Mey20]. In diesem Zusammenhang werden die N-Gramme kurz vorgestellt. Das sind aufeinanderfolgende Sequenzen, die bei der Modellierung berücksichtigt werden. Die Sequenzen können Wörter, Silben, Phoneme, Buchstaben oder ähnliches sein. In diesem Zusammenhang wird mit aufeinanderfolgenden Wörtern gearbeitet. So wird beispielsweise bei einem Monogramm nur das aktuelle Wort ohne Berücksichtigung der vorherigen betrachtet. Bei einem Bigramm wird das vorherige Wort in die Wahrscheinlichkeitsberechnung für die Erkennung mit einbezogen. [RM20]; [PK17]

4.1.5 Erkennungsleistung

Zur Beurteilung der Erkennungsleistung eines Spracherkenners werden die möglichen Fehler in drei Kategorien aufgeteilt. Dies zeigt die Abbildung 4.4:

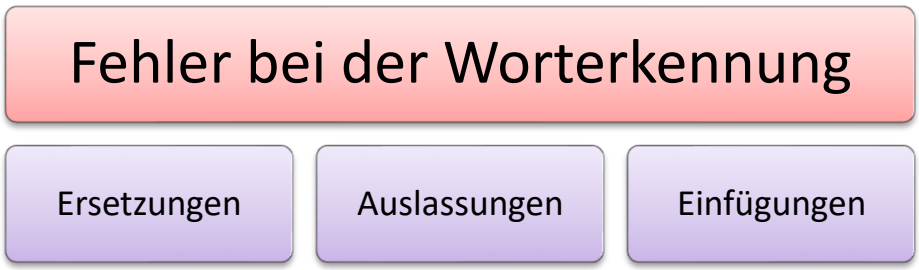

Abbildung 4.4: Fehler bei der Worterkennung

Bei einer Ersetzung wird ein (korrektes) Wort durch ein anderes, falsches Wort ersetzt. Wenn ein Wort durch den Spracherkenner nicht erkannt wird, so nennt man dies Auslassung. Außerdem ist es möglich, dass Wörter hinzugefügt werden durch die Spracherkennung, die nicht gesagt wurden, dies nennt man Einfügung. [PK17]

Aus diesen Fehlerarten lässt sich die Wortfehlerrate (engl.: Word Error Rate, WER) definieren [PK17]; [FS07]:

$$\text{Wortfehlerrate} = \frac{n_{\text{Ersetzungen}} + n_{\text{Auslassungen}} + n_{\text{Einfügungen}}}{n_{\text{zu erkennende Wörter}}} \cdot 100\ \% \qquad (4.1)$$

Die WER ist allem für Texte und längere Sprachsequenzen sinnvoll. Bei der Erkennung einzelner Wörter ist diese schwieriger anzuwenden, da sich oft eine Wortfehlerrate von mehreren hundert Prozent ergibt, wenn beispielsweise ein Wort in drei falsche Wörter erkannt wird. Daher wird in dieser Ausarbeitung oft nur der Anteil der korrekt erkannten Wörter oder Phrasen berücksichtigt.

4.2 DeepSpeech

DeepSpeech ist ein Projekt von Mozilla, dessen Ziel es ist, ein einfaches, offenes und allgegenwärtiges Spracherkennungssystem zu entwickeln. Die Ziele von Mozilla DeepSpeech und deren Bedeutung fasst die Abbildung 4.5 zusammen.

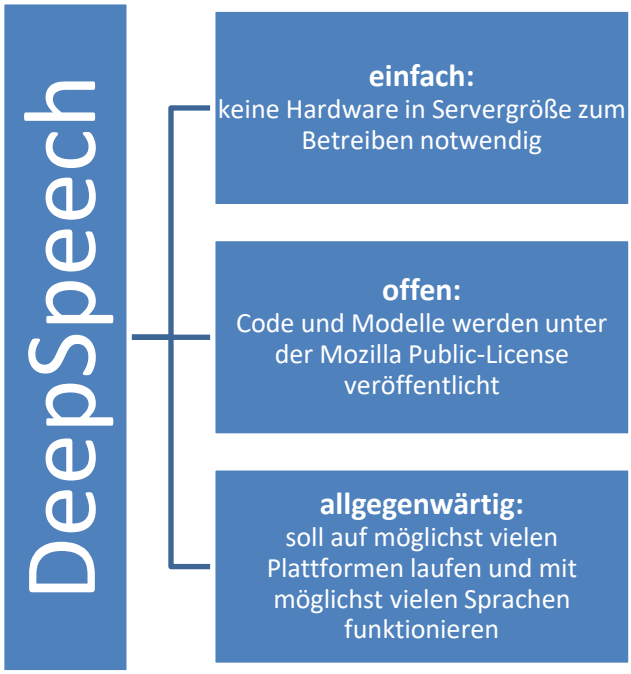

Abbildung 4.5: Ziele von Mozilla DeepSpeech

„Einfach“ bedeutet dabei, dass keine Hardware in Servergröße notwendig sein soll, um die Spracherkennung zu nutzen. „Offen“ ist das Projekt, da der Code und die Modelle unter der Mozilla Public-License veröffentlicht werden und somit zugänglich sind. Als „allgegenwärtige“ Software soll DeepSpeech auf möglichst vielen Plattformen laufen und mit vielen Sprachen funktionieren. [Moz20b]

DeepSpeech wird inzwischen für viele verschiedene Sprachen trainiert. Das englische Modell ist bisher das am meisten trainierte Sprachmodell. Zusätzlich werden von Mozilla auch Sprachdaten gesammelt und von Freiwilligen beschriftet, um die Modelle weiter zu trainieren. [Grü17]

4.2.1 Aufbau von Mozilla DeepSpeech

DeepSpeech ist eine lernbasierte Maschine für die Automatische Spracherkennung. Es ist nach Angabe der Entwickler mit einer einfachen Programmierschnittstelle (API) ausgestattet und wird als quelloffenes Projekt von vielen Entwicklern geschrieben. [Mor19]

Mozilla DeepSpeech hat zwei Hauptbestandteile. Der grobe Aufbau ist in der Abbildung 4.6 dargestellt.

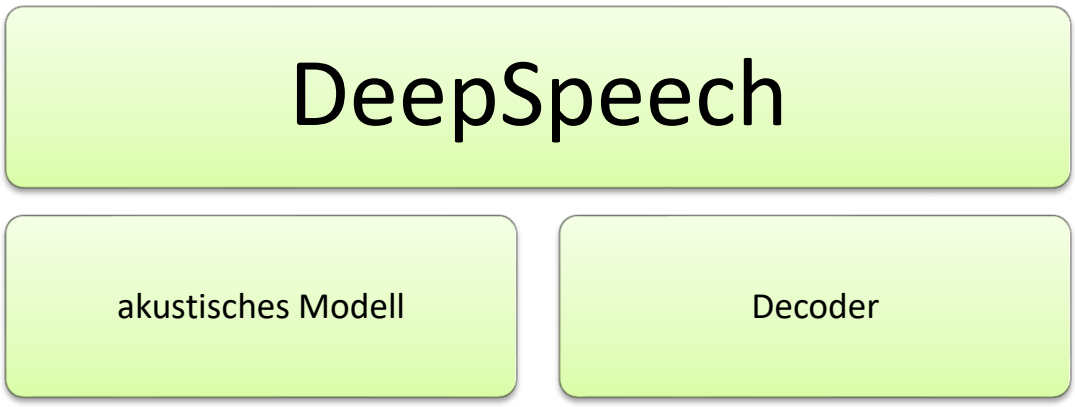

Abbildung 4.6: schematischer Aufbau DeepSpeech

Das akustische Modell ist ein neuronales Netz. Dieses braucht als Input Audiodateien und gibt darauf basierend die Zeichenwahrscheinlichkeiten aus. Mit dem neuronalen Netz beschäftigt sich der Unterabschnitt 4.2.2. Der Decoder ist ein Suchalgorithmus, der aus den Zeichenwahrscheinlichkeiten des akustischen Modells Textteile zusammensetzt. Dazu wird ein Scorer verwendet, der im nächsten Absatz beschrieben wird. Der Aufbau des Decoders lässt sich bei Mozilla [Moz20d] nachlesen. [Mor19]; [Moz20a]; [Moz20d]

Das vortrainierte Modell hat einen externen Scorer zum Berechnen der Wahrscheinlichkeiten von Wort- und Zeichenfolgen. Er kann auch separat erzeugt und dem System vorgegeben werden. Den Aufbau zeigt die Abbildung 4.7. [Moz20c]; [Moz20d]

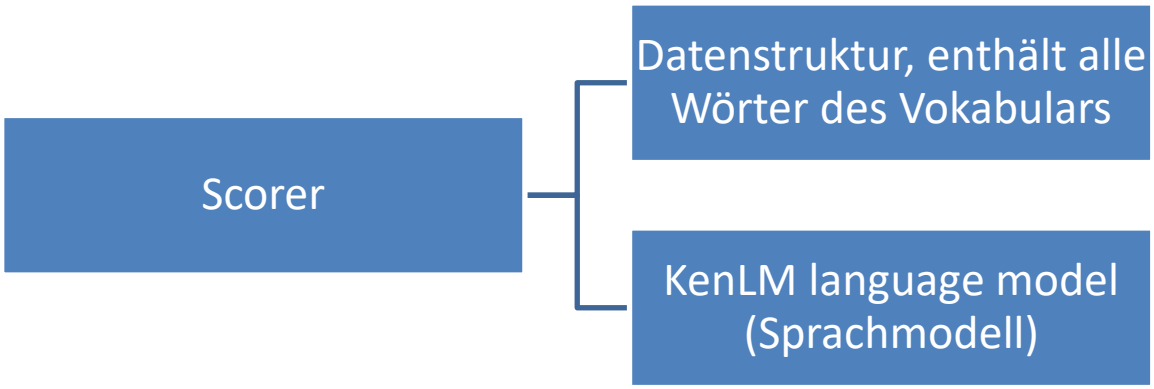

Abbildung 4.7: Aufbau des Scorers von DeepSpeech

Der Scorer wird zur Berechnung der Wahrscheinlichkeiten von Wort- oder Zeichenfolgen der Ausgabe verwendet und besteht grundlegend aus zwei Komponenten. Zum einen gibt es eine Datenstruktur, die alle Wörter enthält, die von dem System erkannt werden sollen. Zum anderen wird ein Sprachmodell (LM) verwendet. Bei DeepSpeech wird dieses mithilfe des Programms KenLM [Git20b]; [Hea20] erzeugt. Dabei werden auch weitere Parameter wie beispielsweise vorgegebenen möglichen Wortreihenfolgen berücksichtigt. [Moz20d]

4.2.2 DeepSpeech als Neuronales Netz

DeepSpeech ist dafür gedacht, durch Training weiterentwickelt zu werden. Dafür besteht es im Kern aus einem rückgekoppelten Neuronalen Netz, es berücksichtigt also zur Berechnung vorherige Erkennungen. Dieses Netz besteht aus fünf Schichten und ist darauf geschult, Sprachdateien in Texte umzuwandeln. Es kann von Grund auf trainiert werden mithilfe von Audiodateien. Zur Erkennung werden die MFCC (vgl. Unterabschnitt 3.2.4) genutzt. Die Aktivierungsfunktion wird mit einem Tangens hyperbolicus (tanh) modelliert. [Moz20b]; [Mor17]

4.2.3 Versionierung und Versionen von Mozilla DeepSpeech

Damit Versionsnummern von Software nach einem einheitlichen Prinzip vergeben werden, folgt diese oft der Bezeichnung: MAJOR.MINOR.PATCH. Bei den Major-Versionen unterscheiden sich die API stark voneinander und sind untereinander nicht kompatibel. Minor-Versionen sind zueinander kompatibel in der gleichen Major-Version und beinhalten neue Funktionalitäten. Wenn nur Fehlerkorrekturen vorgenommen werden, so wird Patch erhöht. [Pre20]

Zum Zeitpunkt dieses Projektes ist DeepSpeech noch in der Major-Version 0 veröffentlicht und befindet sich damit noch in der Entwicklungsphase. Die Minor-Versionen ändern sich bei DeepSpeech innerhalb von einigen Monaten, Patches werden häufiger veröffentlicht. Damit ist DeepSpeech ein agiles Softwarepaket, was sich recht schnell ändert. [Git20c]; [Pre20]

Im Rahmen dieses Projektes wird DeepSpeech 0.7.4 verwendet. Dies ist die letzte Patch-Veröffentlichung der zu Beginn des Projektes aktuellen Minor-Version 0.7. Eine Übersicht über die wichtigsten verwendeten Programme und deren Versionen ist im Anhang G in der Tabelle 7.4 zu finden.

5 Entwicklung und Anpassung der Spracherkennung

Das folgende Kapitel beschäftigt sich mit den im Rahmen dieser Arbeit durchgeführten Ideen, Messungen und deren Auswertung. Dazu werden zunächst die Einflüsse auf die Spracherkennung im Helikopter betrachtet. Dies geschieht im Abschnitt 5.1. Anhand einiger wichtiger Kriterien wird dann im Abschnitt 5.2 ein passendes Spracherkennungsprogramm für die Anwendung ausgewählt und im Abschnitt 5.3 erste Untersuchungen damit beschrieben. Der Abschnitt 5.4 beschäftigt sich mit den Optimierungsmöglichkeiten der Spracherkennung mithilfe des Sprachmodells. Wie die Anpassungen getestet werden, behandelt der Abschnitt 5.5, die Ergebnisse der Tests werden im Abschnitt 5.6 erklärt. Der Abschnitt 5.7 beschäftigt sich mit dem Einfluss des Rauschens auf die Spracherkennung und der Modellierung eines Hubschrauberrauschens. Im Abschnitt 5.8 wird auf den Einfluss des Mikrofons auf die Spracherkennung eingegangen. Die Möglichkeit und Anwendung vom Training des Neuronalen Netzes ist im Abschnitt 5.9 beschrieben.

5.1 Einflüsse auf die Spracherkennung im Helikopter

Die Spracherkennung im Helikopter wird durch einige Einflussfaktoren erheblich erschwert. Die Abbildung 5.1 zeigt eine Auswahl möglicher Einwirkungen auf die Spracherkennung.

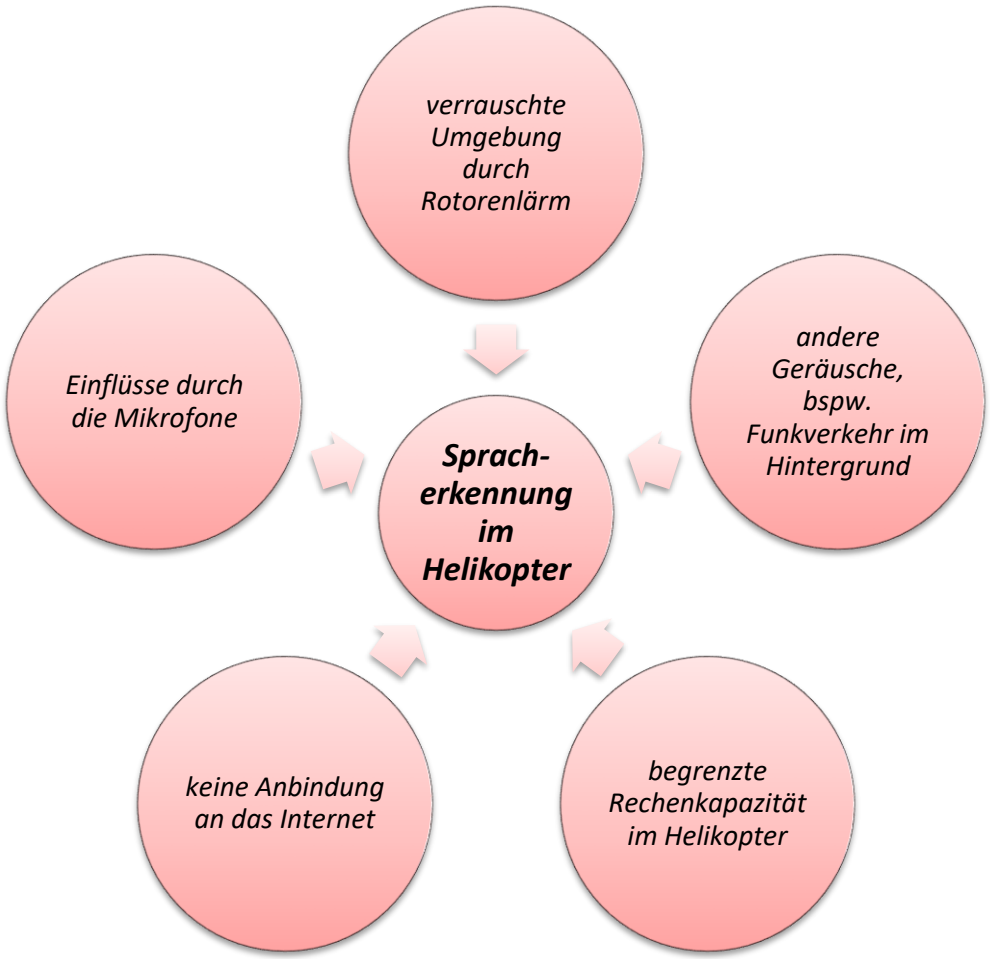

Abbildung 5.1: Mögliche Einflüsse auf die Spracherkennung im Helikopter

Während des Flugs ist es im Innenraum des Helikopters sehr laut, beispielsweise durch die Geräusche der Rotoren. Auch andere Geräusche, wie beispielsweise Funkverkehr, beeinträchtigen die Spracherkennung im Helikopter. Einen weiteren Einfluss können die Mikrofone des Helikopters haben, welche mit einer Rauschsperre ausgestattet sind. Dies ist eine Art der Rauschunterdrückung, bei der erst nach dem Überschreiten eines bestimmten Pegels aufgezeichnet wird. Dadurch wird zu Beginn eines Wortes ein kleiner Teil abgeschnitten. [Neu83] Da keine Anbindung an das Internet besteht, muss das System mit den an Board mitgeführten Geräten funktionieren. Damit ist auch die Rechenleistung stark begrenzt. Dadurch ist auch der Rechenbedarf der Spracherkennung und -verarbeitung beschränkt. [Zei06]

5.2 Auswahl eines Spracherkennungssystems

Für die Spracherkennung im Helikopter muss ein geeignetes Spracherkennungssystem ausgewählt werden, mit dem gearbeitet wird. Dazu werden zunächst die Anforderungen an das Spracherkennungssystem festgelegt. Die Abbildung 5.2 gibt einen Überblick über die wichtigsten Anforderungen.

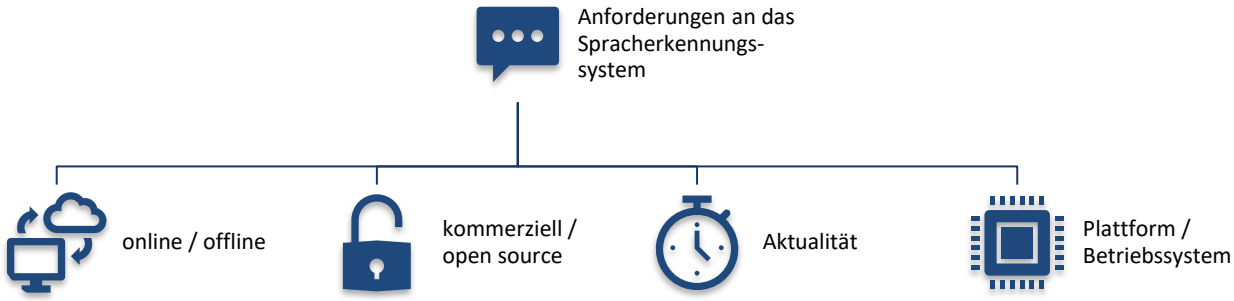

Abbildung 5.2: Anforderungen an das Spracherkennungssystem

Ein Spracherkennungssystem für den Forschungshubschrauber muss in jedem Fall offline funktionieren, da an Bord kein Internet zur Verfügung steht. Außerdem sollte die Anwendung quelloffen sein und nicht kommerziell vertrieben werden. Dies ist wichtig, damit sie ohne größere Einschränkungen optimiert und gut damit geforscht werden kann. Auch steht nur ein begrenztes Budget für das Projekt zur Verfügung. Besonders bei einem aktuellen und schnelllebigen Thema wie der Spracherkennung spielt die Aktualität eine große Rolle. Die Software sollte möglichst kontinuierlich weiterentwickelt und immer wieder an aktuelle Hardware angepasst werden. Außerdem ist es hilfreich, wenn bereits das System möglichst gut erprobt ist und bereits viele Daten zum Training vorhanden sind. Auch die Plattform und die unterstützen Betriebssysteme sind zu beachten. Hier ist zwar keine Vorgabe notwendigerweise einzuhalten, da die Erkennung nicht direkt auf dem Experimentalsystem des Helikopters läuft. Jedoch sollte die Spracherkennung auf einem gängigen PC-Betriebssystem laufen und entwickelt werden können.

Anhand dieser Kriterien wurde am DLR eine Gegenüberstellung unterschiedlicher Programme gemacht. Diese ist im Anhang A dargestellt. Das Softwarepaket DeepSpeech von Mozilla wird daraus ausgewählt, da es offline funktioniert, quelloffen ist, sehr aktuell und auf

unterschiedlichen Betriebssystemen läuft. Weitere Informationen zu Mozilla DeepSpeech sind im Abschnitt 4.2 sowie in weiteren Quellen [Git20d]; [Moz20f] zu finden.

5.3 Erste Messungen und Erkennung

Im Rahmen eines vorhergehenden Praxisprojektes [Roh20] wurden bereits erste Erkennungsversuche mit Mozilla DeepSpeech unternommen. Dort lässt sich auch die Bedienung von Mozilla DeepSpeech nachlesen. Dabei wurde bereits festgestellt, dass das Mikrofon und die Zeit vor den zu erkennenden Wörtern einen Einfluss auf die Erkennungsleistung haben. Daher wird darauf geachtet, dass bei den Tests vor den zu erkennenden Wörtern immer noch Stille hinzugefügt wird, um eine möglichst gute Erkennung zu erzielen. Auch wurde bereits festgestellt, dass die Erkennung mit Mozilla DeepSpeech eine hohe Streuung aufweist. [Roh20]

5.4 Optimierungsmöglichkeiten der Spracherkennung mithilfe des Sprachmodells

Um die Spracherkennung im Forschungshubschrauber zu verbessern, werden einige Maßnahmen ausprobiert. Aufgrund der sehr schlechten Erkennungsleistung bei den Aufnahmen aus dem Hubschrauber (vgl. [Roh20]) wird zunächst mit nachgesprochenen Sprachsamples gearbeitet, um das Sprachmodell zu verbessern. Die Abbildung 5.5 zeigt eine Übersicht über die vorgenommenen Maßnahmen.

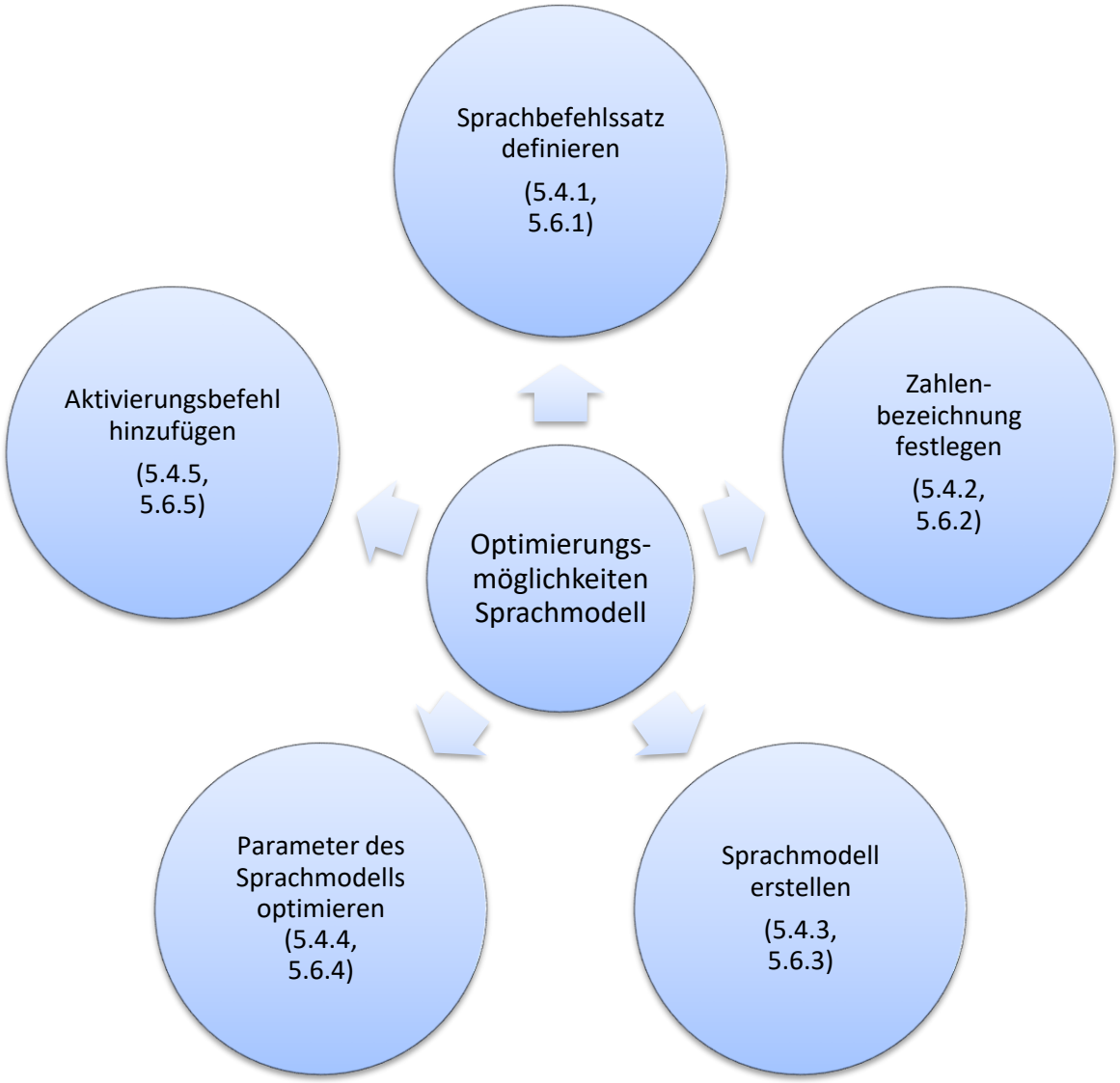

Abbildung 5.3: Optimierungsmöglichkeiten der Spracherkennung mithilfe des Sprachmodells

Um die Spracherkennung zu verbessern, kann ein Sprachbefehlssatz festgelegt werden, der erkannt werden muss. Dies wird im Unterabschnitt 5.4.1 beschrieben. Es gibt für Zahlen unterschiedliche Möglichkeiten der Darstellung, diese werden in Unterabschnitt 5.4.2 vorgestellt und verglichen. Im Unterabschnitt 5.4.3 wird beschrieben, wie ein neues Sprachmodell für die Erkennung erstellt werden kann und warum das sinnvoll ist. Eine weitere Möglichkeit ist der Einsatz eines Aktivierungsbefehls, wie es im Unterabschnitt 5.4.5 erklärt wird. Die Tests der einzelnen Maßnahmen werden im Abschnitt 5.6 ausgewertet.

5.4.1 Definition eines Sprachbefehlssatz

Für die Sprachsteuerung im Forschungshubschrauber ist nur ein begrenzter Befehlssatz notwendig, der erkannt und ausgeführt werden muss. So reicht für die Spracherkennung ein Einzelworterkenner (vgl. Unterabschnitt 4.1.3) aus.

Der Sprachbefehlssatz besteht aus Navigationsbefehlen, Bewegungsbefehlen sowie Zahlen und wurde in Absprache mit den Hubschrauberpiloten nach deren Vorgaben erstellt. Der vollständige Sprachbefehlssatz ist im Anhang A zu finden.

5.4.2 Bezeichnung von Zahlen

Im Sprachbefehlssatz werden sowohl Ziffern als auch bis zu vierstellige Zahlen (0 bis 9999) benötigt. Um diese zu sagen und erkennen zu lassen, gibt es zwei verschiedene Möglichkeiten, wie die Abbildung Abbildung 5.4 zeigt.

Abbildung 5.4: Möglichkeiten der Darstellung von Zahlen

De beiden Möglichkeiten können beispielhaft an der Zahl 4352 veranschaulicht werden. Diese kann entweder als gesamte Zahl, also „four thousand three hundred fifty two“, oder als Ziffernfolge, „four three five two“ gesagt werden. Dabei werden die Beispiele hier auf Englisch gewählt, da die Spracherkennung im Helikopter ebenfalls mit englischen Begriffen realisiert wird.

Beide Darstellungsarten haben Vor- und Nachteile. So ist die Nennung als gesamte Zahl oft etwas einfacher und intuitiver als das einzelne Sagen der Ziffern. Gleichzeitig bedeutet es allerdings auch eine größere Anzahl an Wörtern und Wortkombinationen. Die Darstellung

gesamter, bis zu vierstelliger Zahlen benötigt mit 29 Wörter fast dreimal so viel wie die Ziffern, die mit zehn Wörtern dargestellt werden können. Die Tabelle 7.1 im Anhang C stellt die benötigten Wörter für beide Varianten gegenüber.

5.4.3 Erstellung einer Textgrundlage für ein angepasstes Sprachmodell

Aus dem eigens definierten Sprachbefehlssatz lässt sich ein Sprachmodell (LM) erstellen, welches dann für den Scorer bei der Erkennung verwendet wird. Dabei sollten alle im Sprachbefehlssatz vorkommenden Wörter und möglichst viele der zu erkennenden Wortkombinationen in einem Textdokument zusammengefasst werden.

Dazu wurden zwei unterschiedliche Ansätze entwickelt, um die grundlegende Textdatei für das LM und den Scorer zu erstellen. Diese werden in den folgenden beiden Unterunterabschnitten 5.4.3.1 und 5.4.3.2 beschrieben.

5.4.3.1 Sprachmodell 1:

Die Abbildung 5.5 zeigt das Vorgehen zur Erstellung der Textgrundlage des ersten Sprachmodells.

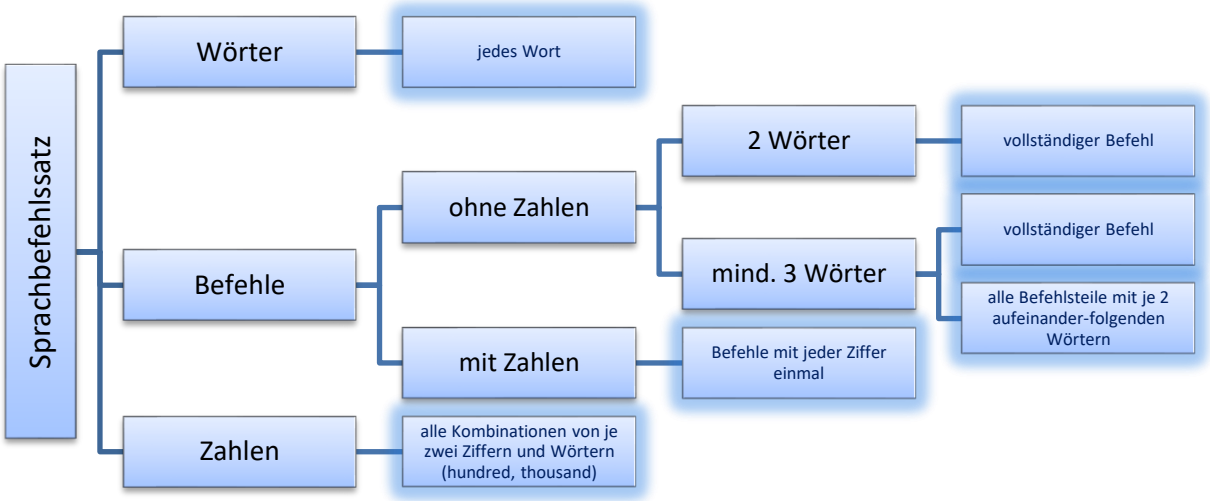

Abbildung 5.5: Vorgehen zur Erstellung der Textgrundlage des Sprachmodells

In blau hinterlegt wird angezeigt, was in die Textdatei aufgenommen wird. Als erstes gehören alle einzelnen Wörter des Sprachbefehlssatzes hinein, danach kommen die Wortkombinationen. Alle Befehle, die keine Zahlen verwenden, werden vollständig aufgenommen. Zusätzlich werden bei den Befehlen, die mindestens drei Wörter umfassen, alle Kombinationen von zwei aufeinanderfolgenden Wörtern aufgenommen. Dies soll eine Erkennung vereinfachen, falls das erste Wort nicht korrekt erkannt wurde. Bei Befehlen mit Zahlen wird der Befehl mit jeder Ziffer einmal in die Aufstellung aufgenommen. Dadurch werden hier nicht alle Möglichkeiten als ein eigener Input aufgenommen, da dies pro Sprachbefehl mehrere tausend Eingaben wären. Stattdessen lassen sich diese alle in Kombination mit den Zahlen zusammensetzen. Auch bei den Zahlen wird nicht jede der tausenden Kombinationen einmal aufgenommen. Stattdessen wird hier wiederum mit Zweierkombinationen gearbeitet, die kombiniert alle möglichen Zahlen ermöglichen.

5.4.3.2 Sprachmodell 2:

Ein zweites Sprachmodell wird erstellt, indem die Befehle spezifischer nach der Anwendung definiert werden. So werden beispielsweise als mögliche Zahlen hinter einer Richtungsangabe alle zwischen null und 360 angegeben, sodass ein Vollkreis abgedeckt ist. Außerdem können bis zu neun Positionen gespeichert werden. Dabei werden immer nur vollständige Befehle aufgenommen. Das Skript zur Erstellung der Befehlszusammensetzungen des zweiten Sprachmodells ist im Anhang E zu finden.

5.4.4 Optimierung der Parameter des Sprachmodells

Bei der Erstellung des Sprachmodells und des Scorers können Parameter festgelegt werden. Diese können ebenfalls für den Anwendungsfall optimiert werden. Dies wird in den folgenden Unterunterabschnitten 5.4.4.1 für das LM und in 5.4.4.2 für den Scorer beschrieben.

5.4.4.1 Erzeugung des Sprachmodells

Die Erzeugung des Sprachmodell ist mithilfe eines Python-Skriptes möglich, welches ebenfalls mit DeepSpeech veröffentlicht ist. Dieses lässt sich beispielsweise folgendermaßen aufrufen: [Moz20e]

```
python3 generate_lm.py --input_txt phrases.txt --output_dir . --
top_k 500000 --kenlm_bins ../../../kenlm/build/bin --arpa_order 5 --
max_arpa_memory "85%" --arpa_prune 0 --binary_a_bits 255 --
binary_q_bits 8 --binary_type trie --discount_fallback
```

Die einzelnen Teile und Parameter des Aufrufs erklärt die Tabelle 5.1. Zusätzlich zu den genannten Quellen wird ebenfalls noch die eingebaute Hilfe-Funktion des Skriptes zur Erklärung genutzt.

Tabelle 5.1: Übersicht über die Eingaben des Befehls generate_lm.py

`python3`	Nutzung der Python3-Version
`generate_lm.py`	Ausführung des Python-Skriptes „generate_lm.py“
`--input_txt phrases.txt`	Inputdatei, Textdatei mit allen Wörtern und den Wortkombinationen des Sprachmodells
`--output_dir .`	Verzeichnispfad, in dem das erstellte Sprachmodell gespeichert werden soll
`--top_k 5000`	Filterparameter
`--kenlm_bins ../../../kenlm/build/bin`	Dateipfad zur Software KenLM, die das Sprachmodell erstellt
`--arpa_order 5`	n-Gramme bei der Erzeugung von ARPA-Dateien
`--max_arpa_memory "85%"`	Maximal zulässiger Speicherverbrauch für die Erzeugung der ARPA-Dateien
`--arpa_prune 0`	ARPA pruning-Parameter
`--binary_a_bits 255`	Binären Quantisierungswert a in Bits aufbauen
`--binary_q_bits 8`	Binären Quantisierungswert q in Bits aufbauen
`--binary_type trie`	Aufbau binärer Datenstrukturen
`--discount_fallback`	Möglichkeit zum Abfangen eines Fehlers

Als erstes werden die zu nutzende Python-Version und das Skript ausgewählt. Für das Skript muss eine Inputdatei mit allen möglichen Wörter und einigen Wortkombinationen, wie zuvor beschrieben, angegeben werden. Der Parameter `top_k` soll die Datengrundlage für das Sprachmodell filtern, um das Sprachmodell zu optimieren [Moz20g]. Dies ist damit vorwiegend für größere Datenmengen wichtig. Es müssen zwischen den Parametern auch Dateipfade zu relevanten Daten bereitgestellt werden. Die Anzahl der n-Gramme wird durch den Parameter `arpa_order` festgelegt. ARPA ist dabei ein Dateiformat [Shm20]. Außerdem kann der maximale Speicher, der für die Dateien genutzt werden darf, begrenzt werden. Der ARPA `pruning`-Parameter bewirkt eine optimierte Anpassung von Verbindungen in neuronalen Netzen [Ban20]. Die `binary-a-bits` und `-q-bits` legen die Anzahl der Quantisierungsstufen fest. Der Aufbau binärer Datenstrukturen wird über `binary_type` festgelegt. Es gibt unterschiedliche Datenstrukturen, wie beispielsweise Listen, Graphen und Bäume. Wenn KenLM meldet, dass eine bestimmte Berechnungsvorschrift für die Parameterberechnung nicht funktioniert, kann mit `discount fallback` dieses Problem abgefangen und durch eine Rückfallebene behoben werden. [Sch07]; [Git18]; [Git20a]; [Moz20e]

Durch das Verändern und Anpassen der beschriebenen Parameter und Testen mit den zusammengesetzten Sprachbefehlen wird das Sprachmodell für die Erkennung der Helikopterdateien bestmöglich erstellt. So ist beispielsweise aufgrund des kleinen Sprachbefehlssatzes ein Zuschneiden (pruning) nicht notwendig. Die schließlich verwendeten Werte sind in der Tabelle 5.1 dokumentiert. Daraus wird dann ein Scorer erstellt, dies wird im nächsten Abschnitt beschrieben.

5.4.4.2 Erzeugung eines eigenen Scorers

Mit dem erzeugten LM kann ein Scorer erzeugt werden, der für die Spracherkennung genutzt wird. Dazu kann der folgende Befehl genutzt werden:

```
python3 generate_package.py --alphabet ../alphabet.txt --lm
lm.binary --vocab vocab-500000.txt --package my_scorer.scorer --
default_alpha 0.931289039105002 --default_beta 1.1834137581510284
```

Die einzelnen Teile des Befehls sowie deren Bedeutung zeigt die Tabelle 5.2.

Tabelle 5.2: Übersicht über die Eingaben des Befehls generate_package.py

`python3`	Nutzung der Python3-Version
`generate_package.py`	Ausführung des Python-Skriptes „generate_package.py)
`--alphabet ../alphabet.txt`	diese befindet sich eine Ebene über dem aktuellen Ordner
`--lm lm.binary`	Pfad und Name der Datei des Sprachmodells
`--vocab vocab-500000.txt`	Textdatei mit den Wörtern
`--package my_scorer.scorer`	Pfad zum Speichern des Scorers
`--default_alpha 0.931289039105002`	Wert des Alpha-Hyperparameter
`--default_beta 1.1834137581510284`	Wert des Beta-Hyperparameter

Zunächst müssen wieder die Python-Version und das Skript ausgewählt werden. Als Alphabet, also die Menge aller möglichen Buchstaben, soll die Datei „alphabet.txt“ verwendet werden. Auf das zu verwendete Sprachmodell und eine Textdatei der wichtigsten Wörter werden ebenfalls verwiesen. Die Werte der Parameter alpha und beta sind Hyperparameter des neuronalen Netzes. Diese können mithilfe eines Skriptes optimiert werden. Dieses Optimizer-Skript ist ebenfalls für Mozilla DeepSpeech veröffentlicht und passt die Hyperparameter für den genutzten Wortschatz an. Dabei werden diese variiert, die Erkennung mit Sprachdateien getestet und die Fehlerrate minimiert.

5.4.5 Hinzufügen eines Aktivierungswortes

Um eine Spracherkennung einzusetzen in einer Umgebung, in der ansonsten auch gesprochen wird, ohne dass dies von der Spracherkennung aufgenommen und erkannt werden soll, muss diese mit einem Befehl aktiviert werden. Dieser wird Aktivierungswort (englisch: hotword) genannt. Wird dieses Wort gesprochen, so soll die Spracherkennung beginnen. [HJP19]; [Rou15] Dies kann ebenfalls für den Hubschrauber eine gute Möglichkeit sein. Dadurch lässt sich beispielsweise ggf. der Funkverkehr etwas besser ausblenden.

5.5 Testmöglichkeiten der Spracherkennung

Um die Spracherkennung zu testen, gibt es mehrere Möglichkeiten. Damit die Tests möglichst realistisch sind, werden die benötigten Wörter einzeln und in gültigen Phrasen getestet. Dies wird in den folgenden beiden Kapiteln beschrieben. Dabei werden alle Tests mit dem Scorer und dem akustischen Sprachmodell von Mozilla DeepSpeech v0.7.4 durchgeführt, soweit nicht anders angegeben.

5.5.1 Erkennung der Einzelwortdateien

Die Wörter des in Unterabschnitt 5.4.1 definierten Sprachbefehlssatzes werden von unterschiedlichen Sprechern außerhalb des Helikopters sowie von Piloten während des Flugs und auf dem Boden aufgesprochen. Die Sprachdateien mit jeweils 44 Wörtern können dann mithilfe von DeepSpeech durchlaufen und die Erkennungsrate bestimmt werden.

Durch Betrachtung der Wörter, welche besonders häufig falsch erkannt werden, wird analysiert, wo evtl. Sprachfehler in der Aussprache oder Ähnlichkeiten auftreten. Dadurch werden die nachgesprochenen Sprachbefehlssätze weiter verbessert.

Die Tabelle 5.3 zeigt den Anteil der richtig erkannten Phrasen von drei unterschiedlichen Sprachbefehlssätzen. Dabei werden die Aufnahmen aus dem Helikopter mit zwei nachgesprochenen Sprachbefehlssätzen (a von Matthias Bodenstein, b von Rilana Rohde) verglichen.

Tabelle 5.3: Erkennungsleistung der Einzelworte bei verschiedenen Sprachbefehlssätzen

Sprachdatensatz (44 Samples)	**Anteil richtig erkannter Wörter**
Aufnahmen aus dem Helikopter	0 %
Nachgesprochener Sprachbefehlssatz a	47,7 %
Nachgesprochener Sprachbefehlssatz b	45,5 %

Die Aufnahmen aus dem Helikopter werden kaum erkannt, von den wenigen Samples, aus denen ein Text extrahiert wird, wird keines richtig erkannt. Bei den nachgesprochenen Sprachbefehlssätzen a und b wird jeweils knapp die Hälfte erkannt.

Als nächstes erfolgt ein Test mit zusammengesetzten Testsamples. Dies wird im Unterabschnitt 5.5.2 beschrieben.

5.5.2 Zusammensetzung von Testsamples

Um die Spracherkennung anwendungsnäher zu testen, werden die zu erkennenden Wörter nicht nur einzeln getestet, sondern auch zu möglichen Sprachsamples zusammengefügt. Diese werden so erstellt, wie sie auch nach den Definitionen des Sprachbefehlssatzes gültig sind. Dabei werden nicht alle möglichen Kombinationen zusammengesetzt, sondern eine Auswahl von 1019 Sprachsamples unterschiedlicher Zusammensetzung. Die Samples werden unmittelbar vor dem Testen aus Einzelwortdateien mit kurzen Pausen vorher und dazwischen zusammengefügt. Dafür wird das Audioprogramm SoX genutzt. Das Skript zum Zusammensetzen der Testsamples ist im Anhang F zu finden. Das Vorgehen wird gewählt, da so viele Phrasen getestet werden können mit einem begrenzten Aufwand zum Erzeugen der Testsamples.

Die Tabelle 5.4 zeigt den Anteil der richtig erkannten Phrasen. Dabei werden wieder die gleichen Sprachbefehlssätze wie in dem Unterabschnitt 5.5.1 genutzt.

Tabelle 5.4: Erkennungsleistung der Phrasen bei verschiedenen Sprachbefehlssätzen

Sprachdatensatz (1019 Phrasen)	**Anteil richtig erkannter Phrasen**
Aufnahmen aus dem Helikopter	0 %
Nachgesprochener Sprachbefehlssatz a	29,3 %
Nachgesprochener Sprachbefehlssatz b	12,7 %

Von den Aufnahmen aus dem Helikopter wird keine Phrase vollständig erkannt. Die nachgesprochenen Sprachbefehlssätze werden teilweise erkannt, die Erkennungsrate liegt dabei zwischen 12,7 % und einem knappen Drittel.

5.6 Messungen mit dem angepassten Sprachmodell

Die im Abschnitt 5.4 beschriebenen Optimierungsmöglichkeiten werden mit den in Abschnitt 5.5 beschriebenen Zusammensetzungen getestet. Zunächst wird der angepasste Sprachbefehlssatz getestet, dies ist in Unterabschnitt 5.6.1 zu lesen. Im Unterabschnitt 5.6.2 ist der Vergleich der Erkennung von Zahlen dargestellt. Die Sprachmodelle der Scorer werden im Unterabschnitt 5.6.3 verglichen. Der Unterabschnitt 5.6.4 stellt die Erkennung mit und ohne Aktivierungswort gegenüber.

5.6.1 Vergleich bei angepasstem Sprachbefehlssatz

Der im Unterabschnitt 5.4.1 beschriebene Sprachbefehlssatz wird als Grundlage für die Erkennung von Einzelwörtern genutzt. Diese wird dann mit der Erkennung bei dem allgemeinen Sprachbefehlssatz verglichen. Die Ergebnisse der Erkennungsanteile sind in der Tabelle 5.5 zu sehen.

Tabelle 5.5: Vergleich der Erkennung mit angepasstem Sprachbefehlssatz

Sprachdatensatz (44 Samples)	**Anteil Erkennung Scorer DeepSpeech 0.7.4**	**Anteil Erkennung eigener Scorer mit angepassten Wörtern**
Aufnahmen aus dem Helikopter	0 %	0 %
Nachgesprochener Sprachbefehlssatz a	47,7 %	79,5 %
Nachgesprochener Sprachbefehlssatz b	45,5 %	100 %

Indem selbst ein Sprachbefehlssatz definiert wird, kann die Erkennung der nachgesprochenen Sprachdateien deutlich verbessert werden. Im Anhang D ist ein Vergleich der Erkennung der benötigten einzelnen Wörter vom Sprachbefehlssatz b zu sehen. Genutzt werden dabei der Scorer von Mozilla DeepSpeech-0.7.4 und ein selbst erzeugter Scorer für den Sprachbefehlssatz. Dieser unterscheidet sich nur in den zu erkennenden Wörtern von dem Scorer von DeepSpeech. Durch den selbst erzeugten Scorer kann der Anteil der erkannten Einzelwörter von etwa 50 % auf bis zu 100 % verbessert werden. Bei den Aufnahmen aus dem Helikopter werden mit dem eigenen Scorer weniger Wörter erkannt als mit der Vorlage von DeepSpeech und weiterhin keines richtig.

5.6.2 Vergleich der Zahlendarstellungen

Bei ersten Erkennungsversuchen mit den in Unterabschnitt 5.4.2 beschriebenen Möglichkeiten, Zahlen zu sagen, wurde festgestellt, dass die Erkennung von zweistelligen Zahlen bei den gesamten Zahlen mehr Fehler aufweist. Die Tabelle 5.6 stellt den Anteil der richtig erkannten Zahlen in unterschiedlichen Zahlenbereichen dar.

Tabelle 5.6: Vergleich der Erkennung mit unterschiedlichen Zahlendarstellungen

Zahlenbereich		**Anteil richtig erkannter Zahlen**
0 - 9	Ziffern	85,3%
10 - 19		63,5%
20 - 90		62,5%
100 - 900	Hunderter	100,0%
1000 – 9000	Tausender	80,6%

Die Erkennung der Ziffern funktioniert mit wenigen Ausnahmen sehr gut. Bei den Zahlen zwischen zehn und 90 werden immer wieder Wörter falsch erkannt. Hier werden beispielsweise ähnlich klingende Wörter wie „thirteen“ und „thirty“ teilweise verwechselt und führen zu Fehlern. Die Hunderter werden vollständig erkannt. Bei den Tausendern ist teilweise die erste Ziffer abgeschnitten. Als Kompromiss aus einer möglichst guten Erkennung und einer möglichst intuitiven und einfachen Bedienungsmöglichkeit wurde entschieden, dass die Zahlen grundsätzlich aus einzelnen Ziffern zusammengesetzt gesagt werden müssen. Dazu kommen die Wörter „hundred“ und „thousand“, sodass glatte Hunderter- und Tausenderzahlen einfach ausgesprochen werden können und nicht mehrere Nullen diktiert werden müssen. Kommen nach der Hunderterstelle noch Ziffern ungleich null, so soll die Zahl in Ziffern angegeben werden.

5.6.3 Vergleich der Scorer mit Sprachmodell 1 und 2

Die beiden neu erstellten Scorer werden mit Sprachbefehlssätzen aus dem Helikopter sowie zwei nachgesprochenen Sprachbefehlssätzen verglichen. Dazu werden die im Unterabschnitt

5.5.2 beschriebenen 1019 Phrasen genutzt. Die Tabelle 5.3 gibt einen Überblick über die erkannten Phrasen.

Tabelle 5.7: Erkennungen mit unterschiedlichen Sprachmodellen

Sprachdatensatz (1019 Phrasen)	**Anteil richtig erkannter Phrasen**		
	Scorer DeepSpeech 0.7.4	**Sprachmodell 1**	**Sprachmodell 2**
Aufnahmen aus dem Helikopter	0 %	0,1 %	0,1 %
Nachgesprochener Sprachbefehlssatz a	29,3 %	100 %	100 %
Nachgesprochener Sprachbefehlssatz b	12,7 %	74,0%	83,6 %

Die Aufnahmen aus dem Helikopter werden mit beiden Sprachmodellen wieder kaum erkannt. Mit dem Sprachbefehlssatz a können alle Phrasen des Tests richtig erkannt werden. Mit dem Sprachbefehlssatz b wird eine Erkennung zwischen 74 % und etwas mehr als 80 % erreicht.

Aufgrund der sehr guten Erkennung der nachgesprochenen Phrasen vom Sprachbefehlssatz a wird dieser für die Untersuchung des Rauscheinflusses genutzt. Diese ist im Abschnitt 5.7 dargestellt.

5.6.4 Vergleich der Erkennung bei angepassten LM-Parametern

Wie im Unterabschnitt 5.4.4 beschrieben, können auch die Parameter bei der Erzeugung des LM und des Scorers angepasst und verbessert werden. Mit den zusammengesetzten

Sprachbefehlen wird die Auswirkung davon getestet. Die Tabelle 5.8 stellt dies den Ergebnissen aus dem vorangegangenen Unterabschnitt 5.6.3 gegenüber.

Tabelle 5.8: Erkennungen mit unterschiedlichen Sprachmodellen

Sprachdatensatz (1019 Phrasen)	**Anteil richtig erkannter Phrasen**			
	Sprachmodell 1		**Sprachmodell 2**	
	Standard-parameter	**Angepasste Parameter**	**Standard-parameter**	**Angepasste Parameter**
Aufnahmen aus dem Helikopter	0,1 %	0,1 %	0,1 %	0,1 %
Nachgesprochener Sprachbefehlssatz a	100 %	100 %	100 %	100 %
Nachgesprochener Sprachbefehlssatz b	74,0 %	76,5 %	83,6 %	82,9 %

Mit den Parametern lassen sich nur bei dem Sprachbefehlssatz b überhaupt Unterschiede erzeugen. Diese sind tendenziell besser als mit den Standardwerten, teilweise jedoch sogar etwas schlechter. Die Unterschiede sind generell so gering, dass diese sich wahrscheinlich eher auf statistische Einflüsse zurückführen lassen.

5.6.5 Vergleich der Spracherkennung mit und ohne Aktivierungswort

Um die Erkennungsrate mit und ohne Aktivierungswort zu vergleichen, wird mit den Einzelwortdateien ein Sprachbefehlssatz mit „command“ als Aktivierungswort und einer ohne zusammengestellt. Dazu wird wieder das Skript im Anhang F genutzt. Zu dem Sprachmodell 2 wird ein identischer Scorer mit dem Aktivierungswort am Anfang erstellt. Die

Tabelle 5.9 stellt die Erkennungen mit und ohne Aktivierungswort für die drei betrachteten Sprachbefehlssätze gegenüber.

Tabelle 5.9: Erkennungen mit und ohne Aktivierungswort

Sprachbefehlssatz (1019 Phrasen)	**Anteil richtig erkannter Phrasen**	
	Ohne Aktivierungswort	**Mit Aktivierungswort**
Aufnahmen aus dem Helikopter	0,1 %	0 %
Nachgesprochener Sprachbefehlssatz a	100 %	100 %
Nachgesprochener Sprachbefehlssatz b	83,6 %	87,4 %

Bei den Aufnahmen aus dem Helikopter wird mit Aktivierungsbefehl keine Phrase mehr erkannt, ohne wurde eine erkannt. Der nachgesprochene Befehlssatz a wird weiterhin vollständig erkannt. Bei dem nachgesprochenen Sprachbefehlssatz b kann die Erkennung um 39 richtig erkannte Phrasen gesteigert werden auf 87,44 %. Insgesamt lässt sich eine leicht positive Tendenz durch den Einsatz eines Aktivierungswortes erkennen. Da kein Unterschied im Sprachbefehlssatz a zu erkennen ist, wird für die folgende Rauschunterdrückung kein Aktivierungswort verwendet.

5.7 Einfluss des Rauschens auf die Spracherkennung

Aufgrund des Antriebs herrscht ein hoher Lärmpegel im Helikopter [Zei06]. Welchen Einfluss dies auf die Spracherkennung hat, soll in diesem Kapitel untersucht werden. Dazu wird im Unterabschnitt 5.7.1 beschrieben, wie das Rauschen des Helikopters nachgebildet werden kann. Die Untersuchung des Rauscheinflusses wird im Unterabschnitt 5.7.2 dargestellt.

5.7.1 Nachbildung des Rauschens im Helikopter

Zur Nachbildung des Rauschens wird die Abbildung 3.2 aus den Grundlagen genutzt. Diese wird mit unterschiedlichen Arten von Rauschen ergänzt. Die Abbildung 5.6 zeigt diese Darstellung.

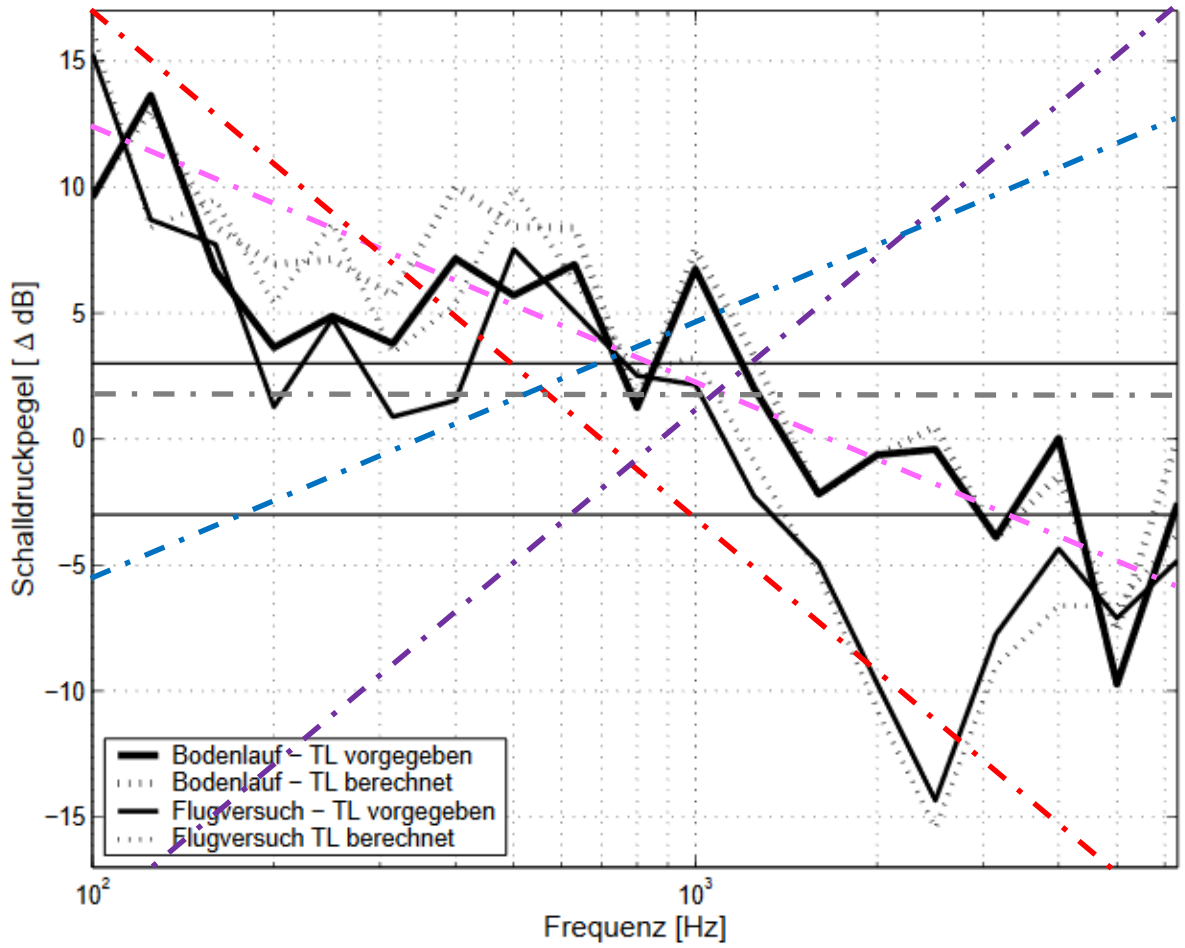

Abbildung 5.6 Innenlärmpegelvergleich bei Bodenlauf und im Schwebeflug [Zei06] im Vergleich mit weißem (in grau), rosa, rotem, blauen und violetten Rauschen

Das rosa Rauschen bildet die Tendenz des Schalldruckpegels grob am besten ab. Daher wird für die Rauschnachbildung des Helikopters für diese Ausarbeitung der Verlauf von rosa Rauschen genutzt. Damit es möglichst allgemein und reproduzierbar ist, wird keine originale Rauschaufnahme aus dem Helikopter verwendet.

5.7.2 Untersuchung des Rauscheinflusses auf die Erkennung

Zu den Phrasen, die nach Unterabschnitt 5.5.2 aus dem Sprachbefehlssatz a zusammengesetzt werden, wird jeweils Rosa Rauschen unterschiedlicher Amplituden hinzugefügt. Die Abbildung 5.7 stellt den Einfluss des hinzugefügten Rosa Rauschens auf die Erkennung dar. Dazu wird die Erkennungsrate der Phrasen über dem SNR (vgl. Unterabschnitt 3.1.3) aufgetragen.

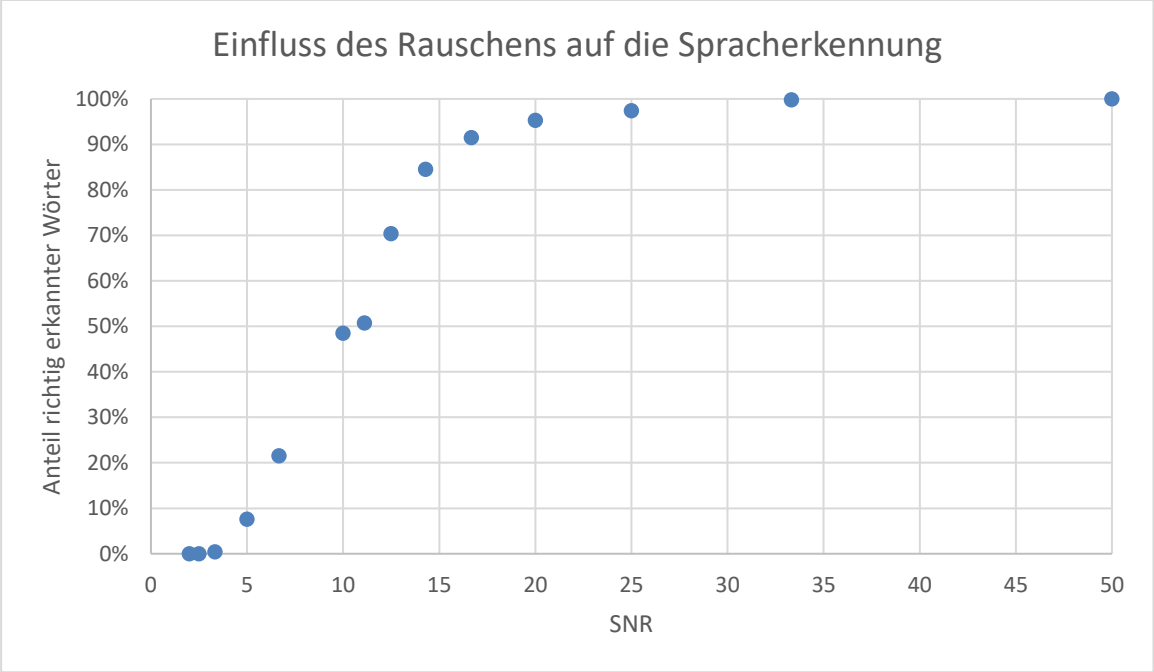

Abbildung 5.7: Einfluss des Rauschens auf die Spracherkennung

Es ist zu sehen, dass das Rauschen einen großen Einfluss auf die Erkennung hat. Bei einem Signal-zu-Rausch-Abstand von über 34 werden die Phrasen vollständig erkannt. Mit sinkendem SNR wird die Erkennung zunehmend schlechter. So wird bei einem Rauschvolumen von einem Zehntel des Signalvolumens (entspricht SNR = 10) nur noch etwa die Hälfte der Phrasen erkannt. Wenn das Signal maximal dreimal so groß ist wie das Rauschen, so wird keine einzige Phrase mehr korrekt erkannt.

5.7.3 Vorbehandlung und Rauschverminderung

Aufgrund des großen Einflusses auf die Spracherkennung sollte das Rauschen bei den Dateien aus dem Helikopter verbessert werden. Um eine beispielhafte Vorbehandlung zu testen, wurden die Dateien mithilfe der Rauschverminderung von Audacity behandelt. Der Aufruf mit den Einstellungen ist in der Abbildung 5.8 zu sehen.

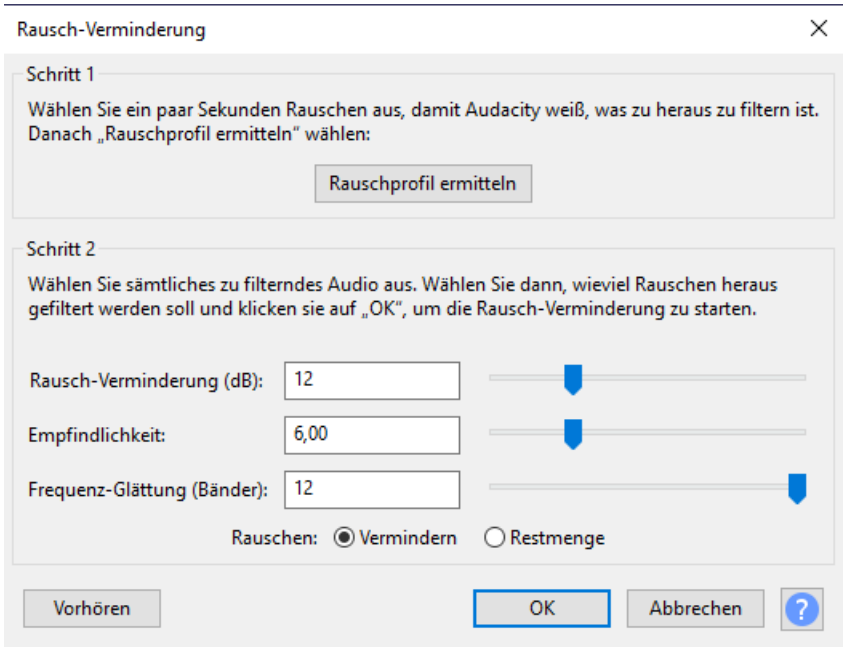

Abbildung 5.8: Rausch-Verminderung Audacity

Dazu wird zunächst das Rauschen des Signals analysiert und dann aus dem Signal herausgerechnet. Die genauen Algorithmen von Audacity sind dabei nicht bekannt. Einen Ausschnitt des zeitlichen Verlaufs des Sprachsignals vor und nach der Rauschunterdrückung zeigt die Abbildung 5.9.

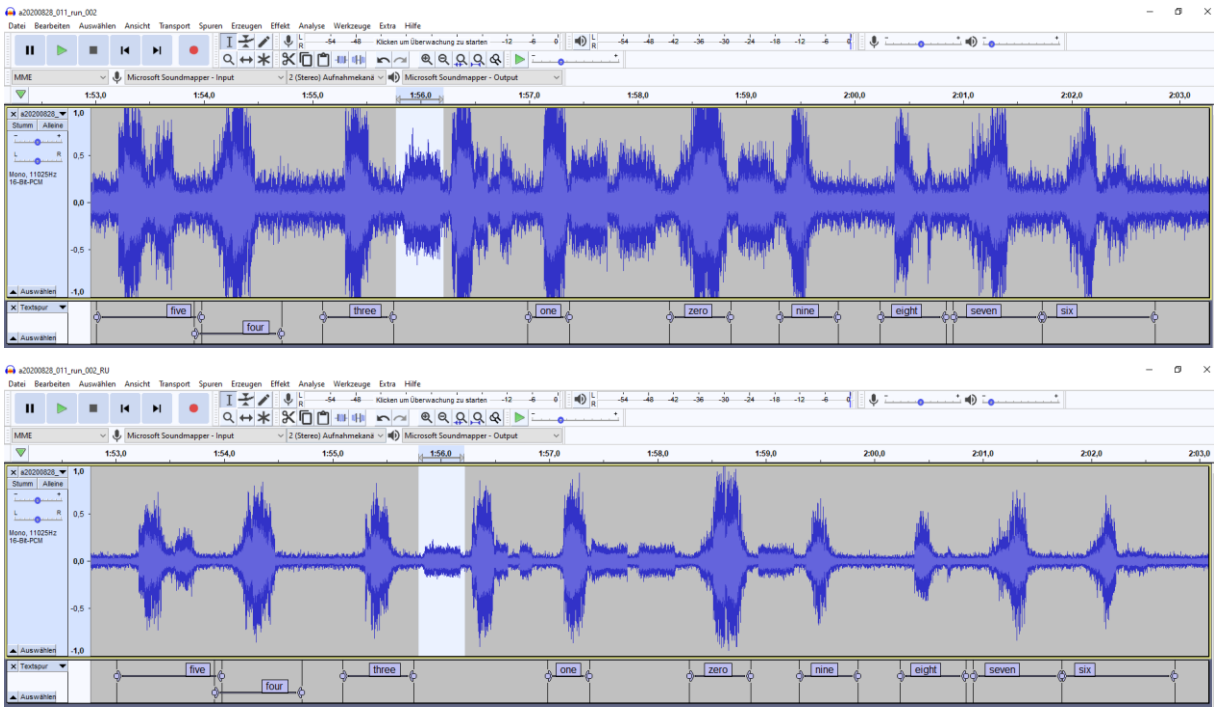

Abbildung 5.9: Zeitlicher Signalverlauf vor (oben) und nach (unten) der Rauschverminderung von Audacity

Dies hat jedoch keine Verbesserung bewirkt, sowohl vor als auch nach der Rauschverminderung wurden keine der 151 getesteten Samples erkannt. Jedoch muss beachtet werden, dass hier die Rauschverminderung ohne Kenntnis der Algorithmen nur angewendet wird. Generell lässt sich sagen, dass auch andere Faktoren großen Einfluss auf die Erkennungsleistung zu haben scheinen.

5.8 Einfluss des Mikrofons auf die Spracherkennung

Bereits in ersten Erkennungsversuchen (vgl. [Roh20]) wurde festgestellt, dass das Mikrofon einen entscheidenden Einfluss auf die Erkennung haben kann. Besonders ein Mikrofon eines Headsets mit Rauschunterdrückung bewirkte eine deutlich schlechtere Spracherkennung. Da im Helikopter ebenfalls ein Mikrofon mit Rauschsperre verwendet wird, kann dies auch Auswirkungen haben. Außerdem wird dadurch der Beginn des Signals abgeschnitten.

Um diese Einflüsse zu untersuchen, könnten unterschiedliche Mikrofone im Helikopter probiert sowie das Mikrofon des Helikopters bei Stillstand getestet werden. Dies ist jedoch

im Rahmen dieses Projektes und aufgrund einer aktuellen Pandemiesituation nicht möglich und kann daher hier nicht betrachtet werden.

5.9 Training des akustischen Modells

Mozilla DeepSpeech besteht im Kern aus einem trainierbaren Neuronalen Netz. Dies soll in diesem Zusammenhang zum Verbessern der Spracherkennung trainiert und eingesetzt werden. Dazu ist zunächst eine Vorbereitung notwendig. Diese und das Training werden im Unterabschnitt 5.9.1 beschrieben. Die Tests und das Ergebnis des trainierten akustischen Modells stellt der Unterabschnitt 5.9.2 dar.

5.9.1 Vorbereitung und Training

Für das Training des Neuronalen Netzes müssen die Trainingsdaten in drei Gruppen aufgeteilt werden: Training, Validierung und Test. Dabei wird in etwa das Verhältnis 70:20:10 % empfohlen. [Tra19]; [MM20]

Dazu werden aus den im Hubschrauber aufgenommenen Sprachsequenzen einzelne Worte extrahiert und möglichst zufällig in die drei Kategorien zugeteilt. Es stehen acht Sprachaufnahmesets zur Verfügung von unterschiedlichen Sprechern und in unterschiedlichen Flugsituationen. Diese Daten werden jeweils mit dem gesprochenen Wort beschriftet und nach den Vorgaben von DeepSpeech für das Training abgespeichert.

Mit dem folgenden Befehl können die Dateien mit der vorbereiteten Umgebung trainiert werden. Dabei soll in diesem Zusammenhang nicht näher auf die einzelnen Parameter eingegangen werden.

```
python3 DeepSpeech.py --n_hidden 2048 --checkpoint_dir
../deepspeech-0.7.4-checkpoint --epochs 25 --train_files
../training/train.csv --dev_files ../training/dev.csv --test_files
../training/test.csv --learning_rate 0.0001 --load_cudnn
```

5.9.2 Testen des trainierten akustischen Modells

Vor und nach dem Training wird das akustische Modell mithilfe der Test-Dateien getestet. Dabei wird die WER des Modells aufgrund der Dateien bestimmt. Die WER kann bei dem durchgeführten Training von 100 % auf 88,5 % verbessert werden.

Mithilfe der Einzelwortdateien sowie den Phrasen wird das akustische Sprachmodell nochmal manuell in der Anwendung getestet. Dazu werden wieder die Aufnahmen aus dem Helikopter sowie die nachgesprochenen Sprachbefehlssätze a und b genutzt. Als Sprachmodell wird das Sprachmodell 2 verwendet. Ein Aktivierungswort sowie Rauschen werden nicht hinzugefügt. Die Tabelle 5.10 zeigt das Ergebnis des Tests mit Einzelworten.

Tabelle 5.10: Erkennungen mit dem trainierten akustischen Modell und der DeepSpeech-Vorlage im Vergleich: Einzelworte

Sprachbefehlssatz (44 Wörter)	**Anteil richtig erkannter Wörter**	
	Akustisches Modell von DeepSpeech 0.7.4	**Trainiertes akustisches Modell**
Aufnahmen aus dem Helikopter	0 %	54,5 %
Nachgesprochener Sprachbefehlssatz a	79,5 %	11,4 %
Nachgesprochener Sprachbefehlssatz b	100 %	0 %

Es ist deutlich zu erkennen, dass die Erkennung der Sprachdateien aus dem Helikopter deutlich gesteigert wurde. Mit dem trainierten akustischen Modell werden über die Hälfte der Wörter richtig erkannt. Gleichzeitig werden fast alle Wörter in Text umgewandelt, was mit dem akustischen Modell von DeepSpeech nicht der Fall ist. Dementsprechend werden viele Dateien falsch statt nicht erkannt. Dies könnte auch von Nachteil sein. Bei den nachgesprochenen Sprachbefehlssätzen ist die Tendenz gegensätzlich. Diese werden kaum bis gar nicht mehr erkannt.

Dabei ist zu beachten, dass die Einzelwortdateien der Helikopteraufnahmen aus den gleichen Umgebungen und von denselben Sprechern stammen wie die Trainingsdateien, mit denen optimiert wurde. Dadurch kann es auch zu Überanpassung kommen. Überanpassung, auf Englisch auch Overfitting genannt, bedeutet, dass ein System zu stark auf Trainingsdaten angepasst ist und dadurch neue Daten nicht mehr gut erkennt. Dadurch lernt ein Netz mehr, sich zu „erinnern“ statt Generalisierungen zu finden. [Ras17]; [Tra19]; [SS19]

Bei einem Test mit zusammengesetzten Phrasen, wie in Unterabschnitt 5.5.2 beschrieben, wurde mit den Einzelwortdateien aus dem Helikopter keine Phrase richtig erkannt. In den meisten Fällen wird nur ein Wort erkannt.

Insgesamt lässt sich sagen, dass für ein wirksames Training deutlich zu wenig Daten zum Trainieren, Validieren und Testen vorhanden sind. Diese Datenmengen im Hubschrauber aufzunehmen ist nicht realistisch durchführbar. Besonders im Rahmen dieser Arbeit und auch des Projektes SALVARE ist das nicht umsetzbar. Dies liegt zum einen an der benötigten Zeit, sowohl für das Aufnehmen der Sprachsamples als auch zum Bearbeiten der Dateien und zum Trainieren des Neuronalen Netzes. Zum anderen sind Flugstunden sehr teuer und es wären viele unterschiedliche Personen zum Einsprechen der Dateien notwendig.

6 Kritische Würdigung und Ausblick

Im folgenden Kapitel sollen die durchgeführten Messungen und Auswertungen kritisch betrachtet werden. Außerdem soll ein Ausblick auf weitere mögliche Messungen und zu untersuchende Einflüsse gegeben werden.

Es wurden einige Einflüsse auf die Spracherkennung herausgefunden und quantifiziert. Jedoch konnten nicht alle möglichen Einflussfaktoren bestimmt werden. Beispielsweise war es nicht möglich, unterschiedliche Mikrofone im Helikopter auszuprobieren.

Der Rauscheinfluss wurde zur allgemeingültigeren Anwendung mit Rosa Rauschen modelliert. Dies ist allerdings eine starke Vereinfachung und kann somit auch zu abweichenden Ergebnissen führen.

Bei den Aufnahmen sind neben dem Hintergrundrauschen noch Stimmen sowie teilweise ein Piepton unterschiedlicher Länge zu hören. Diese können die Spracherkennung auch stark beeinflussen. Auch das Training kann dadurch zu abweichenden Ergebnissen führen.

Um ein richtiges Training des akustischen Sprachmodells durchführen zu können, werden deutlich mehr Testdatensätze benötigt. Außerdem sollten vorher die anderen Einflüsse, wie beispielsweise das bereits genannte Mikrofon im Helikopter, untersucht und für die Spracherkennung optimiert werden.

7 Zusammenfassung

In der vorliegenden Bachelorarbeit wurde untersucht, wie sich eine Spracherkennung für den Forschungshubschrauber ACT/FHS einsetzen lässt und welche Anpassungen dafür notwendig sind. Ein Schwerpunkt lag dabei auf der Verbesserung der Spracherkennung allgemein für diesen Anwendungsfall sowie den Störeinflüssen durch den Helikopter.

Als Spracherkennungsprogramm wurde das freie Softwarepaket DeepSpeech von Mozilla genutzt. Dieses befindet sich noch im fortgeschrittenen Entwicklungsstadium und wird seit einigen Jahren ständig weiterentwickelt.

Mithilfe diverser Anpassungen ließ sich eine gute Erkennung von Wörtern und anwendungsbezogenen Phrasen bei nachgesprochenen, rauschfreien Samples erreichen. Dazu wurden beispielsweise das Sprachmodell angepasst.

Dateien aus dem Helikopter werden weiterhin kaum bis gar nicht erkannt. Eine Rolle dabei spielt das Rauschen, welches einen großen Einfluss auf die Erkennungsleistung hat. Dies sollte besser gefiltert werden, dazu ist evtl. eine Vorbehandlung notwendig. Mit einer beispielhaft angewandten Rauschverminderung von Audacity ließ sich zunächst keine nennenswerte Verbesserung erzielen.

Ein weiterer großer Einflussfaktor scheint das Mikrofon im Helikopter zu sein. Dies konnte jedoch im Rahmen dieser Arbeit nicht untersucht werden. Auch die Hintergrundgeräusche wirken sich auf die Erkennung aus.

Mit einem Training des Neuronalen Netzes ließe sich die Erkennung wahrscheinlich verbessern. Hierfür wären große Mengen an Sprachdaten von Helikopterflügen notwendig. Im Rahmen dieser Arbeit und des Projektes SALVARE war das nicht umsetzbar.

Insgesamt wurden bereits einige Einflüsse quantifiziert und die Spracherkennung mit Mozilla DeepSpeech für den Anwendungsfall verbessert. Außerdem wurden mögliche Parameter und Einflussfaktoren erkannt, die ggf. in weiterführenden Untersuchungen analysiert werden können.

Literaturverzeichnis

Deckblatt: [DLR20b]

[Art19] *Artificial Creativity:* Natural Language Processing (NLP). *URL https://katzlberger.ai/glossar-kuenstliche-intelligenz/natural-language-processing-nlp/. – Aktualisierungsdatum: 2019-09-23 – Überprüfungsdatum 2020-10-11*

[Ban20] *BANDARU, Rohit: Pruning Neural Networks - Towards Data Science. In: Towards Data Science (2020-09-01)*

[Bit05] *BITTNER, Walter: Flugmechanik der Hubschrauber: Technologie, das flugdynamische System Hubschrauber, Flugstabilitäten, Steuerbarkeit. 2., aktualisierte Aufl. Berlin: Springer, 2005 (VDI-Buch)*

[Boo20] *BOOS, Anna: DLR - Institut für Flugsystemtechnik - ACT/FHS-Versuchsanlage. URL https://www.dlr.de/ft/desktopdefault.aspx/tabid-10349/17736_read-42134. – Aktualisierungsdatum: 2020-05-30 – Überprüfungsdatum 2020-05-30*

[Deu20a] *Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR): Das DLR. URL https://www.dlr.de/DE/organisation-dlr/das-dlr/das-dlr.html. – Aktualisierungsdatum: 2020-05-30 – Überprüfungsdatum 2020-05-30*

[Deu20b] *Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR): Institute und Einrichtungen im Kurzporträt. URL https://www.dlr.de/DE/organisation-dlr/das-dlr/institute-und-einrichtungen.html. – Aktualisierungsdatum: 2020-05-30 – Überprüfungsdatum 2020-05-30*

[Deu20c] *Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR): ACT/FHS-Versuchsanlage. URL https://www.dlr.de/content/de/grossforschungsanlagen/act-fhs-versuchsanlage.html. – Aktualisierungsdatum: 2020-05-31 – Überprüfungsdatum 2020-05-31*

[Deu20d] *Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR): Airbus Helicopter EC 135 ACT/FHS. URL https://www.dlr.de/content/de/grossforschungsanlagen/airbus-helicopter-ec-135-act-fhs.html. – Aktualisierungsdatum: 2020-05-31 – Überprüfungsdatum 2020-05-31*

[DLR20a] *DLR Portal: Rettungshubschrauber 2030. URL https://www.dlr.de/content/de/artikel/luftfahrt/leitkonzepte/rettungshubschrauber-2030.html. – Aktualisierungsdatum: 2020-10-07 – Überprüfungsdatum 2020-10-07*

[DLR20b] DLRARTICLE DLR Portal: Airbus Helicopter EC 135 ACT/FHS. URL https://www.dlr.de/content/de/grossforschungsanlagen/airbus-helicopter-ec-135-act-fhs.html. – Aktualisierungsdatum: 2020-11-13 – Überprüfungsdatum 2020-11-13

[Dre19] DREWS, Michael: DLR - Institut für Flugführung - SALVARE (Safe Landing and Takeoff in low Visibility for advanced Rescue Operations). URL https://www.dlr.de/fl/desktopdefault.aspx/tabid-1149/1737_read-66162/. – Aktualisierungsdatum: 2020-09-07 – Überprüfungsdatum 2020-09-07

[FS07] FLOCK, Peter; SPIEKER, Helge: Spracherkennung. Hochschule Bonn-Rhein-Sieg. 15.12.2007 – Überprüfungsdatum 2020-11-01

[FB08] FROHBERG, Wolfgang; BLUSCHKE, Andreas: Taschenbuch der Nachrichtentechnik: Mit 57 Tabellen. 1. Aufl. s.l.: Carl Hanser Fachbuchverlag, 2008

[Git18] GitHub: binary-type-tree. URL https://github.com/marcusjwhelan/binary-type-tree. – Aktualisierungsdatum: 2018-01-16 – Überprüfungsdatum 2020-11-11

[Git20a] GitHub: fix building ARPA file (building_lm using kenlm) by d-a-v · Pull Request #2945 · mozilla/DeepSpeech. URL https://github.com/mozilla/DeepSpeech/pull/2945. – Aktualisierungsdatum: 2020-04-25 – Überprüfungsdatum 2020-11-11

[Git20b] GitHub: kpu/kenlm. URL https://github.com/kpu/kenlm. – Aktualisierungsdatum: 2020-11-04 – Überprüfungsdatum 2020-11-13

[Git20c] GitHub: Releases · mozilla/DeepSpeech. URL https://github.com/mozilla/DeepSpeech/releases. – Aktualisierungsdatum: 2020-11-04 – Überprüfungsdatum 2020-11-08

[Git20d] GitHub: mozilla/DeepSpeech. URL https://github.com/mozilla/DeepSpeech. – Aktualisierungsdatum: 2020-11-05 – Überprüfungsdatum 2020-11-13

[Grü17] GRÜNER, Sebastian: Deep Speech und Common Voice: Mozilla bringt freie Spracherkennung für alle - Golem.de. In: Golem.de (2017-11-30)

[Ham16] Hameg Instruments: Was ist Rauschen: Fachartikel Spektrumsanalyse. URL https://cdn.rohde-schwarz.com/hameg-archive/HAMEG_Rauschen.pdf – Überprüfungsdatum 2020-11-10

[HJP19] HANSEN, Sven; JURRAN, Nico; PORTECK, Stefan: Sprachassistenten durchdringen den Alltag. In: heise Online (2019-09-13)

[Hea20] HEAFIELD, Kenneth: KenLM Language Model Toolkit. kenlm . code . URL https://kheafield.com/code/kenlm/. – Aktualisierungsdatum: 2020-11-10 – Überprüfungsdatum 2020-11-13

[Hei20] HEILMANN, Rolf: Rauschen in der Sensorik. 1. Auflage 2020. Wiesbaden: Springer Fachmedien Wiesbaden GmbH; Springer Vieweg, 2020

[HRL81] HERTER, Eberhard; RÖCKER, Walter; LÖRCHER, Wolfgang: Nachrichtentechnik: Übertragung, Vermittlung u. Verarbeitung. Von Eberhard Herter, Walter Röcker u. Wolfgang Lörcher. 2., erw. u. überarb. Aufl. München, Wien: Hanser, 1981 ((Studienbücher der technischen Wissenschaften))

[Kar17] KARRENBERG, Ulrich: Signale - Prozesse - Systeme: Eine multimediale und interaktive Einführung in die Signalverarbeitung. 7., neu bearbeitete und erweiterte Auflage. Berlin: Springer Vieweg, 2017

[LL19] LANGE, Jörg; LANGE, Tatjana: Fourier-Transformation zur Signal- und Systembeschreibung: Kompakt, visuell, intuitiv verständlich. Wiesbaden: Springer Fachmedien Wiesbaden, 2019 (essentials)

[LSW09] LERCH, Reinhard; SESSLER, Gerhard Martin; WOLF, Dietrich: Technische Akustik: Grundlagen und Anwendungen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009

[Lew85] LEWANDOWSKI, Theodor: Linguistisches Wörterbuch. 4., neubearb. Aufl. Heidelberg: Quelle & Meyer, 1985 (UTB für Wissenschaft Uni-Taschenbücher Linguistik 300)

[Lip10] LIPINSKI, Klaus: Dekade: decade. URL https://www.itwissen.info/Dekade-decade.html. – Aktualisierungsdatum: 2010-05-17 – Überprüfungsdatum 2020-10-29

[Lip15] LIPINSKI, Klaus: Rosa Rauschen: pink noise. URL https://www.itwissen.info/Rosa-Rauschen-pink-noise.html. – Aktualisierungsdatum: 2015-09-23 – Überprüfungsdatum 2020-10-28

[Mey20] MEYER, Carsten: Automatische Sprachverarbeitung: Vorlesungsskript. Wolfenbüttel, WiSe 2020/2021

[Mey17] MEYER, Martin: Signalverarbeitung: Analoge und digitale Signale, Systeme und Filter. 8., verbesserte Auflage. Wiesbaden: Springer Vieweg, 2017 (Lehrbuch)

[Mor17] MORAIS, Reuben: A Journey to <10% Word Error Rate – Mozilla Hacks - the Web developer blog. URL https://hacks.mozilla.org/2017/11/a-journey-to-10-word-error-rate/. – Aktualisierungsdatum: 2017-11-29 – Überprüfungsdatum 2020-05-31

[Mor19] MORAIS, Reuben: DeepSpeech 0.6: Mozilla's Speech-to-Text Engine Gets Fast, Lean, and Ubiquitous – Mozilla Hacks - the Web developer blog. URL https://hacks.mozilla.org/2019/12/deepspeech-0-6-mozillas-speech-to-text-engine/. – Aktualisierungsdatum: 2019-12-05 – Überprüfungsdatum 2020-05-31

[Moz20a] Mozilla Corporation: CTC beam search decoder with external scorer — DeepSpeech 0.7.1 documentation. URL https://deepspeech.readthedocs.io/en/v0.7.1/Decoder.html. – Aktualisierungsdatum: 2020-05-18 – Überprüfungsdatum 2020-11-08

[Moz20b] Mozilla Corporation: DeepSpeech Model — DeepSpeech 0.7.3 documentation. URL https://deepspeech.readthedocs.io/en/v0.7.3/DeepSpeech.html. – Aktualisierungsdatum: 2020-06-04 – Überprüfungsdatum 2020-08-18

[Moz20c] Mozilla Corporation: Training Your Own Model — DeepSpeech 0.7.3 documentation. URL https://deepspeech.readthedocs.io/en/v0.7.3/TRAINING.html. – Aktualisierungsdatum: 2020-06-04 – Überprüfungsdatum 2020-07-16

[Moz20d] Mozilla Corporation: CTC beam search decoder: DeepSpeech 0.7.4 documentation. URL https://deepspeech.readthedocs.io/en/v0.7.4/Decoder.html. – Aktualisierungsdatum: 2020-06-18 – Überprüfungsdatum 2020-10-07

[Moz20e] Mozilla Corporation: External scorer scripts: DeepSpeech 0.7.4 documentation. URL https://deepspeech.readthedocs.io/en/v0.7.4/Scorer.html. – Aktualisierungsdatum: 2020-06-18 – Überprüfungsdatum 2020-11-01

[Moz20f] Mozilla Corporation: Welcome to DeepSpeech's documentation!: DeepSpeech 0.7.4 documentation. URL https://deepspeech.readthedocs.io/en/v0.7.4/. – Aktualisierungsdatum: 2020-06-19 – Überprüfungsdatum 2020-11-13

[Moz20g] Mozilla Discourse: DeepSpeech Language Model parameters. URL https://discourse.mozilla.org/t/deepspeech-language-model-parameters/67230/5. – Aktualisierungsdatum: 2020-09-13 – Überprüfungsdatum 2020-11-01

[MM20] MUELLER, John Paul; MASSARON, Luca: Deep Learning kompakt für Dummies. 1. Auflage. Weinheim: Wiley-VCH, 2020 (… für Dummies)

[MS05] MÜNTER-ELFNER, Mathias (Hrsg.); SCHIEFELBEIN, Nina (Hrsg.): Jugend Brockhaus in drei Bänden: A - Z. 6., neu bearb. Aufl. Mannheim: Brockhaus, 2005

[Nag20] NAGEL, Dörthe: DLR - Institut für Flugsystemtechnik - Institut für Flugsystemtechnik. URL https://www.dlr.de/ft/desktopdefault.aspx/tabid-1336/1837_read-32330/. – Aktualisierungsdatum: 2020-05-30 – Überprüfungsdatum 2020-05-30

[Neu83] NEUFANG, O. (Hrsg.): Lexikon der Elektronik. Wiesbaden: Vieweg+Teubner Verlag, 1983

[Nie03] NIEMANN, Heinrich: Klassifikation von Mustern. 2. überarbeitete und erweiterte Auflage. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003

[OL05] OHM, Jens-Rainer; LÜKE, Hans Dieter: Signalübertragung: Grundlagen der digitalen und analogen Nachrichtenübertragungssysteme. 9., bearb. Aufl. Berlin: Springer, 2005 (Springer-Lehrbuch)

[PK17] PFISTER, Beat; KAUFMANN, Tobias: Sprachverarbeitung: Grundlagen und Methoden der Sprachsynthese und Spracherkennung. 2., aktualisierte und erweiterte Auflage. Berlin: Springer Vieweg, 2017

[Pre20] PRESTON-WERNER, Tom: Semantic Versioning 2.0.0. URL https://semver.org/lang/de/#spec-item-4. – Aktualisierungsdatum: 2020-11-02 – Überprüfungsdatum 2020-11-08

[PJ19] PUENTE LEÓN, Fernando; JÄKEL, Holger: Signale und Systeme. 7., überarbeitete Auflage. Berlin/Boston, Berlin: Walter de Gruyter GmbH, 2019 (De Gruyter Studium)

[RM20] RAO, Delip; MCMAHAN, Brian: Natural language processing mit PyTorch: Intelligente Sprachanwendungen mit Deep Learning erstellen. 1. Auflage. Heidelberg: dpunkt.verlag GmbH; O´Reilly, 2020

[Ras17] RASHID, Tariq: Neuronale Netze selbst programmieren: Ein verständlicher Einstieg mit Python. 1. Auflage. Heidelberg: O'Reilly, 2017 (Animals)

[Roh20] ROHDE, Rilana: Untersuchung der Spracherkennung unter erschwerten Bedingungen mit Mozilla DeepSpeech. Wolfenbüttel, Ostfalia Hochschule für angewandte Wissenschaften. Praxisprojekt. 08/2020

[Rou15] ROUSE, Margaret: hotword. In: TechTarget (2015-06-25)

[rth20] RTH.INFO, Team: rth.info - Faszination Luftrettung | Hubschraubertyp Airbus Helicopters EC 135. URL https://www.rth.info/typen/typen.php?show=ec135. – Aktualisierungsdatum: 2020-05-30 – Überprüfungsdatum 2020-05-30

[Sch07] SCHNITGER, Georg: Datenstrukturen. Frankfurt. Skript zur Vorlesung. 2007 – Überprüfungsdatum 2020-11-01

[Sch95] SCHUKAT-TALAMAZZINI, Ernst Günter: Automatische Spracherkennung: Statistische Verfahren der Musteranalyse (1995)

[SS19] *SCHWAIGER, Roland; STEINWENDNER, Joachim: Neuronale Netze programmieren mit Python. 1. Auflage, 2019 (Rheinwerk Computing)*

[Sem16] *SEMMLER, Tom: HiFi-Voodoo: Einbrennen. URL http://www.highresmac.de/hifi-voodoo-einbrennen/. – Aktualisierungsdatum: 2017-03-18 – Überprüfungsdatum 2020-10-28*

[Shm20] *SHMYREV, Nickolay: ARPA Language models. URL https://cmusphinx.github.io/wiki/arpaformat/. – Aktualisierungsdatum: 2020-06-21 – Überprüfungsdatum 2020-11-01*

[SSK14] *SINAMBARI, Gholam Reza; SENTPALI, Stefan; KUNZ, Frieder: Ingenieurakustik: Physikalische Grundlagen und Anwendungsbeispiele. 5., völlig überarb. u. erw. Aufl. Wiesbaden: Springer Vieweg, 2014 (SpringerLink)*

[Spe14] *Spektrum der Wissenschaft Verlagsgesellschaft mbH: Schall. URL https://www.spektrum.de/lexikon/physik/schall/12766. – Aktualisierungsdatum: 2014-12-04 – Überprüfungsdatum 2020-11-01*

[Tra19] *TRASK, Andrew W.: Neuronale Netze und Deep Learning kapieren: Der einfache Einstieg mit Beispielen in Python. 2019. Auflage. Frechen: MITP, 2019 (mitp Professional)*

[Wie20] *WIESER, Wolfgang: AVES – Air Vehicle Simulator. URL https://www.dlr.de/ft/desktopdefault.aspx/tabid-1387/1915_read-38610/. – Aktualisierungsdatum: 2020-10-28 – Überprüfungsdatum 2020-10-28*

[Wut19] *WUTTKE, Laurenz: Künstliche Neuronale Netzwerke: Definition, Einführung, Arten und Funktion. In: datasolut GmbH (2019-10-17)*

[Zei06] *ZEITLER, Andreas: Untersuchung der Hubschrauberinnenakustik mittels der Methode der statistischen Energieanalyse. München, Technische Universität München, Lehrstuhl für Leichtbau. Dissertation. 2006. URL https://mediatum.ub.tum.de/doc/601995/601995.pdf*

[ZBH87] *ZINKE, Otto (Hrsg.); BRUNSWIG, Heinrich (Hrsg.); HARTNAGEL, Hans L. (Hrsg.): Lehrbuch der Hochfrequenztechnik: Elektronik und Signalverarbeitung. 3., neubearb. u. erw. Aufl. Berlin: Springer, 1987 (Lehrbuch der Hochfrequenztechnik / Zinke; Brunswig; Bd. 2)*

Anhang

A Gegenüberstellung von Spracherkennungssystemen

Tabelle von Jeldrick Powitz und Matthias Bodenstein, DLR

Programm	Website	Offline / Online	OpenSource / Kommerziell	Aktivierung	Eigenschaften	Anforderungen	Aktualität	Betriebssystem
Jarvis	openjarvis.com	offline und online --> verschiedene Dienste möglich, "snowboy" empfohlen	OpenSource	Hotword	auch Sprachausgabe möglich über verschiedene Dienste, "SVOX pico" empfohlen		03.01.2018	Raspian (Rasperry Pi) MacOS
Jasper	jasperproject.github.io		OpenSource	Hotword	modularer, flexibler Aufbau, aber wohl schwierige konfiguration		3-5 Jahre alt	
Mycroft	mycroft.ai	online --> muss mit Entwicklerseite verbunden sein, Daten werden auf Wunsch aber nicht mitgehört/gespeichert ==> basiert auf DeepSpeech	OpenSource		scheinbar recht simpel zu installieren		aktuell	
Kalliope	kalliope-project.github.io/kalliope	offline und online --> verschiedene Dienste möglich, auch "snowboy"	OpenSource		modularer Aufbau		aktuell	RaspberryPi Debian Ubuntu
Simon	simon.kde.org	Spracherkennungs-Engine "Julius"	OpenSource		soll Rechner steuern --> für Simulator daher gut nutzbar?		2013	Linux Windows
DeepSpeech	https://github.com/mozilla/DeepSpeech	offline	OpenSource	Taste | Dauer-aufnahme	Echtzeitanwendung möglich, eigene Bibliotheken können eingelernt werden, GPU-Version ist schneller		aktuell	Linux MacOS Windows (eingeschränkt)
Kaldi	https://github.com/kaldi-asr/kaldi		OpenSource		Von FL in der Towersimulation verwendet		aktuell	
wav2letter++	github.com/facebookresearch/wav2letter		OpenSource					
snips	https://snips.ai/	offline	Kommerziell		eher für SmartHome ausgelegt mit GUI		aktuell	
picovoice	https://picovoice.ai	offline					aktuell	
Dragon Software Developer Kit	https://www.nuance.com/de-de/dragon/for-developers/dragon-software-developer-kit.html		Kommerziell					
LinTO	https://linto.ai/enterprises#linto-linstt	offline						
Stephanie	https://slapbot.github.io/		OpenSource				1-2 Jahre alt	
Pocketsphinx	https://cmusphinx.github.io/	offline	OpenSource				2016	
CMUSphinx	https://www-i6.informatik.rwth-aachen.de/rwth-s	offline	OpenSource				2016	
RASR	https://www.microsoft.com/en-us/download/details.aspx?id=52313						2016	
SCARF	https://github.com/PRHLT/iatros-decoder						2016	
iATROS								

B Sprachbefehle

```
Bewegungsbefehle:
-----------------

SPEED SET <value>
SPEED UP <value>
SPEED DOWN <value>

ALTITUDE SET <value>
ALTITUDE UP <value>
ALTITUDE DOWN <value>

HEADING SET <value> | TARGET
HEADING UP <value>
HEADING DOWN <value>

#nur im Hovermodus
POSITION AHEAD <value>
POSITION BACK <value>
POSITION LEFT <value>
POSITION RIGHT <value>
POSITION UP <value>
POSITION DOWN <value>

MODE SET HOVER | FLY | <digit>

TARGET SET HOSPITAL | BASE | POSITION <digit>
```

```
Navigationsbefehle:

-------------------


SET DECISION HEIGHT <value>

SET SPEED BUG <value>

SET HEADING BUG <value>


WAYPOINT <value> | NEXT | PREVIOUS


Bordcomputer:


RUN START | STOP

SET DISPLAY <value> | HOVER

SELECT CHECKLIST <value>

SELECT PROCEDURE <value>


Aktivierung:

COMMAND


#Definition von <value>:


1.) Folge von Ziffern

<digit> [<digit> [<digit> [<digit>]]]


2.) zusammengesetzte Zahl:

[<digit> THOUSAND] [<digit> HUNDRED]


#Definition von <digit>:

ONE | TWO | THREE | FOUR | FIVE | SIX | SEVEN | EIGHT | NINE |
ZERO
```

C Wortschatz für die Zahlendarstellung

Tabelle 7.1: Gegenüberstellung der notwendigen Wörter für unterschiedliche Darstellungsarten von Zahlen

Gesamte Zahl	**Ziffernfolge**
zero	zero
one	one
two	two
three	three
four	four
five	five
six	six
seven	seven
eight	eight
nine	nine
ten	
eleven	
twelve	
thirteen	
fourteen	
fifteen	
sixteen	
seventeen	
eighteen	
nineteen	
twenty	

thirty	
fourty	
fifty	
sixty	
seventy	
eighty	
ninety	
hundred	
thousand	

D Vergleich der Erkennung mit unterschiedlichen Sprachbefehlssätzen

Tabelle 7.2: Vergleich der Erkennung bei unterschiedlichen Scorern

Wort	**Erkennung Scorer DeepSpeech 0.7.4**	**Erkennung eigener Scorer**
ahead	a head	ahead
altitude	i did too	altitude
back	bake	back
base	base	base
bug	the	bug
checklist	checked lest	checklist
climb	climb	climb
command	command	command
decision	the session	decision
display	this play	display
down	down	down
eight	eight	eight
five	five	five
fly	fly	fly
four	for	four
heading	have	heading
height	it	height
hospital	hospita	hospital
hover	how	hover
hundred	hundred	hundred
left	left	left
mode	more	mode
next	next	next
nine	nine	nine
one	one	one
position	position	position
previous	previous	previous
procedure	to sit	procedure
right	it	right

run	in	run
select	so let	select
set	said	set
seven	seven	seven
six	six	six
speed	speak	speed
start	start	start
stop	stop	stop
target	too	target
thousand	those	thousand
three	three	three
Two	to	two
Up	up	up
waypoint	ray point	waypoint
zero	the	zero
Summe richtig erkannter Wörter:	20	44
Anteil richtig erkannter Wörter:	45,45 %	100 %

E Skript zur Erstellung der Textgrundlage für das Sprachmodell 2

Erstellt von Matthias Bodenstein, DLR.

```
#!/bin/bash


function PrintPhrase()
{
    echo $activator $*
}


function Add0_9 ()
{
    for l in ZERO ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE; do
        PrintPhrase $* $l
    done
}


function Add0_99 ()
{
    #0-9
    Add0_9 $*
    #10-99
    for j in ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE; do
        Add0_9 $* $j
    done
}


function Add0_200 ()
{
```

```
    Add0_99 $*
    for j in ZERO ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE; do
        Add0_9 $* ONE $j
    done
    PrintPhrase $* TWO ZERO ZERO
}

function Add0_360 ()
{
    Add0_99 $*
    for j in ZERO ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE; do
        Add0_9 $* ONE $j
    done
    for j in ZERO ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE; do
        Add0_9 $* TWO $j
    done
    for j in ZERO ONE TWO THREE FOUR FIVE; do
        Add0_9 $* THREE $j
    done;
    PrintPhrase $* THREE SIX ZERO
}
function Add0_9900 ()
{
    Add0_99 $*
    #100-900
    for k in  ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE; do
        PrintPhrase $* $k HUNDRED
    done
    #1000-9900
```

```
    for j in ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE; do
        PrintPhrase $* $j THOUSAND
        for k in  ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE; do
            PrintPhrase $* $j THOUSAND $k HUNDRED
        done
    done
}

model="models/deepspeech-0.7.4-models.pbmm"
testfile=/tmp/test2.wav

if [ "$1" == "" ]; then
    echo "usage: $0 <activator(0|1)>"
    exit 1
fi

if [ "$1" == "1" ]; then
    activator="COMMAND"
else
    activator=""
fi

#process SPEED, ALTITUDE and HEADING command
for h in SPEED ALTITUDE HEADING; do
    for i in SET UP DOWN; do
        Add0_9900 $h $i
    done
done
```

```
#process POSITION command
for h in AHEAD BACK LEFT RIGHT UP DOWN; do
    Add0_99 POSITION $h
done

#process MODE command
PrintPhrase MODE SET HOVER
PrintPhrase MODE SET FLY
Add0_9 Mode SET

#process TARGET command
PrintPhrase TARGET SET HOSPITAL
PrintPhrase TARGET SET BASE
Add0_9 TARGET SET POSITION

#process SET commands
Add0_9900 SET DECISION HEIGHT
Add0_200 SET SPEED BUG
Add0_360 SET HEADING BUG
PrintPhrase SET DISPLAY HOVER
Add0_99 SET DISPLAY

#process WAYPOINT command
Add0_99 WAYPOINT
PrintPhrase WAYPOINT NEXT
PrintPhrase WAYPOINT PREVIOUS

#process RUN command
PrintPhrase RUN START
```

```
PrintPhrase RUN STOP


#process SELECT command

Add0_99 SELECT CHECKLIST

Add0_99 SELECT PROCEDURE
```

F Skript zum Zusammensetzen der Testsamples und Hinzufügen von Rauschen

Erstellt von Matthias Bodenstein, DLR.

```
#!/bin/bash


function TestSample()

{

    res1=`echo $activator $* | sed
's/\.wav//g;s/\(.*\)/\L\1/;s/[_123]*//g'`

    #echo $#: $*

    #echo \"$res1\"

    if [ "$activator" != "" ]; then

        soxfiles="$sample_dir/$activator"

    else

        soxfiles=""

    fi

    while [ $# -gt 0 ]; do

        soxfiles="$soxfiles $sample_dir/$1"

        shift

    done

    sox $soxfiles $testfile pad 0.1

    #play $testfile

    sox -m -v 1.1 $testfile <(sox $testfile -p synth pinknoise vol
$noise_vol ) $testfile2

    #play $testfile2

    if [ "$LM_BETA" == "" ]; then

        res2=`deepspeech --model $model --scorer $scorer --audio
$testfile2 2>/dev/null`

    else
```

```
        res2=`deepspeech --model $model --scorer $scorer --audio
$testfile2 --lm_alpha $LM_ALPHA --lm_beta $LM_BETA 2>/dev/null`

    fi

    rm $testfile

    if [ "$print_mode" == "1" ]; then

        echo "\"$res1\" : \"$res2\""

    fi

    if [ "$res1" == "$res2" ]; then

        hits=`expr $hits + 1`

    else

        if [ "$print_mode" == "2" ]; then

           echo "\"$res1\" : \"$res2\""

        fi

    fi

    samps=`expr $samps + 1`

}


function AddValue()

{

    TestSample $* One.wav Thousand.wav Eight.wav Hundred.wav

    TestSample $* Two.wav Thousand.wav Seven.wav Hundred.wav

    TestSample $* Three.wav Thousand.wav Six.wav Hundred.wav

    TestSample $* Four.wav Thousand.wav Five.wav Hundred.wav

    TestSample $* Five.wav Thousand.wav Four.wav Hundred.wav

    TestSample $* Six.wav Thousand.wav Three.wav Hundred.wav

    TestSample $* Seven.wav Thousand.wav Two.wav Hundred.wav

    TestSample $* Eight.wav Thousand.wav One.wav Hundred.wav

    TestSample $* Nine.wav Thousand.wav Zero.wav Hundred.wav

    for j in One.wav Two.wav Three.wav Four.wav Five.wav Six.wav
Seven.wav Eight.wav Nine.wav; do
```

```
        TestSample $* $j Thousand.wav

        TestSample $* $j Hundred.wav

        TestSample $* $j

    done

    TestSample $* One.wav Eight.wav Two.wav

    TestSample $* Two.wav Seven.wav Three.wav

    TestSample $* Three.wav Six.wav Four.wav

    TestSample $* Four.wav Five.wav Five.wav

    TestSample $* Five.wav Four.wav Six.wav

    TestSample $* Six.wav Three.wav Seven.wav

    TestSample $* Seven.wav Two.wav Eight.wav

    TestSample $* Eight.wav One.wav Nine.wav

    TestSample $* Nine.wav Zero.wav Zero.wav

}


function AddDigit ()

{

    for j in One.wav Two.wav Three.wav Four.wav Five.wav Six.wav
Seven.wav Eight.wav Nine.wav Zero.wav; do

        TestSample $* $j

    done

}


function NextStat ()

{

    echo "Zwischenergebnis: $hits von $samps Treffer"

    sum_hits=`expr $sum_hits + $hits`

    sum_samps=`expr $sum_samps + $samps`

    hits="0"
```

```
    samps="0"

}


model="models/deepspeech-0.7.4-models.pbmm"

testfile=/tmp/test2.wav

testfile2=/tmp/test3.wav


if [ "$1" == "" ]; then

    echo "usage: $0 <single-sample-dir> [scorer] [mode(0|1|2)]
[activator(0|1)] [noise volume] [LM_ALPHA LM_BETA]"

    exit 1

fi


if [ "$2" != "" ]; then

    scorer=$2

else

    scorer="scorer/deepspeech-0.7.4-models.scorer"

fi


if [ "$4" == "1" ]; then

    activator="Command.wav"

else

    activator=""

fi


if [ "$5" == "" ]; then

    noise_vol="1"

else

    noise_vol=$5
```

```
fi


if [ "$7" != "" ]; then
    LM_ALPHA="$5"
    LM_BETA="$6"
fi


echo used scorer: $scorer LM_ALPHA=$LM_ALPHA LM_BETA=$LM_BETA
hits="0"
sum_hits="0"
samps="0"
sum_samps="0"
sample_dir=$1
print_mode=$3

#process SPEED, ALTITUDE and HEADING command
for h in Speed.wav Altitude.wav Heading.wav; do
    for i in Set.wav Up.wav Down.wav; do
        AddValue $h $i
    done
    NextStat
done

#process POSITION command
for h in Ahead.wav Back.wav Left.wav Right.wav Up.wav Down.wav; do
    AddValue Position.wav $h
done
NextStat
```

```
#process MODE command

TestSample Mode.wav Set.wav Hover.wav

TestSample Mode.wav Set.wav Fly.wav

AddDigit Mode.wav Set.wav

NextStat


#process TARGET command

TestSample Target.wav Set.wav Hospital.wav

TestSample Target.wav Set.wav Base.wav

AddDigit Target.wav Set.wav Position.wav

NextStat


#process SET commands

AddValue Set.wav Decision.wav Height.wav

AddValue Set.wav Speed.wav Bug.wav

AddValue Set.wav Heading.wav Bug.wav

TestSample Set.wav Display.wav Hover.wav

AddValue Set.wav Display.wav

NextStat


#process WAYPOINT command

AddValue Waypoint.wav

TestSample Waypoint.wav Next.wav

TestSample Waypoint.wav Previous.wav

NextStat


#process RUN command

TestSample Run.wav Start.wav

TestSample Run.wav Stop.wav
```

```
NextStat


#process SELECT command

AddValue Select.wav Checklist.wav

AddValue Select.wav Procedure.wav

NextStat


echo "Gesamtresultat: $sum_hits von $sum_samps Treffer"
```

G Geräte- und Softwareliste

Tabelle 7.3: Geräteliste

Nr.	Gerät / Verwendung	Bezeichnung	Hersteller
1	Mikrofon im Forschungshubschrauber ACT / FHS		
2	Headset für Sprachbefehlssatz a	Ear Force Call of Duty: Ghosts	Turtle Beach
3	Eingebautes Mikrofon Tablet für Sprachbefehlssatz b	Galaxy Tab A 9.7 SM-T550N	Samsung

Tabelle 7.4: Softwareliste

Nr.	Software	Version	Herausgeber
1	VirtualBox graphische Benutzeroberfläche	6.1.14	Oracle Corporation
2	DeepSpeech	0.7.4	Mozilla Corporation
3	Audacity	2.4.1	Audacity-Team
4	WaveEditor for Andorid™	1.88	Sound-Base Audio, LLC
5	SoX	14.4.2	Chris Bagwell u. a.
6	Python	3.6.7	Python Software Foundation
7	KenLM	1	Kenneth Heafield