Technische Hochschule Wildau
Technical University of Applied Sciences

Masterarbeit

zur Erlangung des akademischen Grades
Master

Technische Hochschule Wildau

Fachbereich Ingenieur- und Naturwissenschaften

Studiengang Telematik (M. Eng.)

Thema (deutsch): Konzeption und Implementierung eines Künstlichen Neuronalen Netzes zur Situationserkennung mit humanoiden Robotern anhand mehrerer Eingangskanäle

Thema (englisch): Conception and implementation of an artificial neural network for situation recognition with humanoid robots based on several input channels

Autor/in:	Lara Sofie Ziemert
Seminargruppe:	TM/19
Betreuer/in:	Prof. Dr. rer. nat. Janett Mohnke
Zweitgutachter/in:	Prof.Dipl.-Inf. Birgit Wilkes
Spätestmögliche Abgabe:	29.01.2022
Eingereicht am:	

Bibliographische Beschreibung

Ziemert, Lara S.

Konzeption und Implementierung eines Künstlichen Neuronalen Netzes zur Situationserkennung mit humanoiden Robotern anhand mehrerer Eingangskanäle

Masterarbeit, Technische Hochschule Wildau 2022, 142 Seiten, 27 Abbildungen, 7 Tabellen, 4 Anlagen

Inhalt

In dieser Masterarbeit soll eine Architektur für ein Künstliches Neuronales Netz erarbeitet werden, mit der mehrere Eingangskanäle eines humanoiden Roboters für eine Situationserkennung zusammengefasst werden können. Dazu werden zunächst die Grundbausteine für eine Situationserkennung am Beispiel des menschlichen Gehirns untersucht und die Erkenntnisse auf die Arbeitsweise Künstlicher Neuronales Netze übertragen.

Anschließend wird ein Konzept für eine Situationserkennung auf humanoiden Robotern erstellt. Dazu werden mögliche Situationen konstruiert und die Aufgabe der Situationserkennung so formuliert, dass sie mit einem Künstlichen Neuronalen Netz bearbeitet werden kann. Daraufhin werden mögliche Architekturen für dieses Netz verglichen und die am besten geeignetste ausgewählt.

Das erstellte Konzept soll im Rahmen dieser Arbeit auf dem Tisch-Roboter ROS-E umgesetzt werden. Dazu soll eine Trainings- und Testumgebung geplant und umgesetzt werden. Nachdem das entwickelte Künstliche Neuronale Netz trainiert und getestet wurde, wird ausgewertet, ob die ausgewählte Architektur für eine Situationserkennung auf humanoiden Robotern geeignet ist.

Abstract

In this master's thesis, an architecture for an artificial neural network is to be developed, with which several input channels of a humanoid robot can be combined for situation recognition. For this purpose, the basic building blocks for situation recognition are examined using the example of the human brain and the findings are transferred to methods of artificial neural networks.

A concept for situation recognition on humanoid robots is then created. For this purpose, possible situations are constructed and the task of recognizing the situation is formulated in such a way that it

can be processed with an artificial neural network. Possible architectures for this network are then compared and the most suitable is selected.

The created concept is to be implemented as part of this work on the robot ROS-E. A training and test environment is to be planned and implemented for this purpose. After the developed artificial neural network has been trained and tested, it is evaluated whether the selected architecture is suitable for situation recognition on humanoid robots.

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt und motiviert haben.

Mein besonderer Dank gilt daher zunächst meinen Betreuern der Abschlussarbeit, Prof. Dr. rer. nat. Janett Mohnke und Prof. Dipl.-Inf. Birgit Wilkes, die es ermöglicht haben, dass ich dieses spannende Thema bearbeiten konnte und mich dabei unterstützt haben, das endgültige Ziel und dessen Nutzen nicht aus den Augen zu verlieren.

Außerdem möchte ich mich bei Valentin Schröter bedanken, ohne den es ROS-E nicht geben würde und der mich beim Lernen der grundlegenden Programmierung von ROS-E unterstützt und so manche zusätzliche Funktion nach meinen Wünschen eingebaut hat.

Meinen Eltern möchte ich dafür danken, dass sie versucht haben, meine komplexen Erklärungsversuche nachzuvollziehen und das Korrekturlesen meiner Arbeit unterstützt haben.

Storkow, den 15. Januar 2022

Lara Ziemert

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Abschlussarbeit selbstständig und ohne unerlaubte fremde Hilfe angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und die den verwendeten Quellen und Hilfsmitteln wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Storkow, 23.01.2022 — L. Zimert

Ort, Datum — Unterschrift

Inhaltsverzeichnis

Hinweise zum Lesen der Arbeit

Nachfolgend wird die in dieser Arbeit verwendete formale Ausdrucksweise beschrieben, um das Lesen der Arbeit zu erleichtern.

Die Quellenangaben werden in eckigen Klammern am Ende des jeweiligen Abschnitts hinterlegt. Die Quellen werden aufeinanderfolgend nummeriert und sind unter dieser Nummer im Quellenverzeichnis zu finden. Die Quellenangaben für Bilder sind hinter der jeweiligen Bildunterschrift zu finden.

Wörtliche Zitate werden in Anführungszeichen gesetzt. Auslassungen werden dabei mit drei Punkten in eckigen Klammern […] gekennzeichnet.

Fachbegriffe werden durch kursive Schrift hervorgehoben. Sie werden im Glossar hinterlegt und eine kurze Beschreibung hinzugefügt.

Abkürzungen von Fachbegriffen werden im Abkürzungsverzeichnis aufgelistet und dort ausgeschrieben.

Der Quellcode und eine kurze Beschreibung dazu sind in der beigelegten CD zu finden.

1 Einleitung

Andrea hört gerade Musik, als sie einen Anruf bekommt.

Elfriede ist gestolpert und hat Schwierigkeiten, wieder aufzustehen.

Thore spielt mit seinen Action-Figuren, aber es fehlt die spannende Hintergrundmusik.

1.1 Ausgangslage

Die Inspiration für diese Arbeit stammt aus der Masterarbeit "Konzeption und prototypische Implementierung eines hybriden Algorithmus zur Emotionserkennung mit humanoiden Robotern" von Tina Lüthe. Die darin durchgeführte Untersuchung sollte zeigen, ob die Ergebnisse vorhandener Dienste zur Emotionserkennung durch die Kombination dieser Ergebnisse verbessert werden könnten. Dazu wurden die Dienste von IBM Watson und Microsoft Azure mit jeweils einem Eingangskanal (Bild und Text) zusammengeführt. Es wurde ein Algorithmus entwickelt, der die Ergebnisse der einzelnen Dienste gemeinsam auswertet. Der Ausblick der Arbeit für die Weiterentwicklung des Themas schlägt die Nutzung von anderen Eingangskanälen wie Ton oder Video und zusätzlichen Komponenten wie Alter und Geschlecht vor. Im Fazit der Arbeit wurde vermutet, dass die Umsetzung dieses Problems mit einem eigenen Künstlichen Neuronalen Netz genauere Ergebnisse erzielen würde. [1]

Eine wichtige Frage bei der Umsetzung ist, welche Emotionen erkannt werden sollten. Der Dienst von Microsoft Azure kann zum Beispiel aus einem Bild die Emotionen Ärger, Verachtung, Ekel, Furcht, Freude, Neutral, Traurigkeit und Überraschung erkennen. Der Dienst von IBM Watson analysiert den Tonfall in einem Text und kann dabei Ärger, Furcht, Freude, Traurigkeit, analytisch, zuversichtlich und zögernd erkennen. [2] [3]

Wie nützlich wären diese „Grundemotionen“ für die Kommunikation zwischen Roboter und Mensch?

Die Emotion zu „erkennen“ ist nur der halbe Schritt. Die andere Hälfte besteht darin, die interpretierte Emotion auch in einer Anwendung zu nutzen. Damit die „Emotionserkennung“ überhaupt einen Nutzen hat, müsste für jede Emotion eine andere Entscheidung in der Interaktion mit dem Nutzer getroffen werden können. Für das folgende Beispiel wird angenommen, dass die Emotion „Ärger“ für eine Person interpretiert werden konnte. Schon an dieser Stelle fällt auf, dass es ohne weiteren Kontext sehr viele mögliche Gründe für diese Emotion geben könnte. Ist die Person mit dem Roboter selbst unzufrieden oder mit einem vom Roboter unabhängigen Ereignis? Die Anwendung, die die Auswertung „Ärger“ verwendet, hat zusätzlich die Information, in welcher Ansicht der App sich die Person aktuell befindet. In diesem Beispiel zeigt die Anwendung eine Liste mit Kacheln, die jeweils ein Icon und ein Wort beinhalten. Ist die Person nicht sicher, was die Icons bedeuten? Ist

die Schrift zu klein? Ist die gewünschte Auswahl nicht verfügbar? Weiß die Person nicht mehr was passiert, wenn sie eine Kachel antippt? Weiß die Person nicht, dass sie eine Kachel antippen kann, um diese zu markieren? Bei dieser Überlegung wird auch deutlich, dass Emotionen nicht eindeutig „erkannt“ werden können, sondern immer von anderen Gehirnen interpretiert werden. Auch ein anderer Mensch kann sich nie sicher sein, ob seine Interpretation von Emotionen seines Gegenübers richtig war.

Damit eine Anwendung eine solche Interpretation der aktuellen Emotion sinnvoll verwenden kann, müsste sie mit der Interpretation eine Entscheidung treffen können, welche Reaktion angemessen ist. Schon an diesem ersten Beispiel wird deutlich, dass die verschiedenen Reaktionen, die nötig wären, um alle Nutzer mit „Ärger“ optimal zu unterstützen, sich nicht mit Hilfe von einer beschränkten Anzahl von 5 oder 6 „Grundemotionen“ auswerten lassen.

An diesem Beispiel kann auch gut gezeigt werden, dass Emotionen nicht die ausschlaggebenden Merkmale für eine Reaktion sind. Dieselben Fragen könnten auftreten, wenn der Nutzer als „zögernd“ erkannt wurde. Emotionen sind ein sehr komplexes Thema und nicht jeder Mensch zeigt in einer Situation dieselbe Emotion. Um mit einer Anwendung auf die Emotionen eines Nutzers reagieren zu können ist es denkbar, statt der Emotionen die Ziele des Nutzers zu interpretieren. Ein solches Ziel könnte sein, die Schrift vergrößern zu wollen, einen Hinweis zu bekommen, was ein bestimmter Button oder Sprachbefehl auslöst oder den aktuellen Vorgang abzubrechen. Daher soll in dieser Arbeit nicht die Emotion des Nutzers interpretiert werden, sondern die Gesamtsituation. Es soll untersucht werden, ob der Ansatz einer Situationserkennung für den Einsatz mit einem humanoiden Roboter einsatzfähige Ergebnisse erzielen kann.

Um die aktuelle Situation interpretieren zu können, soll ein eigenes *Künstliches Neuronales Netz* (KNN) entwickelt werden, dessen Ergebnisse von humanoiden Robotern oder mit ihnen verknüpften Anwendungen verwendet werden. Das Ziel dabei ist es, aufgrund der aktuellen Situation eine angepasste Reaktion festlegen zu können. Wenn der Nutzer vor der App mit dem bestimmten Gesichtsausdruck sitzt und eine Weile keine Aktion mehr ausgeführt hat, kann der Roboter nachfragen, ob der Nutzer ein Problem hat und wie er helfen könnte. Wenn die Schrift zu klein ist, könnte diese vom Roboter (oder der Anwendung) angepasst werden. Weiß der Nutzer nicht, was die Icons bedeuten, könnte der Roboter eine Beschreibung vorlesen. Wenn ein Nutzer lächelt, könnte die Form des Mundes oder der Augen des Roboters ebenfalls angepasst werden.

1.2 Motivation

Viele Anwendungen humanoider Roboter werden entwickelt, um die Menschen bei ihren alltäglichen Aufgaben zu unterstützen. Eine solche Aufgabe könnte zum Beispiel sein, sich sportlich zu betätigen, einen Termin einzuhalten, eine bestimmte Therapie-Methode anzuwenden oder etwas Neues zu lernen. Dabei soll die Unterstützung so einfach wie möglich zugänglich sein. Es gibt jedoch meist keine Lösung, die von den sehr verschiedenen Nutzergruppen gleichsam als „einfach“ empfunden wird. Humanoide Roboter können für die Arbeit mit verschiedenen Altersgruppen, Kulturen oder auch gesundheitlichen und geistigen Einschränkungen eingesetzt werden. Was macht den Umgang mit humanoiden Robotern für alle Menschen einfach?

Im Rahmen der Arbeit wird eine mögliche Antwort auf diese Frage angenommen: Wenn sich Menschen bei der Kommunikation mit einem Roboter so verhalten, wie sie es schon von menschlicher Kommunikation gewöhnt sind, wird die Kommunikation einfacher. Die Menschen müssten so nicht noch eine zusätzliche Art der Kommunikation verstehen lernen.

Ein Roboter müsste also in der Lage sein, die verschiedenen Eigenheiten in der Kommunikation eines Menschen zu lernen. Derzeit kommunizieren Menschen mit den meisten Anwendungen über Touch-Displays und eindeutige Eingabemöglichkeiten. Für eine Anwendung bedeutet dies, dass keine Möglichkeit besteht, die Mimik, Gestik, den Tonfall oder die Stimmung seines Kommunikationspartners zu berücksichtigen. Die Menschen nutzen diese Kommunikationskanäle jedoch ganz automatisch und meist unterbewusst.

Ein kleiner Schritt zur Anpassung von Anwendungen an menschliche Kommunikation sind Spracheingaben. Die meisten Menschen kommunizieren häufig und mühelos über gesprochene Worte. Die Worte erst in ein Gerät eintippen zu müssen, dauert meist länger und erscheint für manche Nutzergruppen umständlicher. Doch selbst wenn der Mensch über die Spracheingabe-Funktion mit der Anwendung kommuniziert, bekommt diese nur den Text, der aus der Audiospur umgewandelt wurde. Bei dieser Kommunikation fehlen der Anwendung trotzdem die zusätzlichen Informationen aus der Audiospur, wie Tonfall und Lautstärke. Darüber hinaus kann ein Mensch nicht nur Sprache, sondern auch Tonfall, Gesichtsausdruck, Körpersprache, Handlungen und andere Mittel zur Kommunikation verwenden. Auch die Interpretation dieser Ausdrucksweisen als Emotionen sind ein wichtiges Mittel menschlicher Kommunikation. In bestimmten Fällen ist das Zeigen eines Gesichtsausdrucks effektiver, als den Inhalt der gewünschten Aussage in Worte zu fassen. Anstatt wiederholt aussprechen zu müssen „Ich bin glücklich.“, kann ein Mensch einfach lächeln.

Der Mensch verarbeitet zur Kommunikation alle Sinneseindrücke zu einem Gesamtkonzept, das als Situation bezeichnet werden kann. Es werden nicht nur der Inhalt der Sprache, sondern auch Tonfall,

Körpersprache, Gesichtsausdruck, die Umgebung und das eigene Körperempfinden in die Interpretation einer Situation aufgenommen. Diese vielen Informationsquellen sind nötig, um die nahezu unzählbare Anzahl an Situationen berücksichtigen zu können, denen ein Mensch begegnen kann.

Auch eine Anwendung benötigt ausreichend viele Informationen aus der Umwelt, um alle gewünschten Situationen erkennen zu können. Der Gesichtsausdruck des Nutzers allein ist nicht ausreichend. Es könnten der Tonfall, die Lautstärke, der Inhalt der Sprache, die Anzahl der anwesenden Personen, andere Gegenstände, der Raum, die Tageszeit, die Aufmerksamkeit der Nutzer, die aktive Anwendung und weitere Bedingungen genutzt werden, um die aktuelle Situation zu interpretieren. [4] [5]

Eine Situation zu erkennen, scheint für einen Roboter nahezu unmöglich. Eine Situation kann ein komplexes Konstrukt aus Handlungen und Zielen verschiedener Beteiligter sein. Es gibt jedoch bereits Ansätze, in denen versucht wird, Situationen in Bildern mit Machine-Learning-Techniken zu beschreiben. Dabei bekommt zum Beispiel ein Künstliches Neuronales Netz die Aufgabe, eine Beschreibung für ein Bild zu generieren. Es können dabei Orte, Objekte und sogar Handlungen berücksichtigt werden. [6]

Diese Arbeit soll eine mögliche Grundlage dafür schaffen, dass mehr Anwendungen unter der Berücksichtigung der aktuellen Situation für humanoide Roboter entwickelt werden können. Diese Anwendungen könnten zum Beispiel ihre Funktionen danach anpassen, ob ein Mensch gerade aufmerksam oder abgelenkt ist. Sie könnten zusätzliche Hinweise anzeigen, wenn der Nutzer überfordert ist oder auch andere Funktionen anbieten, sobald mehrere Personen anwesend sind.

1.3 Zielstellung

Im Rahmen dieser Arbeit soll ein Künstliches Neuronales Netz entwickelt werden, das trainiert wird, verschiedene Situationen zu unterscheiden. Dazu sollen geeignete Informationsquellen bzw. Eingangskanäle aus der Umwelt identifiziert werden. Das Künstliche Neuronale Netz soll ebenso wie der Mensch multimodal lernen, also verschiedene Typen von Eingangsdaten zusammenfassen. Die vom Netz ermittelte Situation soll anschließend an andere Anwendungen weitergeleitet werden, sodass diese je nach Situation auf den Nutzer eingehen können.

Das Künstliche Neuronale Netz soll später gemeinsam mit dem Tisch-Roboter ROS-E eingesetzt werden. Dieser besitzt verschiedene Sensoren wie eine Kamera und ein Mikrofonarray, über die einzelne Bilder, Videos und Ton als Eingangskanäle verwendet werden können. Die Audio-Daten könnten über eine Speech-to-Text-Funktion auch als Text-Daten verwendet werden. Das Ziel dieser Arbeit ist es, eine Auswahl der Eingangskanäle und der möglichen Ausgaben zu treffen. Die Ergeb-

nisse des Netzes sollen auch unabhängig von einer konkreten Anwendung von ROS-E verwendet werden können. Dazu soll ein prototypisches Test-Modul entwickelt werden, was die Ergebnisse des Netzes auswertet und zum Beispiel den Gesichtsausdruck (Augen- und Mundform) von ROS-E anpasst.

Diese Arbeit richtet sich an Entwickler, die eine Situationserkennung bereitstellen wollen und an Entwickler, die die Ergebnisse der Situationserkennung bei der Implementierung ihrer Anwendungen auf einem humanoiden Roboter verwenden möchten.

1.4 Methoden und Aufbau der Arbeit

Zu Beginn soll der Leitfaden dieser Arbeit anhand der Forschungsfragen vorgestellt werden (siehe Kapitel 2). Um einen leichten Einstieg in das Thema zu ermöglichen, sollen in Kapitel 3 zunächst die grundlegenden Begriffe und Themengebiete definiert werden, die im Rahmen dieser Arbeit behandelt werden. Dazu zählt zunächst die Definition eines humanoiden Roboters (siehe Abschnitt 3.1). Anschließend wird der Begriff der Situation und dessen Bedeutung in dieser Arbeit definiert (siehe Abschnitt 3.2). In Vorbereitung auf die Kapitel, die sich im Detail mit Künstlichen Neuronalen Netzen beschäftigen, werden die Begriffe der Künstlichen Intelligenz, des Machine Learning und der Künstlichen Neuronalen Netze selbst definiert (siehe Abschnitt 3.3).

In Kapitel 4 werden die Grundbausteine einer Situationserkennung vorgestellt. Dazu werden mehrere Verarbeitungsschritte am Vorbild des menschlichen Gehirns identifiziert und diese auf die Arbeitsweise von Künstlichen Neuronalen Netzen übertragen. Im ersten Schritt wird die Wahrnehmung der Umwelt untersucht, also die Sinne des Menschen und die möglichen Eingangsdaten eines Künstlichen Neuronalen Netzes (siehe Abschnitt 4.1). Anschließend wird untersucht, wie die Sinne beim Menschen zu einer Gesamtwahrnehmung verknüpft werden und wie multimodales Lernen funktioniert (siehe Abschnitt 4.2). Nachdem die Verknüpfung der Sinneseindrücke und Eingangsdaten beschrieben wurde, soll die interne Repräsentation einer Situation und die Priorisierung von Sinneseindrücken bzw. Eingangsdaten näher untersucht werden (siehe Abschnitt 4.3).

Nachdem die Grundbausteine einer Situationserkennung vorgestellt wurden, soll in Kapitel 5 das Konzept für die im Rahmen dieser Arbeit erstellten Situationserkennung genauer beschreiben werden. Dazu wird zunächst der Tisch-Roboter ROS-E vorgestellt, auf dem die Situationserkennung getestet werden soll (siehe Abschnitt 5.1). Um festlegen zu können, welche Situationen von einem Roboter erkannt werden sollten, wird eine Sammlung von Beispiel-Situationen erstellt. In Vorbereitung auf diese Sammlung wurden Personae erstellt, anhand denen die Beispiele beschrieben

werden (siehe Abschnitt 5.2). Die Sammlung möglicher Situationen soll zudem bei der Beantwortung der Frage helfen, welche Ergebnisse die Situationserkennung später liefern soll (siehe Abschnitt 5.3). Auch die Beschreibung möglicher Reaktionen eines Roboters auf diese erkannten Situationen soll dabei unterstützen (siehe Abschnitt 5.4). Mit Hilfe dieser Grundlage soll die Aufgabe der Situationserkennung so definiert werden, dass sie mit Hilfe eines Künstlichen Neuronalen Netzes bearbeitet werden kann. Dazu werden verschiedene Möglichkeiten der Formulierung der Aufgabe betrachtet und anschließend eine dieser Möglichkeiten ausgewählt (siehe Abschnitt 5.5).

Mit der Formulierung der Aufgabe der Situationserkennung aus Kapitel 5 und den Grundbausteinen einer Situationserkennung aus Kapitel 4 soll in Kapitel 6 die Gesamtarchitektur des umzusetzenden Künstlichen Neuronalen Netzes ausgewählt werden. Dazu werden zunächst die Kriterien für die Auswahl einer dieser Architekturen definiert (siehe Abschnitt 6.1). Anhand dieser Kriterien werden drei mögliche Architekturen verglichen (siehe Abschnitt 6.2) und anschließend eine Architektur gewählt (siehe Abschnitt 6.3). Nachfolgend wird das Netz beschrieben, mit dem die Situationserkennung auf ROS-E getestet werden soll (siehe Abschnitt 6.4). Bevor die ausgewählte Architektur umgesetzt wird, sollen in Kapitel 7 die Vorbereitungen für das Training beschrieben werden. Dabei werden zunächst die einzelnen Trainingsphasen beschrieben und mögliche Trainingsmethoden für die Architektur diskutiert, um zu ermitteln welche Trainingsdaten jeweils benötigt werden (siehe Abschnitt 7.1). Anschließend wird ein Plan erstellt, welche Trainingsdaten beschafft werden müssen (siehe Abschnitt 7.2) und diese anschließend beschafft (siehe Abschnitt 7.3). Nach der Beschaffung werden die verwendeten Methoden der Datenaufbereitung beschrieben (siehe Abschnitt 7.4).

In Kapitel 8 werden alle Komponenten identifiziert und beschrieben, die für eine Trainings- und Testumgebung benötigt werden (siehe Abschnitt 8.1). Anschließend wird das Konzept derjenigen Komponenten erstellt, die im Rahmen dieser Arbeit umgesetzt werden können (siehe Abschnitt 8.2). Die Umsetzung dieser Trainings- und Testumgebung wird in Kapitel 9 beschrieben. Dazu zählen zunächst die Konfiguration und das Training des Künstlichen Neuronalen Netzes für die Situationserkennung (siehe Abschnitt 9.1). Als nächstes wird die Umsetzung eines zweiten Netzes beschrieben, das die Ergebnisse der Situationserkennung verwendet, um Aktionen auf ROS-E, wie das Ändern des Gesichtsausdruckes passend zur aktuellen Situation, auszuführen (siehe Abschnitt 9.2). Diese beiden Teilnetze werden anschießend zur Trainings- und Testumgebung auf ROS-E zusammengeführt (siehe Abschnitt 9.3).

Im Fazit werden die Erkenntnisse dieser Arbeit zusammengefasst, indem zunächst die Forschungsfragen kurz beantwortet werden (siehe Abschnitt 10.1). Anschließend soll die Eignung des Gesamtkonzeptes für die Umsetzung einer Situationserkennung auf humanoiden Robotern eingeschätzt

werden (siehe Abschnitt 10.2). Zum Abschluss sollen offene Aufgaben und Probleme angesprochen werden, um Ideen aufzuzeigen, die in Zukunft umgesetzt werden können, um die Situations-erkennung weiterzuentwickeln (siehe Abschnitt 10.3).

2 Forschungsfragen

Die Forschungsfragen bilden den Leitfaden dieser Arbeit. Anhand der Beantwortung dieser Fragen soll das Thema untersucht und die Ergebnisse festgehalten werden. So können die Erkenntnisse am Ende der Arbeit zusammenfassend ausgewertet werden.

Im Folgenden werden die Forschungsfragen genannt und genauer beschrieben:

1. Welche Situationen treten im Umgang mit dem Tisch-Roboter ROS-E auf?

   Um zu ermitteln, wie die Aufgabe der Situationserkennung für ein Künstliches Neuronales Netz formuliert werden kann, wird zunächst eine Sammlung möglicher Situationen im Umgang mit dem Tisch-Roboter ROS-E angelegt. Auf diese Weise sollen die Ziele der Benutzer und Entwickler bei der Nutzung der Situationserkennung in die Formulierung der Aufgabe einfließen.

2. Kann mit Hilfe von Künstlichen Neuronalen Netzen eine Situationserkennung so entwickelt werden, dass die zu erkennenden Situationen nicht fest definiert, sondern gelernt werden können?

   Selbst, wenn eine sehr umfangreiche Liste möglicher Situationen aufgestellt wird, die von der Situationserkennung erkannt werden sollen, ist es unmöglich, alle Situationen abzubilden, die auftreten könnten. Zudem könnten sich die Situationen, die ein Entwickler vorhersieht, sehr stark von denen unterscheiden, die später im Einsatz auftreten. Daher soll untersucht werden, ob Künstliche Neuronale Netze es erlauben, die möglichen Situationen nicht vorgeben zu müssen, sondern durch das Netz selbst lernen zu lassen.

3. Welche Eingangskanäle können in einem Künstlichen Neuronalen Netz für die Situationserkennung kombiniert werden?

   Für die Beantwortung dieser Frage soll besonders die Arbeitsweise des menschlichen Gehirns als Grundlage dienen und recherchiert werden. Anschließend wird der aktuelle Forschungsstand der Umsetzung dieser Mechanismen (z.B. multimodales Lernen) mit Hilfe von Künstlichen Neuronalen Netzen dargestellt. Als letzter Schritt werden die möglichen Eingangskanäle des Tisch-Roboters ROS-E mit den aktuell in Künstlichen Neuronalen Netzen verwendbaren Daten abgeglichen.

4. Auf welche Weise können Situationen in einem Künstlichen Neuronalen Netz abgebildet werden?

   Anhand dieser Frage soll untersucht werden, auf welche Weise die Eingabedaten der verschiedenen Kanäle (Video, Audio, Text, …) in einem Künstlichen Neuronalen Netz in eine komprimierte Zusammenfassung umgewandelt werden und wie mit dieser Abstraktion anschließend verschiedene Problemstellungen vom Netz bearbeitet werden können.

5. Wie können die Ergebnisse der Situationserkennung für die Nutzung in anderen Anwendungen bereitgestellt werden?

   Nachdem das Künstliche Neuronale Netz entwickelt und trainiert wurde, müssen die Ergebnisse auf dem Tisch-Roboter ROS-E und von anderen Anwendungen (z.B. zusätzlichen Apps) erreichbar sein. Die möglichen Architekturen der Bereitstellung sollen diskutiert und eine prototypisch umgesetzt werden.

Anhand dieser Fragen sollen die nötigen Erkenntnisse gewonnen werden, um das Künstliche Neuronale Netz zu entwickeln und zu trainieren. Eine Trainings- und Testumgebung soll prototypisch umgesetzt werden, um damit die Ergebnisse einzuschätzen und eine Weiterentwicklung im Anschluss an die Arbeit zu ermöglichen.

3 Grundlagen

In diesem Kapitel wird zunächst das grundlegende Wissen festgehalten, das für das weitere Verständnis der Arbeit notwendig ist. Dazu zählen die Definitionen für humanoide Roboter und Situationen. Die drei grundlegenden, häufig synonym verwendeten Begriffe *Künstliche Intelligenz*, *Machine Learning* und *Künstliche Neuronale Netze* werden voneinander abgegrenzt.

3.1 Humanoide Roboter und deren Einsatzgebiete

Zum besseren Verständnis wird als erstes die Bedeutung der Bezeichnung „humanoider Roboter“ im Rahmen dieser Arbeit geklärt.

Ein Roboter wird traditionell als humanoid bezeichnet, wenn seine Gestalt der menschlichen Gestalt ähnelt. Laut dieser Definition besitzen humanoide Roboter zwei Arme, zwei Beine einen Kopf mit zwei Augen und einen Körper. Dabei sind auch die Gelenke an ähnlichen Positionen wie beim Menschen. Der humanoide Roboter ist somit auch in seinen Bewegungsabläufen dem Menschen möglichst ähnlich. Da der menschliche Körper und dessen Bewegung sehr komplex sind, gibt es in der Praxis jedoch unterschiedliche Grade der Ähnlichkeit. Es werden auch Roboter als humanoid bezeichnet, deren Oberkörper dem Menschen nachempfunden ist, die aber auf Rollen fahren.

Die Gestalt ist allerdings nur eine der Eigenschaften, die dem Menschen nachempfunden werden kann. Auch die Sensorik kann einen Roboter humanoid machen. Es gibt eine Vielzahl an Sensoren, die bei Robotern eingesetzt werden können. Einige Beispiele sind Touch-Sensoren, zwei Kameras mit denen die Stereosicht des Menschen nachgebildet werden kann und Mikrofon-Arrays mit denen die Richtung der Tonquelle bestimmt werden kann. Die Sensorik definiert die Möglichkeiten der Kommunikation mit dem Menschen. Ohne eine Kamera könnte der Roboter zum Beispiel keine Mimik oder Gestik berücksichtigen.

Eine weitere Eigenschaft, in der Roboter dem Menschen ähneln können, ist die Intelligenz. Dabei kann diese Intelligenz auf verschiedene Weise definiert werden. Eine Möglichkeit wäre der Grad an Autonomie, den ein Roboter besitzt. Ist er in der Lage, sich selbst durch die Welt zu bewegen oder wird er ferngesteuert? Intelligenz könnte sich auch auf die Fähigkeit zu Lernen beziehen oder die Fähigkeit, selbstständig Probleme zu lösen. Eine Situationserkennung könnte die Grundlage dafür bieten, dass ein Roboter diese Autonomie erreicht oder auch selbstständig lernt, welche Aktionen in welcher Situation zur Lösung eines Problems führen. Beispiele für autonome Entscheidungen von humanoiden Robotern könnten sein, ein Gespräch zu starten oder eine aufgebrachte Person zu beruhigen. Dabei muss Autonomie nicht bedeuten, dass der Roboter alle Funktionen ohne ein Nach-

fragen beim Nutzer ausführt. Es soll aber ermöglicht werden, dass der Roboter selbstständig Vorschläge macht, die anschließend vom Nutzer angenommen oder abgelehnt werden können. So wäre es auch möglich, diese Entscheidung für eine bestimmte Funktion vorzugeben, sodass der Nutzer wählen kann, welche Funktionen selbstständig ausgeführt werden dürfen. [6] [7]

Im Rahmen dieser Arbeit soll die Situationserkennung mit Hilfe eines humanoiden Roboters prototypisch umgesetzt werden. Dazu wird der Tisch-Roboter ROS-E verwendet, der an der TH Wildau entwickelt wird.

Das Aussehen von ROS-E wurde an einen menschlichen Kopf angelehnt. Sie besitzt zwei Displays, die die Augen darstellen und ein Display, das den Mund darstellt (siehe Abbildung 1).

Abbildung 1: Der Tisch-Roboter ROS-E (Quelle: eigene Abbildung)

Humanoide Roboter können bereits an einer Vielzahl von Orten und für verschiedenste Aufgaben verwendet werden. Zur Unterhaltung könnten Spiele wie „Schiffe versenken“, „4 gewinnt“ oder „Schere, Stein, Papier“ angeboten werden. Diese könnten auch über eine zusätzliche App visualisiert werden. Neben der Unterhaltung durch Spiele und Musik können vor allem für Kinder viele Anwendungen für die Unterstützung beim Lernen entstehen. Beispiele dafür wären eine Anwendung, die Kinder (oder auch Erwachsene) beim Lesen lernen unterstützt oder das Einmaleins abfragt. In den letzten Jahren wurden auch positive Ergebnisse durch den Einsatz humanoider Roboter zur Unterstützung des Lernens von Kindern mit Autismus erzielt. Dieser Anwendungsbereich eröffnet die Möglichkeit für eine Vielzahl verschiedener Anwendungen, um den Betroffenen zu helfen und diese Krankheit besser zu verstehen. [7]

Ein weiteres Anwendungsgebiet ist die Unterstützung älterer Menschen in ihrem Alltag. Dabei könnte ROS-E (und andere humanoide Roboter) vor allem bei der Einhaltung der täglichen Routine und wichtiger Termine unterstützen. Die Einhaltung eines Schlafrhythmus, einer ausgewogenen Er-

nährung und möglichst viel Bewegung kann nicht nur bei älteren Menschen die Wahrscheinlichkeit für den Ausbruch einer Krankheit reduzieren. ROS-E könnte daran erinnern, vor dem Schlafen gehen das Licht zu dimmen und keine aufregenden Tätigkeiten auszuüben oder sogar durch die Verknüpfung mit einem Fitnessarmband die Qualität des Schlafes für den Menschen auswerten. Für die Ernährung könnte es zum Beispiel ein interaktives Kochbuch geben, das Rezepte je nach körperlichem Zustand vorschlägt. Diese können dann gemeinsam mit ROS-E zubereitet werden.

Die Einhaltung einer täglichen Routine kann vor allem bei Krankheiten wie Demenz oder Depressionen hilfreich sein. ROS-E könnte mit Hilfe von eingestellten Erinnerungen an die Zubereitung von Mahlzeiten, das Einkaufen, die Einnahme von Medikamenten und weitere tägliche Aufgaben erinnern. Von Demenz betroffenen Personen könnte sie zum Beispiel bei Therapiemethoden wie Kognitivem Training (Gedächtnisübungen), Autobiografische Arbeit (Erzählen von Erinnerungen), Realitätsorientierung (Uhrzeit, Tag und eigenen Standort bestimmen) oder Musiktherapie unterstützen, wenn dies mit dem Therapeuten vereinbart wird. [8] [9]

In einem Haushalt kann eine Anruf-Funktion, bei der ROS-E als Telefon verwendet wird, für die Kommunikation mit entfernt lebenden Familienmitgliedern verwendet werden. In einem Krankenhaus könnte über diese Funktion zusätzlich das Personal gerufen werden (besonders wenn der Patient in der Bewegung eingeschränkt ist). Diese Anwendungen bilden einen fließenden Übergang zum nächsten großen Einsatzort humanoider Roboter: dem Gesundheitswesen. Neben dem Einsatz zur Unterstützung weiterer Therapiemethoden kann ROS-E auch in Krankenhäusern oder Pflegeeinrichtungen eingesetzt werden. Einerseits kann eine Person, die schon eine ROS-E besitzt, diese bei einem Krankenhaus- oder Pflegeaufenthalt mitbringen. Dadurch kann die Person moralische Unterstützung durch ein „bekanntes Gesicht“ und Ablenkung von Schmerzen oder Stress durch die fremde Umgebung gewinnen. Andererseits kann ROS-E aber auch in die vorhandene Infrastruktur eingebunden werden, z.B. als Alternative für das Drücken eines Rufknopfes. Für den Fall, dass der Rufknopf außer Reichweite oder der Patient nicht in der Lage ist, ihn zu betätigen, kann er durch Sprache nach Hilfe fragen. Dabei können sogar zusätzliche Informationen übermittelt werden. Auch in Einrichtungen, in denen die Bewohner für längere Zeit leben, wie Pflegeheime oder Wohnheime könnte ROS-E die Bewohner und die Pflegekräfte unterstützen.

In jedem dieser Einsatzgebiete können viele verschiedene Situationen mit und um ROS-E auftreten, die erkannt werden müssen, um darauf reagieren zu können. Wie eine solche Situation im Rahmen dieser Arbeit definiert wird, ist im folgenden Abschnitt aufgeführt.

3.2 Definition einer Situation

Um Situationen erkennen zu können, muss zunächst geklärt werden, was eine Situation ist. Der abstrakte Begriff „Situation“ soll für die Verwendung im Zusammenhang mit Künstlichen Neuronalen Netzen und humanoiden Robotern eingegrenzt werden.

Zunächst soll festgehalten werden, was das Ziel der Betrachtung von Situationen ist. Die Erkennung einer Situation hätte weder für einen Roboter, noch für einen Menschen (oder andere Lebewesen) eine Relevanz, wenn aus der Erkennung nicht eine angepasste Handlung abgeleitet werden könnte. Angenommen ein Mensch oder ein Roboter legt jeden Tag zur gleichen Zeit denselben Weg zurück. An einem Tag liegt jedoch ein Balken quer über dem Weg. Die Erkennung der Situation hätte keinen Nutzen, wenn der Mensch oder der Roboter ihre Handlung nicht z.B. auf das Umgehen oder Übersteigen des Balkens anpassen könnten. Daher besteht die grundlegendste Aufgabe des menschlichen Gehirns darin, die Sinneseindrücke zu interpretieren, um eine Handlungsempfehlung geben zu können. Das grundlegendste Ziel dieser Handlungen ist es, den Menschen am Leben und die Körperfunktionen in einem stabilen Zustand zu halten. Auch humanoide Roboter sollen in der Lage sein, Handlungen abhängig von ihrer Umgebung auszuführen, um nicht an einem Balken zu scheitern.

Eine mögliche Quelle für eine Definition des Begriffes „Situation“, die hier betrachtet werden soll, ist der Duden. Die Bedeutung des Wortes im allgemeinen Sprachgebrauch ist vor allem relevant, da diese zeigt, welche Aspekte einer Situation für Menschen im Alltag wichtig sind. Folgende Beschreibung ist im Duden für das Wort Situation zu finden:

„Verhältnisse, Umstände, in denen sich jemand [augenblicklich] befindet; jemandes augenblickliche Lage“ [10]

Diese Definition zeigt zunächst, dass eine Situation eine zeitliche Komponente besitzt. Das Wort „augenblicklich“ gibt dabei jedoch keine feste Zeitspanne für die Dauer der Situation an. Eine Situation kann auch rückblickend oder über einen längeren Zeitraum betrachtet werden.

Daraus ergibt sich die erste wichtige Frage für die Erkennung einer Situation mit einem KNN: Wie lange dauert eine Situation? Wird lediglich ein Bild als Eingabe verwendet, kann dies den „Augenblick“ abbilden. Sollen Audio-Daten verwendet werden, muss zwangsläufig ein Zeitraum gewählt werden, da z.B. eine Frequenz nur über einen gewissen Zeitraum bestimmt werden kann. Weiterhin würde eine Zeitspanne von einigen Millisekunden zwar ausreichen, um Frequenzen zu bestimmen, es wäre jedoch nicht möglich, gesprochene Wörter abzubilden.

Die Definition des Duden enthält weitere abstrakte Begriffe: Verhältnisse, Umstände und Lage. Um seine Lage, Verhältnisse und Umstände zu ermitteln, muss der Mensch zunächst Informationen aus

der Umwelt aufnehmen. Der Mensch besitzt mehrere Sinne (Sehen, Hören, Fühlen, Riechen, Schmecken, Gleichgewichtssinn, Muskelsinn und *Interozeption*), um den Zustand seines Körpers und seine Umwelt zu erfassen. Diese Sinneseindrücke werden als Nervenimpulse an das Gehirn weitergegeben. Die wichtigsten Mechanismen dahinter werden in Abschnitt 4.2 kurz betrachtet, um zu ermitteln, ob diese auf Künstliche Neuronale Netze übertragen werden können.

Ein humanoider Roboter kann eine Vielzahl an Sensoren besitzen, um Informationen aus der Umwelt aufzunehmen. Einige davon könnten sogar mehr oder genauere Informationen liefern, als ein Mensch aus der Umwelt aufnehmen kann. Einige Beispiele dafür sind Infrarot-Kameras, Thermometer oder Sensoren, die Anteile bestimmter Gase in der Luft ermitteln.

Eine Situation stellt sich für einen Roboter an diesem Punkt als eine Aufzeichnung bestimmter Sensordaten über einen definierten Zeitraum dar. Diese Daten müssen im nächsten Schritt interpretiert werden, um eine Handlungsempfehlung daraus ableiten zu können. Folgendes Beispiel für eine Situation soll betrachtet werden:

Ein Mensch sitzt vor einem Buch an einem Tisch und sagt „Ja“.

Der Mensch würde das Buch und den Tisch sehen, vielleicht etwas zu Essen riechen, die Oberfläche des Tisches fühlen, sich selbst sprechen hören, aber auch sein Gleichgewichtssinn und die Lage seines Körpers im Raum und sogar der Zustand seiner inneren Organe werden im Gehirn zur aktuellen Situation zusammengefasst. Diese Eindrücke nimmt der Mensch nicht immer bewusst wahr, sie werden aber zu jeder Zeit im Gehirn verarbeitet. [4] [5]

Der Roboter würde eine kurze Aufzeichnung aus Video- und Audio-Daten und vielleicht den schon vorverarbeiteten Text „Ja“ erhalten.

Die Interpretation des Menschen würde daraus bestehen, die Lage seines Körpers auszuwerten, um sich selbst als sitzend einzuschätzen. Er würde die Position seines Körpers im Verhältnis zum Tisch auswerten, um zu ermitteln, dass er an diesem Tisch sitzt. Dazu muss er allerdings zuvor das Objekt als Tisch interpretiert haben. Auch das Buch muss als solches erkannt worden sein. Schließlich muss der Mensch auswerten, dass das gehörte Wort von ihm selbst ausgesprochen wurde. Mit dieser Interpretation wäre es dem Menschen zumindest möglich, den oben genannten Beispiel-Satz wiederzugeben, wenn er seine aktuelle Situation beschreiben soll.

Der Roboter könnte die Objekte ebenfalls erkennen. Auch die Positionen der Objekte in Beziehung zu einander können mit Künstlichen Neuronalen Netzen ermittelt werden. [11]

Eine Situation kann jedoch von Robotern und Menschen nur selten als unabhängiger Datensatz betrachtet werden. Auch wenn der Roboter alle Objekte und gesprochenen Worte erkennt, stellt sich

die Frage, wozu der Mensch „Ja“ gesagt hat. Hat der Roboter zuvor gefragt, ob er das Licht zum Lesen einschalten soll? Oder ob er im Internet recherchieren soll, ob es eine Fortsetzung des Buches gibt? An dieser Stelle wird deutlich, dass die einzelnen Aufnahmen der Eingabedaten auf eine geeignete Weise miteinander verknüpft werden müssen. Der Roboter benötigt eine Art Gesprächsverlauf für Situationen, um diese im Kontext eines längeren Zeitraumes betrachten zu können. Wie ein Künstliches Neuronales Netz einen längeren Kontext abbilden kann, wird in Abschnitt 6.2 näher untersucht.

Wie zu Beginn dieses Kapitels dargestellt, wäre die reine Erkennung der Situation nur begrenzt hilfreich. Die Situationserkennung könnte entweder von anderen Anwendungen auf dem Roboter verwendet werden, um die aktuelle Situation zu berücksichtigen oder aber die Situationserkennung selbst kann die Funktionen des Roboters zur passenden Situation ausführen. Diese möglichen Arbeitsweisen werden in Abschnitt 5.5 diskutiert. Das Resultat der Auswertung könnte in beiden Fällen das durch ROS-E eingeschaltete Licht zum Lesen oder die aus dem Internet vorgelesene Antwort sein. Der Vollständigkeit halber soll erwähnt werden, dass eine mögliche Reaktion auch darin bestehen kann, keine Funktion auszuführen.

Unter einer Situation wird in dieser Arbeit also die Verarbeitung von Eingabedaten eines bestimmten Zeitraumes durch ein KNN zu einer Interpretation bezeichnet, auf deren Grundlage eine Handlungsempfehlung gegeben werden kann. Um besser einzugrenzen, welche Situationen im Rahmen dieser Arbeit betrachtet werden, sollen in Abschnitt 5.3 weitere Beispiele für Situationen konstruiert werden.

3.3 AI, Machine Learning und Künstliche Neuronale Netze

Die Begriffe *Artificial Intelligence* (AI), im Deutschen *Künstliche Intelligenz* (KI), *Machine Learning* (ML), im Deutschen *Maschinelles Lernen* und *Artificial Neural Network* (ANN), im Deutschen *Künstliches Neuronales Netz* (KNN) werden im alltäglichen Sprachgebrauch vor allem durch die verschiedenen Arten der Medien (Zeitung, Internet, Radio, Fernsehen, …) eingesetzt. Die mathematischen Methoden dahinter werden für eine Vielzahl von Anwendungen eingesetzt, die Menschen täglich nutzen. Einige Beispiele sind Spam-Filter, personalisierte Werbung und Produktempfehlungen oder Spracherkennung. Um die wissenschaftliche Bedeutung der Begriffe zu klären, soll im Folgenden deren Entstehung zusammengefasst werden.

- **1950:** In seinem Paper *Computing Machinery and Intelligence* stellt Alan Turing die Frage vor: „Can machines think?“. In einem Versuch, eine wissenschaftliche Antwort geben zu können, entwickelt er den Turing Test. Dadurch soll es möglich sein, die Intelligenz einer Maschine mit der eines Menschen zu vergleichen. [12]

Der Turing Test ist in der Wissenschaft nicht unumstritten. Dennoch trug dieses Paper dazu bei, dass viele Wissenschaftler darüber nachdachten, wie geistige, menschliche Fähigkeiten mit Maschinen, später hauptsächlich Computern, nachgebildet werden können.

- **1956:** Der Begriff *Artificial Intelligence* wird von John McCarthy erfunden, indem er die erste Konferenz zu diesem Thema am Dartmouth College organisierte. Das sogenannte *Dartmouth Summer Research Project on Artificial Intelligence* war der Grundstein für die Eröffnung des neuen Forschungs- und Fachgebietes der Künstlichen Intelligenz. [13]

Künstliche Intelligenz zu definieren ist mit dem heutigen Forschungsstand nur schwer möglich, da der Begriff der Intelligenz selbst nicht eindeutig definiert werden kann. Der Ansatz von Turing bestand daher darin, einen Test zu entwickeln, mit dem eine Maschine als „so intelligent wie ein Mensch“ eingeschätzt werden kann, oder eben nicht, anstatt die Maschine anhand einer Definition dieser Kategorie zuzuweisen. Wie für viele abstrakte Begriffe gibt es auch für den Begriff der Künstlichen Intelligenz mehrere Definitionen aus verschiedenen Perspektiven. Die wortwörtliche Bedeutung besteht aus der Nachahmung (meist durch ein Computerprogramm) der menschlichen Intelligenz. Diese Intelligenz kann jedoch eine Vielzahl an Fähigkeiten, wie die Fähigkeiten zur „Problemlösung“ oder „Entscheidungsfindung“ des Menschen meinen. Aus der technischen Perspektive werden darunter alle Computerprogramme zusammengefasst, die menschliches Verhalten nachahmen. Dieses Verhalten muss jedoch nicht selbst erlernt sein, sondern kann durch einen Algorithmus fest vorgegeben sein. Ein Beispiel hierfür wäre ein sogenannter NPC (Nicht-Spieler-Charakter) in einem Videospiel. Diesem kann zum Beispiel eine feste Laufstrecke zugewiesen werden und eine festgelegte Animation

wird abgespielt, wenn ein Spieler sich nähert. Dadurch ahmt er menschliches Verhalten nach, das jedoch fest vorgegeben wurde. [14]

Ein wichtiger Teil menschlicher Intelligenz ist die Fähigkeit, zu lernen. Auch für den Begriff des Lernens gibt es eine Vielzahl an Definitionen. Auf verschiedenen Ebenen wird dabei ein Prozess definiert, der bestimmte Sinneseindrücke ins menschliche Gehirn aufnimmt, dieses daraufhin verändert (z.B. durch Hinzufügen von Wissen), sodass sich anschließend das Verhalten des Menschen ändert. Diese Verhaltensänderung kann sich zum Beispiel in Form einer neuen Fertigkeit, wie Radfahren oder dem Abrufen eines Faktes, wie der Jahreszahl der Erfindung des Wortes „Künstliche Intelligenz" zeigen. Dieser Mechanismus des Aufnehmens von Eingangsdaten (Sinneseindrücken), der Veränderung des aktuellen Wissensstandes auf deren Grundlage und der anschließenden Veränderung der Ausgabedaten (Verhalten) wurde durch verschiedene mathematische Modelle abgebildet. Einem Programm werden in der Lernphase (viele) Beispiele gezeigt, die auf der Grundlage aller zuvor gesehenen Beispiele verrechnet werden. Ein einfaches Beispiel hierfür ist der *k-Nearest Neighbor Algorithm* (k-Nächste Nachbarn Algorithmus), bei dem Datensätze in Gruppen (Cluster) eingeteilt werden. Dazu wird für jeden neuen Datensatz der Abstand (Euklidischer Abstand, Manhattan-Metrik, …) zu den zuvor hinzugefügten Datensätzen ermittelt. Der neue Datensatz wird der Klasse zugeordnet, denen die Mehrzahl der k nächsten Nachbarn angehört. Maschinelles Lernen bezeichnet also alle Verfahren, die anhand von Beispielen ein statistisches Modell aufstellen und so Zusammenhänge in den Daten selbst lernen, ohne dass ein Mensch dieses Wissen vorgeben musste. Beispiele für solche Verfahren sind Bayessche Netze, Hidden Markov Models, Entscheidungsbäume, Support Vector Machines oder die Hauptkomponentenanalyse (PCA). Einige davon besitzen bereits eine netzartige Struktur, sind aber keine Künstlichen Neuronalen Netze. [15] [16] [17]

Als KNNs werden Netze bezeichnet, deren Knoten den Neuronen im menschlichen Gehirn nachempfunden wurden. Damit sind KNNs eine Teilmenge des Maschinellen Lernens.

- **1967:** Das Modell eines menschlichen Neurons wird von Frank Rosenblatt in Form eines Rechners, dem „Mark 1 Perceptron" gebaut.

Das *Perceptron* bildet die Grundlage für die heutzutage verwendeten Künstlichen Neuronalen Netze. Die mathematischen Grundlagen wurden übernommen und weiterentwickelt. Die Grenzen eines einzelnen Knotens (*Perceptron*) wurden durch die Zusammenstellung immer komplexerer Netze aus diesen einzelnen Knoten überwunden. Diese Netze können Probleme wie Klassifizierung (Objekterkennung, Spracherkennung, …) und Clusterbildung lösen, aber auch Texte übersetzen oder Bilder und Videos generieren. [13] [14] [18]

Künstliche Intelligenz ist der Oberbegriff für alle Algorithmen und Methoden, die menschliches Verhalten mit einem Computer oder einer Maschine nachahmen. Ein Teil dieser Algorithmen und Methoden erreicht dies, indem anhand von Beispieldaten ein Modell (z.B. ein Regelsatz bei einem Spiel) erlernt wird, ohne dass ein Mensch dieses definieren musste. Dieses Teilgebiet der *Künstlichen Intelligenz* heißt *Machine Learning*. *Künstliche Neuronale Netze* sind eine solche Methode des *Machine Learning*.

Zusammenfassung des Kapitels

In diesem Kapitel wurden die drei wichtigen Eigenschaften humanoider Roboter definiert: die Gestalt, die Sensorik und die Intelligenz oder Autonomie des Roboters. Die inhaltliche Abgrenzung der Begriffe „Künstliche Intelligenz“, „Machine Learning“ und „Künstliche Neuronale Netze“ ist für das Verständnis der nächsten Arbeitsschritte erforderlich.

4 Grundbausteine für eine Situationserkennung

Im vorherigen Kapitel wurde festgestellt, dass ein humanoider Roboter nicht nur aufgrund seiner Bauform als solcher bezeichnet werden kann, sondern auch andere Eigenschaften von Menschen nachahmen kann. Dazu könnte auch die Verarbeitung von Sinneseindrücken bzw. Sensordaten als Situation zählen.

In diesem Kapitel wird untersucht, wie das Neuronale Netz im menschlichen Gehirn eine Situation erkennt und wie ein Künstliches Neuronales Netz diesen Vorgang nachahmen könnte. Die Aufgabe der Situationserkennung soll in einzelne Teilprobleme aufgeteilt werden, die es im Rahmen dieser Arbeit zu lösen gilt. Dazu wird das menschliche Gehirn als Vorbild verwendet, um ein Beispiel für eine mögliche Lösung der Probleme zu analysieren. Anschließend wird versucht, diese Lösung auf Roboter und Künstliche Neuronale Netze zu übertragen.

4.1 Wahrnehmung

Im nachfolgenden Abschnitt soll näher untersucht werden, wie ein Mensch eine Situation erkennt, wie das menschliche Gehirn Sinneseindrücke verarbeitet und welche Gemeinsamkeiten und Unterschiede zur Arbeitsweise von KNNs bestehen.

4.1.1 Die 8 Sinne des Menschen

Die grundlegende Aufgabe des Gehirns ist es, den Menschen am Leben und zu halten, sodass er seine Gene an die nächste Generation weitergeben kann. Dazu reguliert es die Körperfunktionen und den Stoffhaushalt, sodass der Körper in einem stabilen Zustand bleibt. Dieses Regulieren wird Allostase genannt und beschreibt den Prozess kurzfristiger Änderungen, um langfristig eine Stabilität zu erreichen. Die kurzfristigen Veränderungen kann das Gehirn durch die Ausschüttung von Hormonen und anderen Stoffen hervorrufen. Ein Beispiel dafür wäre die Ausschüttung von Energie in Form von Glukose für Muskelzellen, wenn ein Mensch plötzlich zum Zug sprinten muss. Um diese Aufgabe zu bearbeiten, benötigt das Gehirn Informationen über den Zustand des Körpers und dessen Umwelt. Diese Informationen werden von verschiedenen Sinnesorganen an das Gehirn geleitet.

Eine Sinneszelle (Rezeptor) erzeugt aus einer chemischen oder physikalischen Größe (z.B. Lichtintensität bzw. Photonenstromdichte beim Sehen) eine elektrische Spannung (lokales Potential). Dieses Signal wird an andere Nervenzellen weitergegeben und dabei der Informationsgehalt in Form der Frequenz, also der Anzahl der Impulse pro Sekunde, codiert. Dieses sogenannte Aktionspotential wird zum Gehirn geleitet. Ein Sinn ist demnach die Fähigkeit zur Umwandlung einer externen Energie

in einen Nervenimpuls durch eine Sinneszelle, dessen Weiterleitung und Verarbeitung im Gehirn um letztendlich eine Handlung daraus abzuleiten. Welche Sinne besitzt ein Mensch nach dieser Definition?

Zum einen gibt es die fünf „klassischen“ Sinne. Zu ihnen zählen:

- der Sehsinn

Über die Sinneszellen (Zapfen und Stäbchen) der Augen kann ein Mensch elektromagnetische Wellen der Wellenlänge zwischen 780nm (rot) und 380nm (violett) bzw. der Frequenz von 384THz bis 789THz in elektrische Impulse umwandeln. Dabei besitzt die Retina bereits fünf verschiedene Arten von Rezeptoren.

- der Geruchsinn

In der Riechschleimhaut des Menschen gibt es ca. 30 Millionen Sinneszellen mit 350 verschiedenen Rezeptortypen. Gerüche sind chemische Verbindungen in der Luft, die von bestimmten Rezeptoren erkannt werden können. Dabei ist ein Rezeptortyp häufig auf einen bestimmten Duftstoff spezialisiert. Duftgemische werden über das Zusammenspiel verschiedener Rezeptortypen erkannt.

- der Geschmackssinn

Der Geschmackssinn ist ein wichtiger Sinn, da dieser zur Entscheidung beiträgt, welche Stoffe in den Körper aufgenommen werden und wie die Qualität der Nahrung ist. Der Mensch hat nicht nur auf der Zunge, sondern auch am Gaumen und in anderen Bereichen des Mundraumes Rezeptoren, die chemische Bestandteile der Nahrung erfassen.

- der Tastsinn

Der Tastsinn setzt sich aus verschiedenen Arten von Rezeptoren zusammen. Die Mechanorezeptoren werden in Berührungsrezeptoren für kurze, leichte Berührungen, die Druckrezeptoren die eine längere, stärkere Verformung der Haut codieren und Vibrationsrezeptoren aufgeteilt. Dur Empfindung der Temperatur besitzt der Mensch zwei verschiedene Arten von Rezeptoren, Kälterezeptoren und Wärmerezeptoren, da es entscheidend ist, ob dem Körper Wärme zugeführt oder entzogen wird.

- der Hörsinn

Ein Mensch kann Schallwellen der Frequenz von ca. 20 Hz bis 20kHz in einem Intervall von bis zu 50 Tönen in der Sekunde über die Ohrmuschel aufnehmen. Dort werden sie über das Trommelfell aufgenommen, über Hammer, Amboss und Steigbügel verstärkt und zur Hörschnecke (Cochlea) geleitet. Diese besitzt eine Membran mit vielen Haarzellen, die durch den Schall bewegt werden und diesen mechanischen Reiz in elektrische Impulse umwandeln.

Aus dieser Aufzählung wird bereits deutlich, dass die Einteilung der Sinne keinen eindeutigen regeln folgt. Sie wird von vielen Wissenschaftlern unterschiedlich vorgenommen und nicht alle Sinne des Menschen sind derzeit erforscht. Es gibt aber einige weitere Sinne des Menschen, die hier betrachtet werden sollen:

- der Gleichgewichtssinn

Dieser Sinn wird durch die sogenannten Vestibular-Apparate ermöglicht. Diese enthalten je zwei waagerecht und senkrecht ausgerichtete Unterorgane, die Maculae, die die Gravitation der Erde und die zugehörige Horizontale abbilden. So kann ein Mensch oben und unten und den Horizont als Waagerechte wahrnehmen. Mit deren Hilfe kann das Gehirn ebenfalls die Neigung des Kopfes ermitteln. Die Bogengänge sind ein weiteres Unterorgan des Apparates, die Drehbewegungen des Kopfes ermitteln.

- die Propriozeption (Körpersinn)

Dieser Sinn bildet die Lage der Gliedmaßen im Raum, sowie deren Bewegung, Geschwindigkeit und Schwere ab. Dadurch ist der Mensch in der Lage zu wissen, dass er auf dem Boden steht, ohne hinzusehen oder im Dunkeln zu wissen, wo sein Arm sich befindet.

- die Interozeption

Dieser Sinn fasst alle Wahrnehmungen der inneren Organe zusammen. In mancher Literatur wird dieser in mehrere Sinne unterteilt. Zur *Interozeption* gehört beispielswiese ein Sinn, der den Blutdruck messen kann. Dafür gibt es spezielle Sinneszellen, die in der Wand der Herzvorhöfe, im Aortenbogen und in der Gabelung der Halsschlagadern liegen. Mit Hilfe dieser Sinneszellen ist es dem Gehirn möglich, den Herzschlag zu regeln. Der Mensch besitzt ebenfalls einen Sinn für den CO_2-Gehalt des Blutes, mit dem die Atmung reguliert wird. Im Hypothalamus liegen Nervenzellen, die den Blutzuckerwert (Glukose) ermitteln und bei Unterzuckerung benachbarten Gehirnareale signalisieren, das Gefühl von Hunger zu erzeugen. Für die Wahrnehmung der inneren Organe besitzt der Mensch zum Beispiel Sinneszellen in Magen und Darm, die den Dehnungszustand abbilden. In Niere, Leber und Hypothalamus gibt es zusätzlich sogenannte Osmorezeptoren, die den Salzgehalt des Blutes messen und bei hypertonen Lösungen (das Blut ist salzhaltiger als die Zelle) einen Stoff ausschütten, der Durst hervorruft. Ein weiterer Sinn, der dem Menschen nicht bewusstwird, ist der für die Messung der Bluttemperatur. Nicht zuletzt besitzt der Mensch Schmerzsensoren in verschiedenen Organen.

Diese Sinne sind sehr schwer in Laborversuchen zu untersuchen, da die Sinneszellen im Körper eines Menschen untersucht werden müssten. Daher sind viele Aspekte der Weiterleitung und Verarbeitung der Sinneseindrücke nicht vollständig erforscht. [19] [20]

Die wichtigste Erkenntnis dieses Abschnittes ist, dass der Mensch nur einen sehr kleinen Teil der Aufnahme und Verarbeitung der Sinne bewusst wahrnehmen kann. Das bedeutet, dass das menschliche Gehirn im Laufe des Lebens feste Abläufe lernt, über die der Mensch in vielen Fällen nicht mehr nachdenken muss. So kommunizieren Menschen häufig, ohne sich bewusst zu machen, was sie ausdrücken. Diese Tatsache ist einer der Gründe, warum es so schwer ist, die menschliche Kommunikation für Roboter in Regeln zu fassen. Selbst wenn der Roboter einen bestimmten Gesichtsausdruck anhand eines Bildes mit sehr hoher Genauigkeit erkennen kann, müsste der Mensch diesen Gesichtsausdruck auch immer nur verwenden, wenn er eine bestimmte Reaktion des Roboters erzielen möchte. Eine mögliche Lösung wäre, einem Roboter nicht nur die Bedeutung von Gesichtsausdrücken vorzugeben, sondern ein eigenes System, mit dem er die Umwelt durch seine „Sinne“ wahrnehmen und eigene Reaktionen auf die Sinneseindrücke selbst lernen kann.

> Der menschliche Körper besitzt viele verschiedene Sinne, die mit dem heutigen Stand der Technik nicht vollumfänglich erforscht werden können. Das Gehirn nutzt die Sinneseindrücke, um die aktuelle Situation der Umwelt und des eigenen Körpers wahrzunehmen.

Im nächsten Abschnitt soll untersucht werden, welche Sinne ein Roboter besitzen könnte, um eine Situation wahrnehmen zu können.

4.1.2 Die Eingangsdaten von humanoiden Robotern

Die Sinne des Menschen wurden so genau betrachtet, um einen Referenzwert für die Art der übertragenen Informationen und deren benötigte Genauigkeit zu ermitteln. Die Sinne des Menschen wurden über Millionen von Jahren durch die Evolution beeinflusst. Das menschliche Gehirn ist ein Beispiel dafür, wie gut diese Sinne geeignet sind, um die Aufgaben der Allostase (Regulierung der Körperfunktionen) und der Erhaltung der Art zu lösen. Es ist jedoch nicht das einzige Beispiel, das diese Aufgaben erfolgreich löst. Im Tierreich gibt es viele verschiedene weitere Sinne. Ein Mensch kann zum Beispiel nur einen kleinen Teil des Lichtspektrums sehen, während Vögel auch ultraviolettes Licht wahrnehmen können. Es stellt sich also die Frage, ob die Sensoren eines humanoiden Roboters so genau wie möglich an die des Menschen angelehnt werden oder ob andere Sensoren ein besseres Ergebnis für eine Situationserkennung liefern würden. Ein Roboter könnte so z.B. über Infrarot-Kameras oder Abstandssensoren auch Daten aus der Umwelt verarbeiten, die der Mensch nicht wahrnehmen kann. Dieselbe Überlegung ergibt sich bei der Wahl der Genauigkeit der Eingangsdaten. Die Sinne des Menschen sind nicht alle so genau, wie sie mit Sensoren abgebildet werden könnten. Der Temperatursinn oder der Sinn für den CO_2-Gehalt des Blutes sind beispielsweise viel ungenauer

als ein Thermometer oder ein CO_2-Sensor wären. Dabei ist der Mensch durch diese Ungenauigkeit nicht unbedingt unterlegen, denn diese Ungenauigkeit spart Ressourcen. [19]

Der Einfluss der einzelnen Sinne des Menschen auf die Erkennung einer Situation ist, genau wie die Sinne selbst, nicht vollständig erforscht. Daher kann nicht vorhergesagt werden, ob z.B. die Bluttemperatur einen größeren Einfluss auf die Entscheidung hat, als man bisher untersuchen konnte. Von anderer Seite betrachtet, können Menschen auch ohne die als sehr wichtig angesehenen Sinne wie Sehen und Hören eine Situation bestimmen. Die Anzahl der erkennbaren Situationen ist bei einem Menschen mit Sehbehinderung oder Blindheit reduziert, er kann aber dennoch überleben und seine Körperfunktionen regeln.

Beide Fragen, nach der Auswahl der Sensoren und deren Genauigkeit, müssten wissenschaftlich untersucht werden, indem mehrere Künstliche Neuronale Netze entwickelt werden, die jeweils alle möglichen Kombinationen von Sensoren und Genauigkeiten testen (oder zumindest eine repräsentative Auswahl). Dieser Ansatz wird besonders im Bereich des *Machine Learning* häufig verwendet, da die Ergebnisse schwer vorhersehbar sind und häufig das nötige Vorwissen fehlt. Im Rahmen dieser Arbeit steht dafür allerdings nicht genügend Zeit zur Verfügung.

An dieser Stelle sollen einige Sensoren aufgezählt werden, deren Daten ein Roboter als „Sinnesorgan“ verwenden könnte: eine Kamera, eine Infrarot-Kamera, zwei Kameras für Stereovision, ein Mikrofonarray, ein Touch-Display (externes Gerät) oder Touch-Sensoren, Hautwiderstandssensoren, Abstandssensoren (LiDAR, Ultraschall …), einen Temperatursensor (außen und innen), Gassensoren (CO_2, VOC, …), einen Feuchtigkeitssensor, einen GPS-Sensor, einen Helligkeitssensor, einen Lichtschrankensensor, einen Gyro-Sensor, Vibrationssensoren oder einen Kompass.

Neben diesen Sensoren, die Informationen über die Umwelt sammeln, wäre es denkbar, auch andere Datenquellen zu nutzen. Möchte man die Einschätzung der aktuellen Situation möglichst menschenähnlich nachbilden, könnte die folgende Frage sehr entscheidend sein: Benötigt der Roboter auch Daten über seinen eigenen Zustand, um eine Situation zu erkennen bzw. anschließend mit seiner Umwelt zu interagieren? Ist es dieser Aspekt, der Robotern heutzutage meist noch fehlt?

Folgende interne Daten könnten bereits digital auf humanoiden Robotern vorliegen: die Uhrzeit, die CPU-Auslastung, die RAM-Auslastung, der Akkustand bzw. Netzmodus oder Text (von gesprochenen Sätzen vom Nutzer). Genau wie in der Schule nur die nach außen gerichteten Sinne des Menschen betrachtet werden, werden auch beim *Machine Learning* mit Robotern vorrangig Daten der Außenwelt verarbeitet. Die Einbeziehung interner Daten und vor allem die Definition einer übergeordneten Aufgabe des Roboters, ähnlich der des Menschen, könnten zu ganz neuen Ergebnissen des *Machine*

Learning mit Robotern führen. Leider kann diese Frage nicht in Rahmen dieser Arbeit untersucht werden.

Die Auswahl der Sensoren im Rahmen dieser Arbeit hängt von drei Kriterien ab. Das erste dieser Kriterien ist die Häufigkeit der gewählten Datentypen in aktuellen Forschungsarbeiten und Industriezweigen. Die Erfolgreiche Umsetzung eines KNNs hängt stark davon ab, ob geeignete Trainingsdaten vorhanden sind. Da es sehr wahrscheinlich ist, dass nicht ausreichend viele Trainingsdaten im Rahmen dieser Arbeit selbst gesammelt werden können, können keine Sensordaten gewählt werden, zu denen keine Datensets im Internet veröffentlicht wurden. Da sehr viele Entwickler von KNNs dieses Problem haben, gibt es auch die meisten Veröffentlichungen von Netz-Architekturen zu diesen häufig verwendeten Datentypen. Zu den sehr häufig verwendeten Datentypen gehören Bilder, Audiodaten und Texte. Zahlreiche Unternehmen und Forschungseinrichtungen veröffentlichen heutzutage sehr große Datensets, die kleinere Unternehmen oder Forscher für das Training ihrer Netze verwenden können. Diese haben meist nicht die Ressourcen, um genügend Daten selbst zu sammeln. Besonders im Bereich *Computer Vision*, also der Verarbeitung von Bildern und Videos gibt es große Datensets. [21]

Das zweite Kriterium bildet die Häufigkeit der gewählten Sensoren in humanoiden Robotern. Kameras und Mikrofone werden vermutlich häufiger in humanoiden Robotern eingesetzt als spezielle Gassensoren oder Hautwiderstandssensoren. Selbst humanoide Roboter sind nicht unbedingt mit zwei Kameras oder Touch-Displays ausgestattet. Da das erstellte Künstliche Neuronale Netz mit dem Tisch-Roboter ROS-E getestet werden soll, ist das dritte Kriterium die Verfügbarkeit auf ROS-E. Die Kamera und das Mikrofonarray sind die Sensoren, über die ROS-E selbst Informationen über ihre Umwelt erhalten kann.

Im Rahmen dieser Arbeit werden daher Video- und Audiodaten aus einer Kamera und einem Mikrofon für die Situationserkennung verwendet. Damit kann die Situationserkennung auf den meisten humanoiden Robotern, aber auch auf einem Computer oder Smartphone verwendet werden. Außerdem ist sichergestellt, dass genügend Trainingsdaten für einen Test beschafft werden können.

Da das Netz nicht nur einen Eingangskanal berücksichtigen soll, muss im nächsten Schritt ermittelt werden, wie diese Eingangskanäle verknüpft werden können. Dazu wird wieder das menschliche Gehirn als Vorbild verwendet und die Erkenntnisse anschließend auf Künstliche Neuronale Netze übertragen.

4.2 Sensorische Integration und multimodales Lernen

In diesem Abschnitt soll untersucht werden, wie die Sinneseindrücke eines Menschen zu einem Abbild der Umwelt und des eigenen Körpers verarbeitet werden. Anschließend werden davon inspirierte Mechanismen vorgestellt, mit denen Künstliche Neuronale Netze verschiedene Eingangsdaten zu einer internen Repräsentation verknüpfen können.

4.2.1 Zusammenfassung der Sinne im menschlichen Gehirn

Durch die Sinneszellen werden chemische oder physikalische Reize in elektrische Signale umgewandelt. Diese elektrischen Signale werden vom jeweiligen Sinnesorgan an das Gehirn weitergeleitet. Dabei kann sogar schon auf dem Weg zum Gehirn eine erste Verarbeitung der Daten stattfinden. Diese Verarbeitung ist allerdings noch weniger erforscht als jene im Gehirn. Auf eine noch unbekannte Weise werden die Sinneseindrücke eines Menschen also von den Nervenbahnen und dem Gehirn so verarbeitet, dass er daraufhin seine Umwelt einschätzen kann. Einige Erkenntnisse der Gehirnforschung sollen im Folgenden betrachtet werden, um zu sehen, ob diese mit Künstlichen Neuronalen Netzen nachgebildet werden können.

Im menschlichen Gehirn finden zu jeder Zeit Übertragungen zwischen Nervenzellen statt. Das Gehirn benötigt die kontinuierlichen Informationen der Sinne, um den Körper regulieren zu können. So werden ständig alle Sinneseindrücke verwendet, um den aktuellen Zustand des Körpers und dessen Lage in der Umwelt zu interpretieren, auch wenn der Mensch nur einen kleinen Teil davon bewusst wahrnimmt. Besonders die Sinne der *Interozeption*, also der inneren Organe können bewusst nicht genau wahrgenommen werden. [4] [5]

Das menschliche Gehirn hat durch die Evolution einen guten Mittelweg zwischen der Aufnahme möglichst vieler Informationen aus der Umwelt und dem möglichst geringen Energieverbrauch für die Verarbeitung von Sinneseindrücken entwickelt. Dadurch wird das folgende grundlegende Problem der Verarbeitung mehrerer Eingangskanäle gelöst. Bei der Aufnahme verschiedener Sinneseindrücke können generell drei Situationen eintreten:

1. die Sinne liefern völlig unabhängige Daten
2. die Sinne liefern Daten, die zur selben Interpretation führen
3. oder sie widersprechen sich

Wenn die Sinne völlig unabhängige Daten liefern, wird zwar keine Energie für redundante Daten verbraucht. Sollten die Daten jedoch fehlerhaft sein, können sie von keinem anderen Sinn korrigiert werden.

Wenn die Sinneseindrücke zum selben Ergebnis führen, wird zwar die Sicherheit einer richtigen Interpretation erhöht, aber gleichzeitig auch Energie verschwendet, um redundante Daten zu verarbeiten.

Wenn sich die Sinneseindrücke widersprechen, muss das Gehirn eine Entscheidung treffen. Dabei lernt es lebenslänglich weiter, welcher Sinn in welchem Fall der verlässlichere sein wird. Eine Gelegenheit, dieses Verhalten des Gehirns bewusst erleben zu können, ist eine audiovisuelle Täuschung, der sogenannte Mc Gurk-Effekt. Dabei wird eine Audioaufnahme vorgespielt, in der eine einfache Silben wie „Ba“ wiederholt ausgesprochen werden. Gleichzeitig werden Videoclips dazu gezeigt, in denen ein Mensch verschiedene Mundbewegungen wie „Ba“ oder „Fa“ zur Audiospur macht. Der Zuschauer kann beobachten, wie er glaubt, die Silbe „Fa“ nun auch wirklich zu hören, obwohl die Audiospur nicht geändert wurde. Eine weitere audiovisuelle Täuschung führt ein Bauchredner vor, wobei der Ursprung eines Geräusches scheinbar in ein anderes Objekt interpretiert wird. Der Effekt tritt auch im Kino auf, wo die Stimmen der Schauspieler aus Lautsprechern kommen, aber der Ursprung von den Zuschauern auf die Personen auf der Leinwand projiziert wird. [22]

Das Gehirn verknüpft die Sinneseindrücke auf eine Weise, sodass es zum Beispiel möglich wird, den Ursprung eines Geräusches mit dem Anblick einer Katze zu verknüpfen und somit die vielen Möglichkeiten einer Situation unterscheiden zu können. Wie genau diese Verknüpfung stattfindet ist mit heutigen Mitteln nicht vollständig erforschbar. Die Gehirnforscher arbeiten aber immer öfter mit Forschern für Künstliche Neuronale Netze zusammen, da viele der Abläufe im Gehirn mit diesen Netzen sehr vereinfacht untersucht werden können. Andererseits können neue Erkenntnisse der Gehirnforschung auch wieder zu neuen Architekturen und Vorgehensweisen für Künstliche Neuronale Netze führen. [4] [5]

Ein Beispiel dafür sind die sogenannten *Convolutional Neural Networks* (CNNs), die durch die Verarbeitung von visuellen Reizen im menschlichen Gehirn inspiriert wurden. Im sogenannten Visuellen Cortex werden die visuellen Reize der Netzhaut verarbeitet. Dieser Bereich des Gehirns bildet eine Karte, die die Netzhaut so abbildet, dass die Nachbarschaft der Sinneszellen, also der aufgenommenen „Pixel“ erhalten bleibt. Dabei wird die Netzhaut nicht gleichmäßig im Gehirn abgebildet. Der Mensch kann nur in einem kleinen Bereich seines Sichtfeldes scharf sehen. Dieser Bereich wird auf ein Fünftel des visuellen Cortex projiziert. Dies ist ein weiteres Beispiel für die effiziente Nutzung von Ressourcen durch das menschliche Gehirn. Auch für CNNs spielt die Nachbarschaft der Pixel eine wichtige Rolle. Beide Netze arbeiten in mehreren Schichten, die zunächst einfache Linien, dann Muster und schließlich komplexe Objekte erkennen. [23] [13]

Wenn ein Mensch geboren wird, ist er noch nicht in der Lage Gesichter zu erkennen oder Sprache zu verstehen. Das neuronale Netz (das Gehirn) muss erst die Trainingsphase durchlaufen. Dabei lernt es,

wie auch Künstliche Neuronale Netze, die statistischen Wahrscheinlichkeiten der Eingangsdaten. Beim Sehen tauchen bestimmte Muster immer wieder auf, die als Kanten und später als Gesichter erkannt werden. Beim Hören bedeutet eine bestimmte Abfolge an Frequenzen, dass gleich ein bestimmter Gegenstand vor den Menschen gehalten wird oder eine bestimmte Person anwesend ist.

> Das menschliche Gehirn verknüpft die eintreffenden Sinneseindrücke zu einer komprimierten, internen Repräsentation. Dabei werden unabhängige Sinne zu einem Bild zusammengefasst, redundante Informationen aussortiert und Konflikte sich widersprechender Sinneseindrücke aufgelöst.

Damit das Gehirn die vielen verschiedenen Sinneseindrücke nach ihrer Bedeutung, oder einer nötigen Reaktion ordnen kann, werden die Sinneseindrücke zu einer internen Repräsentation verrechnet. Im folgenden Abschnitt soll untersucht werden, wie Künstliche Neuronale Netze ihre Eingangsdaten zusammenfassen und solche internen Repräsentationen bilden können.

4.2.2 Zusammenfassung von Eingangskanälen durch Machine Learning und Künstliche Neuronale Netze

Die genauen Arbeitsweisen des Gehirns können derzeit noch nicht erforscht werden, da das Gehirn selbst ein sehr komplexes Neuronales Netz ist. Zur Vereinfachung und Veranschaulichung bestimmter Prinzipien verwenden Gehirnforscher daher Künstliche Neuronale Netze, um bestimmte Teile des Gehirns zu modellieren. Die verschiedenen Architekturen der Künstlichen Neuronalen Netze wurden in den letzten 5-10 Jahren komplex genug, um auch die abstrakteren Aufgaben des Gehirns, zum Beispiel die Verknüpfung von Eingangsdaten, nachstellen zu können. Die Forschung ist damit an einem Punkt angekommen, an dem es möglich wird, nicht mehr nur kleine Teilaufgaben zu lösen, sondern größere, zusammengefasste Aufgaben zu betrachten. Die Zerlegung der Aufgaben war historisch bedingt, kann aber für die Lösung komplexerer Aufgaben mit KNNs kontraproduktiv sein. Forscher, wie Rosenblatt, haben sich seit den 1960er Jahren mit unterschiedlichen Anwendungen für „Künstliche Intelligenz“ beschäftigt. Dabei standen nur sehr begrenzte Ressourcen zur Verfügung. Da die Künstlichen Neuronalen Netze noch nicht weit genug entwickelt waren, gab es andere Verfahren, wie die Support Vector Machine (SVMs) oder die Hauptkomponentenanalyse (PCA), bei denen mit Regression und anderen statistischen Algorithmen eine Klassifizierung durchgeführt wird. Die Aufgaben mussten in kleine Teilschritte aufgeteilt werden, um diese nacheinander lösen zu können. Meist konnte dabei nur ein Eingangskanal betrachtet werden. Es bildeten sich einzelne Forschungsbereiche, wie *Computer Vision*, die sich ausschließlich mit der Analyse von Bildern beschäftigte. Auch

heutzutage wird meist ein ganz bestimmtes Problem definiert und dieses mit möglichst wenigen Eingangskanälen (Modalitäten) bearbeitet, da die Beschaffung der Trainingsdaten einfacher wird und Ressourcen gespart werden können. Es haben sich bestimmte Architekturen für bestimmte Aufgaben als besonders effektiv gezeigt. So wird die Analyse von Bildern häufig mit CNNs durchgeführt und für die Verarbeitung von Daten mit einem zeitlichen Ablauf *Recurrent Neural Networks* (RNNs) verwendet. [23] [13]

Mit der zunehmenden Rechenleistung und den immer größeren Datensets werden aber auch Netze entwickelt, die mehrere Modalitäten (meist Text und Bild) verknüpfen. Diese Netze können häufig sehr viel bessere Ergebnisse erzielen. Diese Ergebnisse werden in sogenannten Papers veröffentlicht. Darin wird der Aufbau des Netzes beschrieben und welche neue Frage damit untersucht werden soll. Anschließend wird das Netz in einer bestimmten Aufgabe, zum Beispiel Objektklassifizierung, mit einem öffentlichen Datenset validiert. Da die Aufgabe und die Daten für alle gleich sind, können die Ergebnisse anschließend mit anderen Papers verglichen werden. Zu den derzeit häufig gewählten Aufgaben für Netze mit mehreren Modalitäten zählt beispielsweise das *Visual Question Answering* (VQA), bei dem ein Bild zusammen mit einer Frage „Was liegt neben dem Apfel auf dem Tisch?“ gezeigt wird und das Netz eine Antwort als Text ausgeben muss. Während für Netze heutzutage immer komplexere Aufgaben gewählt werden, werden die Netze selbst jedoch nicht zwingend komplexer. Forscher haben entdeckt, dass viele Architekturen die Aufgaben, die früher einzeln analysiert und umgesetzt wurden, implizit lernen können. Ein CNN lernt zum Beispiel selbstständig die verschiedenen Filter, die zum Erkennen bestimmter Kanten und Muster notwendig sind, die früher noch als Algorithmus auf das Bild angewandt wurden. Ein solches Netz alleine ist aber nicht dafür geeignet, Texte zu verarbeiten. Für die Aufgabe des *Visual Question Answering* ist ein Netz notwendig, das Bilder und Texte verarbeiten und diese darin enthaltenen Informationen sinnvoll verknüpfen kann. [24]

Sollen mehrere Modalitäten verwendet werden, müssen zwei grundlegende Probleme gelöst werden: *Alignment* und *Fusion*. *Alignment* (Ausrichtung) bezieht sich auf die Zuordnung der Eingabedaten verschiedener Modalitäten, die zu einer Sequenz gehören (siehe Abbildung 2).

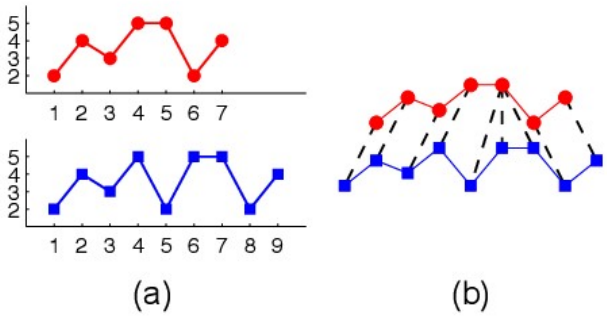

Abbildung 2: Veranschaulichung des Alignment-Problems (Quelle: https://www.semanticscholar.org/paper/Canonical-Time-Warping-for-Alignment-of-Human-Zhou-Torre/51cc757cc4ce56385471c21e47d48084d8f8356b)

Dieses Problem kann auf verschiedene Weise gelöst werden. Die erste Möglichkeit wäre eine manuelle Zuteilung, wenn zum Beispiel ein einzelnes Bild und eine zugehörige Frage durch den Menschen vorgegeben werden. Für Daten mit einem zeitlichen Ablauf ist es sehr aufwändig, diese Zuordnung per Hand durchzuführen. Die zweite Möglichkeit besteht darin, einen Algorithmus zu verwenden, der die Zuordnung errechnen kann. Wenn die Aufgabe durch einen Algorithmus gelöst werden kann, dann gibt es auch Netze, die diese Zuordnung selbst lernen können. Die letzte Möglichkeit besteht darin, eine Architektur für das Netz zu wählen, die das *Alignment* implizit lösen kann. Diese Netze verwenden zum Beispiel den sogenannten *Attention-Mechanismus*, der im nächsten Abschnitt genauer erklärt wird. [25]

Das zweite Problem, das es zu lösen gilt, ist die *Fusion* (Vereinigung) der verschiedenen Modalitäten. Ein KNN muss nun das soeben vom Gehirn gelöste Problem ebenfalls lösen: die Daten können unabhängig sein, redundant sein oder sich widersprechen.

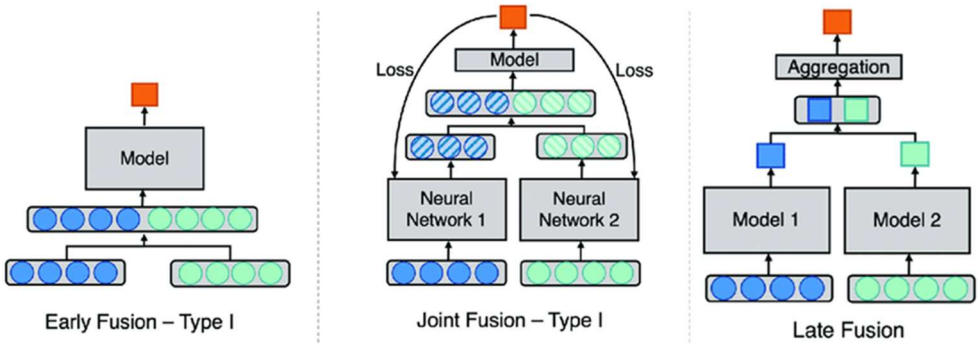

Abbildung 3: Verschiedene Arten der Fusion (Quelle: https://www.researchgate.net/figure/Fusion-strategies-using-deep-learning-Model-architecture-for-different-fusion_fig2_346295743)

Das Problem der *Fusion* kann durch KNNs auf unterschiedliche Arten gelöst werden, die sich ebenfalls historisch weiterentwickelt haben. Die einfachsten Arten sind Early Fusion und Late Fusion. Bei

der Early Fusion werden z.B. aus dem Bild und der Audiospur jeweils ein Vektor erstellt und diese aneinander geschrieben, bevor sie durch das KNN verarbeitet werden (siehe Abbildung 3). Die Late Fusion ist der entgegengesetzte Extremfall, bei dem erst beide Vektoren getrennt durch zwei Netze verarbeitet werden und anschließend aneinander geschrieben werden, bevor eine letzte Auswertung (eine letzte Schicht eines Netzes oder ein Algorithmus) durchgeführt wird. Das Zusammenführen dieser beiden Vektoren kann bei der Joint Fusion auch an einer beliebigen Stelle dazwischen durchgeführt werden. Bei der Entscheidung, an welcher Stelle die *Fusion* durchgeführt wird, kann gezielt Domänenwissen eingebracht werden. Das Ziel der *Fusion* ist es, eine gemeinsame Repräsentation der Eingangsdaten in Form eines Vektors zu erhalten. Auf dessen Grundlage soll anschließend eine Entscheidung, z.B. eine Klassifizierung oder eine Auswahl einer Reaktion getroffen werden. Dieser Vektor ist unso effektiver, je besser er die verschiedenen Zustände der Umwelt in Bezug auf die aktuelle Aufgabe unterscheidbar macht. Wenn dieser Vektor sich kaum verändert, obwohl es hell oder dunkel ist, aber die Aufgabe des Netzes lautet, in der Nacht das Licht einzuschalten, ist der Vektor nicht gut geeignet. Es gibt eine bestimmte Architektur eines KNNs, das dieses Prinzip veranschaulicht: den *Autoencoder* (siehe Abbildung 4). [26]

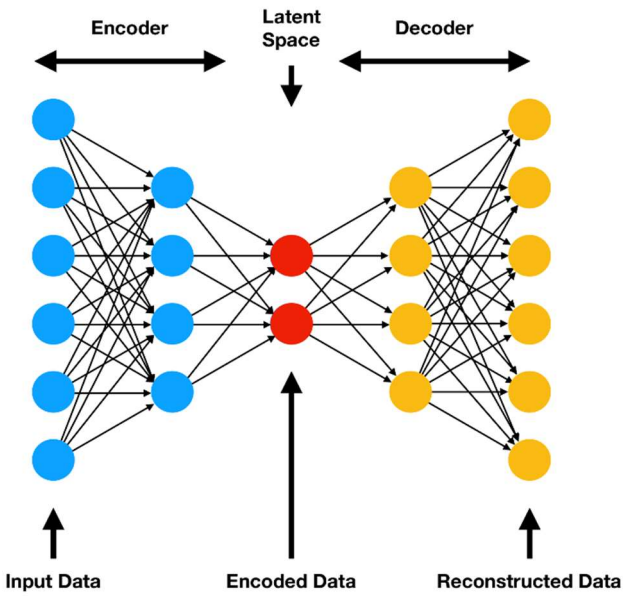

Abbildung 4: Das Prinzip des Autoencoders (Quelle: https://www.compthree.com/blog/autoencoder/)

Dieses Netz wird verwendet, um den Input-Vektor in eine komprimierte Form zu bringen, bei der der *Autoencoder* allerdings noch in der Lage ist, den ursprünglichen Vektor wiederherzustellen. Außer für das Komprimieren von Daten erscheint diese Architektur zunächst nicht sehr nützlich. Sie kann allerdings auch für das Verknüpfen mehrerer Modalitäten verwendet werden (siehe Abbildung 5).

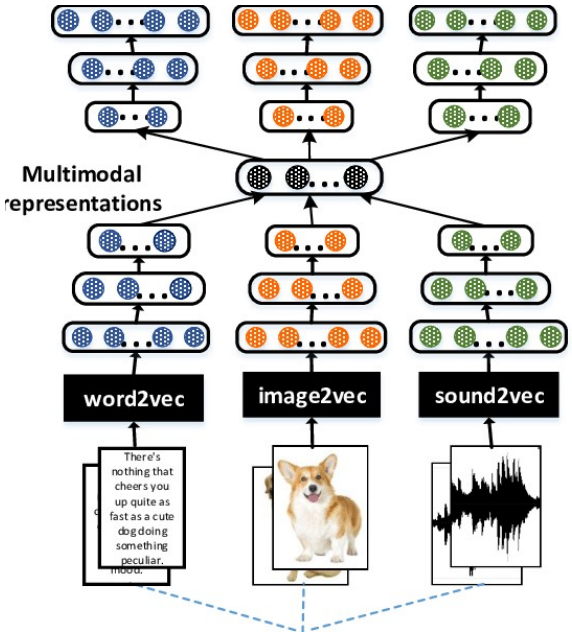

Abbildung 5: Fusion mit einem Autoencoder (Quelle: https://www.researchgate.net/figure/Architecture-of-the-multichannel-autoencoder-with-inputs-of-textual-visual-and-auditory_fig1_334116277)

Dabei werden zum Beispiel Vektoren für ein Bild, eine Audiospur und einen Text erstellt und zu einem gemeinsamen Vektor verknüpft. Die erste sehr nützliche Eigenschaft des *Autoencoders* ist es, dass er keine *Label* für das Training benötigt, das heißt es muss nicht jeder einzelne Datensatz von einem Menschen benannt werden. Da am Ende wieder die Ausgangsdaten erstellt werden sollen, können diese einfach mit dem Ergebnis des Netzes verglichen werden. Nach dem Training kann der *Autoencoder* für eine weitere Aufgabe als die Komprimierung verwendet werden: werden nur zwei der Input-Vektoren gegeben, also z.B. nur Bild und Text, gibt das Netz trotzdem die Vektoren der drei Modalitäten wieder aus und auf diese Weise kann eine Audiospur generiert werden, die zum eingegebenen Bild und Text passt. Diese Eigenschaft wäre vor allem für den Einsatz auf ROS-E wichtig, da der Benutzer das Mikrofon oder die Kamera ausschalten könnte und die Situationserkennung trotzdem weiter funktionieren würde. [27] [28] [29]

Um mehrere Eingangskanäle mit einem KNN zu verknüpfen, müssen die zwei grundlegenden Probleme des *Alignments* und der *Fusion* gelöst werden. Es gibt bestimmte Architekturen von KNNs, die diese Aufgaben explizit oder implizit lösen.

Nicht zuletzt könnte die *Fusion* ebenfalls durch den *Attention-Mechanismus* gelöst werden. Wie genau dieser funktioniert, wird im nächsten Abschnitt erläutert.

4.3 Aufmerksamkeit

Das Konzept der Aufmerksamkeit spielt im menschlichen Gehirn und auch bei Künstlichen Neuronalen Netzen eine wichtige Rolle. In diesem Abschnitt soll die Bedeutung dieses Mechanismus genauer untersucht werden.

4.3.1 Die Rolle von Aufmerksamkeit im menschlichen Gehirn

Ein Mensch reagiert anders auf den eigenen Namen, als auf den einer fremden Person. Er kann die Stimme einer bestimmten Person in einem vollen Raum heraushören. Er kann ein Buch lesen, während ständig Menschen an ihm vorbeilaufen. Um diese Aufgaben zu lösen, benötigt das Gehirn einen Mechanismus, mit dem die eintreffenden Sinneseindrücke priorisiert werden können. Dabei werden alle Sinneseindrücke aussortiert, die keinen Einfluss auf die Reaktion des Körpers haben, um Energie zu sparen. Dabei ist es entscheidend, dass zwei Arten der Auswahl getroffen werden können. Zum einen muss sich der Mensch willentlich auf seine aktuelle Aufgabe konzentrieren können, andererseits muss diese Aufgabe auch durch wichtige Veränderungen der Umwelt unterbrochen werden können. Wenn ein Mensch auf einer Wiese voller Menschen lesen möchte, muss er in der Lage sein, bestimmte Sinneseindrücke auszublenden, um nicht abgelenkt zu werden. Fliegt allerdings ein Ball auf ihn zu, muss er in der Lage sein, das Lesen zu unterbrechen und sich zu ducken, andernfalls könnte er verletzt werden. Daher kann die Aufmerksamkeit in reizgetriebene (stimulus-driven attention) und zielgerichtete (goal-driven attention) Aufmerksamkeit unterteilt werden. [30]

Das Gehirn erhält also die Sinneseindrücke und muss entscheiden, welche davon für die aktuelle Aufgabe oder das Überleben entscheidend sind. Wird diese Entscheidung aufgrund einer erkannten Form oder Farbe getroffen? Oder anhand des Klanges oder ob ein Objekt kalt oder warm ist? In manchen Fällen kann die Entscheidung vermutlich nur anhand mehrerer Faktoren getroffen werden. Das Gehirn benötigt also eine interne Repräsentation, die unabhängig vom jeweiligen Sinn ist, damit Formen und Klänge untereinander verglichen werden können.

Gehirnforscher haben einen Teil des Gehirns identifiziert, der für die Berechnung der Aufmerksamkeit zuständig sein könnte. Das *Frontoparietal Attention Network* (FPAN) codiert die Wichtigkeit von Sinneseindrücken und erstellt eine Repräsentation der Umgebung unabhängig von bestimmten Eigenschaften, Modalitäten und Reaktionen. Die Erkenntnisse stammen meist aus Laborversuchen, in denen mit Hilfe der fMRI (Funktionelle Magnetresonanztomographie) die Aktivität des Gehirns beobachtet wird oder aus Untersuchungen von Patienten mit Gehirnverletzungen. Dabei besteht das Problem, dass das fMRI die Gehirnaktivität in einem Zeitraum von Sekunden sichtbar machen kann, viele Abläufe im Gehirn aber in einem Bruchteil dieser Zeit geschehen. Daher sind auch heutzutage

nicht alle Schlüsse der Forscher vollständig beweisbar. Die zielgerichtete Aufmerksamkeit kann aufgrund der aktuellen Aufgabe, von Zielen oder den Erwartungen des Menschen beeinflusst werden. Solche aktuellen Aufgaben werden in einem Teil des präfrontalen Cortex in einer Art Arbeitsspeicher gehalten, dieser ist mit dem *FPAN* verbunden und beeinflusst die Auswahl der Aufmerksamkeit, so können unwichtige Sinneseindrücke ignoriert und eine Ablenkung von der Aufgabe verhindert werden. Das *FPAN* ist auch in die andere Richtung mit den Teilen des Gehirns für die Vorverarbeitung der Sinneseindrücke verbunden und kann deren Ergebnisse beeinflussen. An dieser Stelle kann auch die Auswahl getroffen werden, welche Sinne ignoriert werden, falls diese redundante oder widersprüchliche Informationen liefern. [30]

Das *FPAN* bildet also die Verbindung zwischen den Sinneseindrücken und den Zielen des Menschen. Es kann entweder die Ziele anhand der Sinneseindrücke oder die Sinneseindrücke anhand der Ziele beeinflussen. Dafür benötigt es eine einheitliche interne Repräsentation der Sinneseindrücke. Diese Repräsentation ist nicht nur unabhängig von den jeweiligen Sinnen, sondern auch von der aktuellen Lage des Körpers. Um diese Repräsentation zu errechnen benötigt das *FPAN* zusätzliche Informationen, wie die Position der Blickrichtung, die Position des Kopfes und des Körpers. Dazu kann das Gehirn zum Beispiel die Bewegung des Auges in die Verarbeitung der Sinneneindrücke der Augen mit einbeziehen. Andernfalls würde sich die Aufmerksamkeit ständig verändern, wenn der Mensch die Blickrichtung verändert oder den Körper bewegt, während er dasselbe Objekt betrachten soll. [30]

> Das *Frontoparietal Attention Network* (FPAN) ist ein Teil des menschlichen Gehirns, das die interne Repräsentation der Sinneseindrücke nach ihrer Wichtigkeit einteilen kann. Es verbindet die Teile des Gehirns, die die Sinneseindrücke verarbeiten mit denen, die für übergeordnete Ziele des Menschen, wie Lesen oder ein Spiel spielen, zuständig sind.

Zwei der hier beschriebenen Eigenschaften des Gehirns wurden auch für Künstliche Neuronale Netze übernommen. Die erste Eigenschaft ist die Fähigkeit, eine unabhängige Repräsentation zu konstruieren. Dabei geht es zum Beispiel darum, ein Objekt unabhängig vom Hintergrund oder Formen unabhängig von ihrer Farbe in einer internen Repräsentation abzubilden. Die zweite Eigenschaft ist das gezielte Aussortieren oder Priorisieren von Eingangsdaten. Im folgenden Abschnitt soll näher betrachtet werden, wie diese Eigenschaften in einem Künstlichen Neuronalen Netz umgesetzt werden können. [31]

4.3.2 Der Attention-Mechanismus für Machine Learning und Künstliche Neuronale Netze

Im menschlichen Gehirn dient der *Attention-Mechanismus* dazu, eine Verbindung zwischen der abstrakten Repräsentation der aktuellen Aufgabe oder dem aktuellen Ziel und den Sinneseindrücken

zu schaffen. Diese Verbindung wird für Künstliche Neuronale Netze jedoch erst dann entscheidend, wenn das Netz eine abstraktere Aufgabe lösen soll. Viele Aufgaben des *Machine Learning*, wie die Objekterkennung der *Computer Vision*, sind historisch gewachsen. Die Aufgabe der Objekterkennung wurde zunächst als Klassifizierung eines Bildes in eine bestimmte Kategorie, wie „Fahrzeug“ oder „Säugetier“ definiert. Diese Aufgabe entspricht in etwa der Vorverarbeitung der Daten, bevor sie im *FPAN* des Menschen ankommen. Daher kann diese Aufgabe von einem CNN auch ohne Attention gelöst werden. Künstliche Neuronale Netze bearbeiten heutzutage jedoch immer komplexere Aufgaben, wie das Visual Question Answering, bei dem eine Frage wie „Wie viele Personen sind im Bild?“ beantwortet werden soll. Auch Künstliche Neuronale Netze benötigen für diese Aufgaben eine Methode, um die Eingangsdaten nach ihrer Priorität für die aktuelle Aufgabe zu bewerten. [6]

Eine der komplexeren Aufgaben des *Machine Learning* stammen aus dem *Natural Language Processing* (NLP). Dieser Bereich beschäftigt sich unter anderem mit der Übersetzung und Generierung von Texten. Bis vor ca. vier Jahren wurden für *NLP* hauptsächlich *Recurrent Neural Networks* (RNNs) verwendet. Diese Architektur besitzt Schleifen, sodass der aktuelle Verarbeitungsschritt Informationen der vorangegangenen Schritte berücksichtigen kann. Ein Text wird dabei Wort für Wort verarbeitet. Sollte ein Satz übersetzt werden, wurde dieser in seine Wörter unterteilt. Das erste Wort wurde an das Netz gegeben und es hat die Übersetzung für das Wort ausgegeben. So wurde jedes Wort an das Netz übergeben und es hat mit Hilfe des aktuellen und dem Einfluss aller vorangegangenen Wörter das aktuelle Wort übersetzt. Dabei nimmt dieser Einfluss im Laufe der Schritte immer weiter ab (siehe Abbildung 6). Dieses Problem wird Vanishing-Gradient-Problem genannt. Je länger der Satz ist, desto weniger Einfluss haben Wörter, die am Anfang stehen auf das aktuelle Wort. [32]

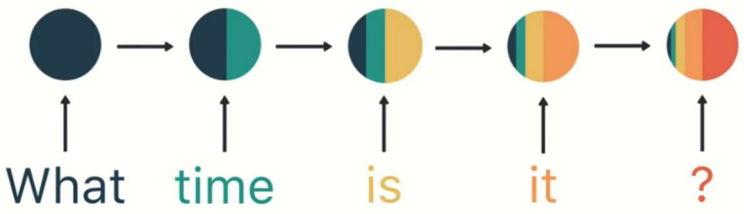

Abbildung 6: Visualisierung des Vanishing-Gradient-Problems (Quelle: https://youtu.be/LHXXI4-IEns)

In einem Satz haben die Wörter allerdings unterschiedlich starken Einfluss auf die Bedeutung. Die folgende Abbildung soll ein Beispiel einer Übersetzung veranschaulichen. Die Aufgabe besteht darin, den unteren englischen Satz in den oberen französischen Satz zu übersetzen. Ein RNN würde das aktuelle Wort „the“ bekommen und in seinem internen Status die verrechneten Informationen aus den drei vorangegangenen Wörtern berücksichtigen.

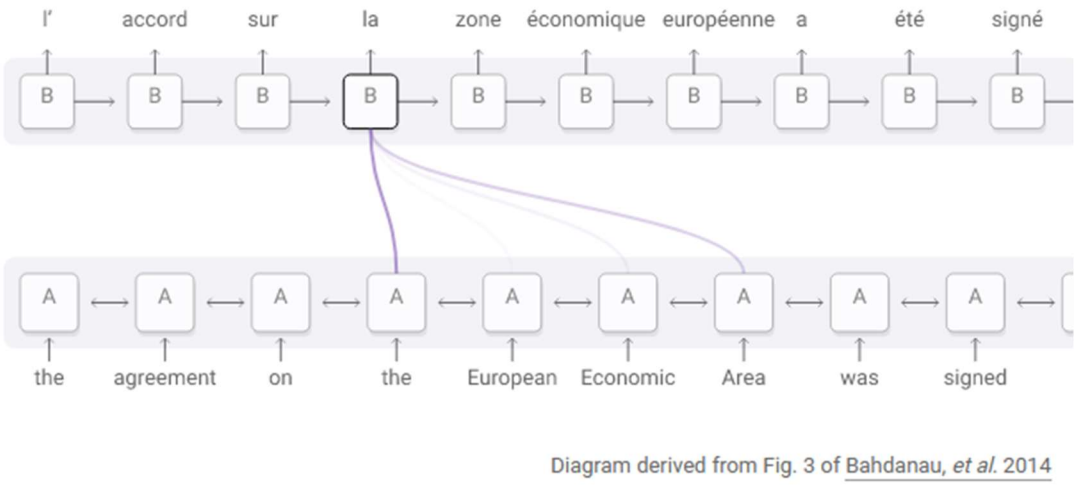

Abbildung 7: Visualisierung der Attention auf bestimmte Worte während der Übersetzung (Quelle: https://distill.pub/2016/augmented-rnns/)

Abbildung 7 zeigt dagegen ein *RNN*, das den *Attention-Mechanismus* verwendet. Die violetten Linien sollen die Aufmerksamkeit bei der Übersetzung des Wortes „la“ darstellen. Das Netz erhält wieder das Wort „the“, aber auch alle anderen Wörter des englischen Satzes. Dabei wird deutlich, dass zur Übersetzung des Wortes „la“ die Information entscheidend ist, auf welches Wort sich der Artikel bezieht, da im Französischen die Entscheidung über das Geschlecht des Subjektes das Wort „la“ bestimmt. Daher muss das Netz seine Aufmerksamkeit für die Übersetzung auf das aktuelle Wort, aber auch das Wort „Area“ richten, das viel später im Satz steht. [33]

Das Netz kann mit diesem Mechanismus nicht nur auswählen, welcher Input für einen bestimmten Schritt am wichtigsten ist, sondern auch den gesamten Satz betrachten. Mathematisch betrachtet ist ein KNN ein Funktionsaproximator, das heißt, es bildet eine beliebige Funktion ab, die Eingaben annimmt, diese verrechnet und ein Ergebnis ausgibt. Dabei werden die Inputs einzeln gewichtet (multipliziert) und in jedem Knoten zusammenaddiert. Je nach Architektur kann damit jede beliebige Funktion als Matrix-Multiplikation und Addition abgebildet werden. Der *Attention-Mechanismus* ist deshalb so effizient, da er eine Möglichkeit hinzufügt, die Inputs nicht nur untereinander zu addieren, sondern auch untereinander zu multiplizieren. Dadurch können Funktionen mit anderen (meist kleineren) Architekturen abgebildet werden. Attention ermöglicht es KNNs also, bestimmte Inputs für die Berechnung zu berücksichtigen oder diese zu ignorieren. [33] [34]

Es gibt verschiedene Arten von Attention für KNNs. Zunächst können verschiedene Funktionen verwendet werden, um die Werte der Attention-Matrix, also der Beziehungen zwischen den Inputs zu berechnen. Diese können z.B. auf der Cosinus-Funktion, gelernten Gewichten oder dem Skalar-

produkt zweier Input-Vektoren basieren. Nachdem diese Funktion festgelegt wurde, kann diese auf verschiedene Weise angewendet werden:

Hard Attention: ist eine diskrete Funktion über die Inputs, die für jeden Input besagt, ob dieser betrachtet oder ignoriert wird (siehe Abbildung 8). Die Attention-Matrix enthält die Werte 0 oder 1. Da die Funktion diskret ist, kann sie nicht differenziert werden. Das Training kann deshalb keine Backpropagation mit Gradient Descent verwenden und muss auf andere Methoden wie *Reinforcement Learning* zurückgreifen, was diese Art der Attention schwerer zu trainieren macht. [6]

Soft Attention: ist eine kontinuierliche Funktion über die gesamten Inputs (siehe Abbildung 8). Sie ist ein stochastischer Ansatz für die Attention. Es kann der Backpropagation-Algorithmus für das Training verwendet werden, bei sehr großen Inputs ist diese Art der Attention jedoch sehr rechenintensiv. [6]

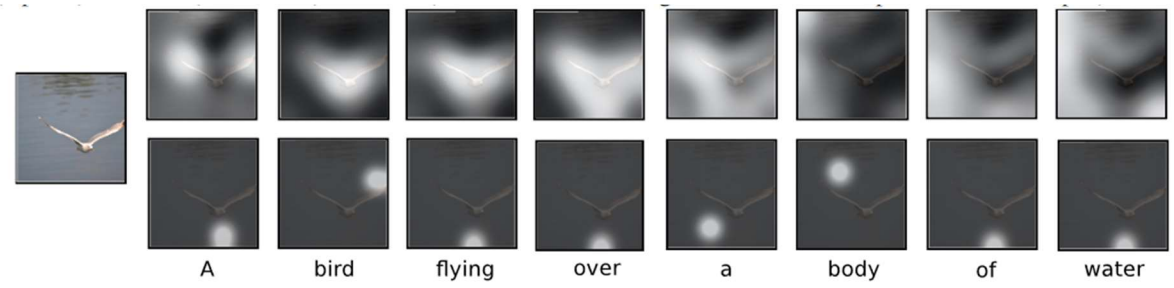

Abbildung 8: Soft Attention (oben) und Hard Attention (unten) (Quelle: [6])

Die Attention-Mechanismen können auch in globale und lokale Attention unterteilt werden. Globale Attention betrachtet alle Inputs. Dies kann bei sehr großen Inputs, wie Bildern oder langen Texten dazu führen, dass sehr viel Rechenleistung benötigt wird. Dabei ist es in manchen Fällen nicht nötig, den gesamten Input zu betrachten. Lokale Attention verbindet Hard Attention mit Soft Attention und betrachtet nur einen kleinen Teil der Inputs, aber mit kontinuierlichen Werten. [6] [33] [34]

In den bisher behandelten Beispielen wurde die Attention immer zwischen Input und Output gebildet. Der Input war ein Text oder ein Bild, der Output war die Übersetzung oder die Bildunterschrift. Eine weitere Art der Attention, die sogenannte Self-Attention, nimmt für beide Seiten denselben Input. Auf diese Weise können zum Beispiel die Beziehungen zwischen Wörtern eines Satzes abgebildet werden. Mit dieser Abbildung können Sätze in eine interne Repräsentation verrechnet werden, die für verschiedene weitere Verarbeitungsschritte geeignet ist. Eine Erweiterung der Self-Attention stellt die Multi-Head Attention dar. Dabei werden mehrere Self-Attention-Matrizen parallel erstellt, die jeweils einen anderen Aspekt der Aufmerksamkeit abbilden können. Anschließend werden diese Matrizen zusammengefügt. Im Jahr 2017 wurde eine Architektur für ein KNN basierend auf den

letzten beiden Arten der Attention in einem Paper vorgestellt. Das Paper „Attention Is All You Need“ zählt zu den einflussreichsten Beiträgen der letzten Jahre. Darin wird der sogenannte *Transformer* vorgestellt, der für *NLP* entwickelt, aber inzwischen auch für andere Bereiche wie *Computer Vision* eingesetzt wird. [31]

Im letzten Abschnitt wurden die zwei Probleme behandelt, die bei der Verarbeitung mehrerer Modalitäten (Arten von Eingangsdaten) entstehen. Das erste Problem bestand im *Alignment*, also der Synchronisation der Eingangsdaten. Der *Attention-Mechanismus* löst dieses Problem implizit, da er genau diesen Wert für die Zusammengehörigkeit zweier Inputs berechnet. Ein weiteres Beispiel für die Verwendung des *Attention-Mechanismus* zeigt Abbildung 9.

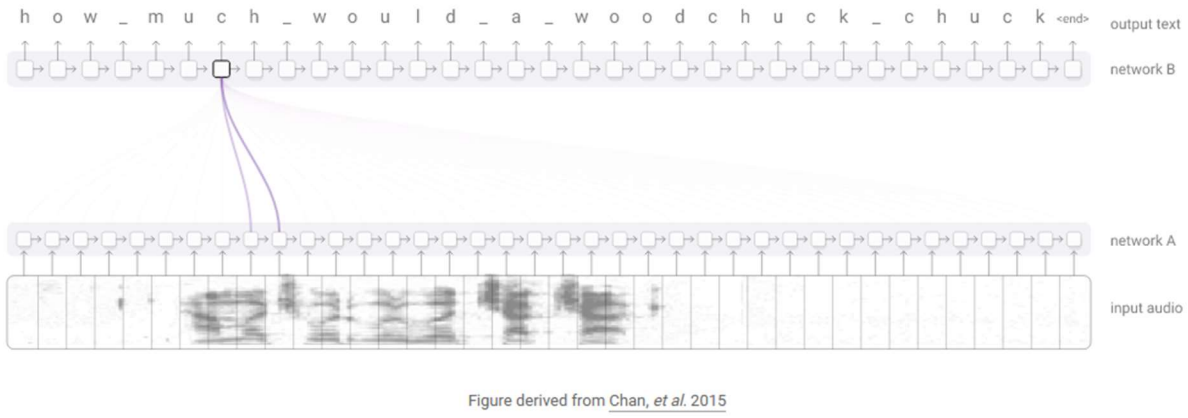

Abbildung 9: Visualisierung der Attention auf bestimmte Bereiche beim Erstellen eines Transkriptes (Quelle: https://distill.pub/2016/augmented-rnns/)

Das untere Netzwerk erzeugt Vektoren, die jeweils einen kleinen Teil einer Audiospur abbilden. Das obere Netzwerk erzeugt eine Transkription des gesprochenen Inhalts. Die violett dargestellten Linien zeigen die berechnete Zugehörigkeit mit Hilfe des *Attention-Mechanismus*. So wird nicht nur die Aufgabe gelöst, einen Text zu generieren, sondern intern auch der jeweilige Buchstabe zur Stelle der Audiospur zugeordnet. Diese Eigenschaft lässt sich jedoch noch weiterdenken. Selbst wenn die Audiospur und das Video synchronisiert sind, kann es wie bei der Übersetzung vorkommen, dass nicht nur der aktuelle Video-Frame und die aktuellen 100 Millisekunden der Audiospur für die Interpretation der Situation relevant sind, sondern eine Mimik oder Gestik, die erst zeitlich früher oder später auftritt.

Das zweite Problem des Lernens mit mehreren Modalitäten besteht in der *Fusion*, also der Erstellung einer gemeinsamen Repräsentation aus allen Eingangskanälen. Dieses Problem lässt sich mit zwei Attention-Mechanismen lösen.

Cross Attention: die Inputs einer Modalität werden verwendet, um die Attention für eine andere Modalität zu bestimmen. Ein Beispiel wäre die Erstellung einer gemeinsamen Repräsentation aus einem Bild und einer zugehörigen Beschreibung, wobei aus dem Satz die Attention-Matrix errechnet wird, die für das Bild verwendet wird. [35]

Co-Attention: die Inputs beider (oder mehrerer) Modaltäten werden verwendet, um eine gemeinsame Attention-Matrix zu erstellen. [36]

Der *Attention-Mechanismus* erlaubt es Künstlichen Neuronalen Netzen, bestimmte Eingangsdaten zu priorisieren oder zu ignorieren. Dadurch ist es nicht nur möglich, komplexe Aufgaben wie die Übersetzung von Texten sondern auch die Probleme der Fusion und des *Alignment* für die Verknüpfung mehrerer Eingangskanäle zu lösen.

Zusammenfassung des Kapitels

Im menschlichen Gehirn werden die Sinneseindrücke zunächst über die Sinnesorgane aufgenommen, anschließend zu einer komprimierten, internen Repräsentation zusammengefasst und schließlich gewichtet und ausgewählt mit der aktuellen Aufgabe verknüpft. Um eine übergeordnete Aufgabe lösen zu können und somit zu entscheiden, welche der aktuellen Sinneseindrücke am wichtigsten oder irrelevant sind, gibt es den *Attention-Mechanismus*. Damit ist es dem Menschen möglich, bestimmte unwichtige Informationen zu ignorieren und sich auf eine bestimmte Aufgabe zu konzentrieren. Der Mechanismus funktioniert allerdings auch in die andere Richtung, wenn die aktuelle Aufgabe von Sinneseindrücken (wie einem lauten Knall) unterbrochen wird.

Künstliche Neuronale Netze wurden aus der historischen Entwicklung zunächst nur für die Verarbeitung einzelner Modalitäten („Sinne“, Arten von Eingangsdaten) verwendet. Heutzutage werden immer komplexere Aufgaben mit mehreren Modalitäten gelöst. Auch die Netze arbeiten dabei sehr effektiv mit einem *Attention-Mechanismus*, der die Brücke zwischen den Eingangsdaten und der Aufgabe bildet. Der Mechanismus löst implizit die zwei großen Probleme des Lernens mit verschiedenen Modalitäten: *Alignment* und *Fusion*.

5 Konzept

In diesem Kapitel soll das Konzept für die Situationserkennung erstellt werden. Es wird genauer betrachtet, welche Ziele und konkreten Ergebnisse mit einer Situationserkennung erreicht werden sollen. Dazu wird als erstes der Tisch-Roboter ROS-E mit seinen Funktionen ausführlicher vorgestellt. Anschließend werden Personae definiert, um mit deren Hilfe konkrete Situationen zu konstruieren, die von der Situationserkennung behandelt werden sollen.

5.1 Der Tisch-Roboter ROS-E

Der Tisch-Roboter ROS-E wird seit 2019 an der TH-Wildau entwickelt. Die erste Version entstand im Rahmen einer Bachelorarbeit. Derzeit wird ROS-E von einem kleinen Team weiterentwickelt. Hier sollen die derzeit zur Verfügung stehenden Sensoren, Bauteile und Funktionen vorgestellt werden.

Zur Aufnahme von Informationen aus der Umwelt besitzt sie eine vertikal bewegliche Kamera und ein Mikrofon-Array. Der ganze Kopf ist horizontal drehbar (siehe Abbildung 10).

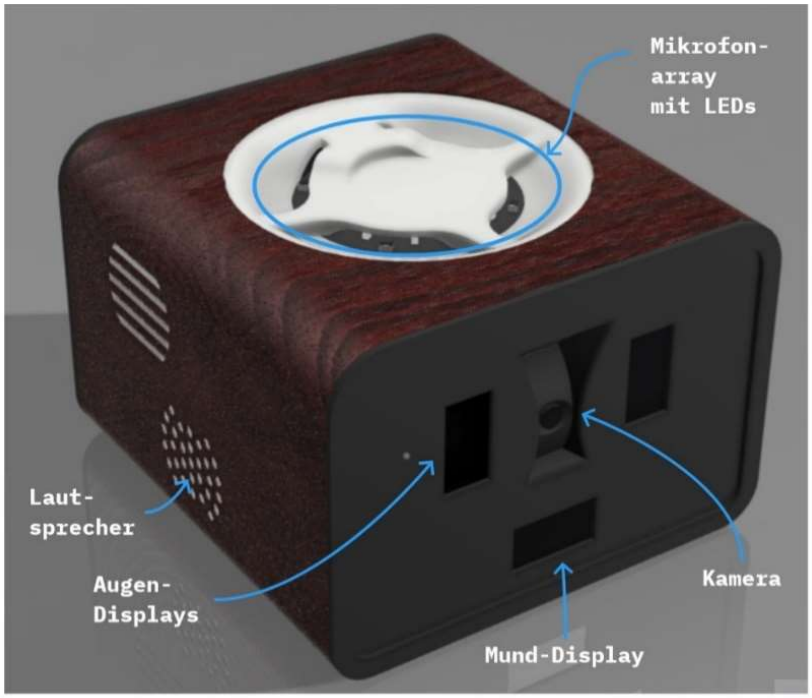

Abbildung 10: Komponenten von ROS-E (Quelle: RoboticLab der TH Wildau)

Mit diesen Bauteilen können grundlegende Funktionen für einen humanoiden Roboter implementiert werden. In der folgenden Tabelle sind die wichtigsten Grundfunktionen von ROS-E zusammengetragen.

Tabelle 1: Übersicht der Grundfunktionen von ROS-E

Funktion	**Beschreibung**
Augenform festlegen	Aus einer Auswahl vordefinierter Augenformen für verschiedene Emotionen oder Zustände kann eine Form ausgewählt werden.
Menschen angucken	Über die Kamera werden Gesichter erkannt und so eine Blickrichtung gewählt, in die die Augen bewegt werden.
Menschen mit Blick verfolgen	Die Blickrichtung wird kontinuierlich angepasst.
Mundform festlegen	Aus einer Auswahl vordefinierter Mundformen für verschiedene Emotionen oder Zustände kann eine Form ausgewählt werden.
Ton abspielen	Über die Lautsprecher kann eine Audio-Datei abgespielt werden.
Sprechen (Text-to-Speech)	Es kann ein Text angegeben werden, der mit einer künstlichen Stimme synthetisiert und über die Lautsprecher ausgegeben wird.
Geräusche aufnehmen	Mit dem Mikrofon-Array können Geräusche in einer Audio-Datei gespeichert werden.
Richtung des Geräusches feststellen	Über das Mikrofon-Array kann die Position der Geräuschquelle ermittelt werden.
Zuhören (Speech-to-Text)	Die aufgenommenen Geräusche können in Text umgewandelt werden.
Bild aufnehmen	Mit der Kamera kann ein Foto aufgenommen und gespeichert werden.
Video aufzeichnen	Mit der Kamera kann ein Video aufgenommen und gespeichert werden.
Kamera vertikal bewegen	Die Kamera kann vertikal bewegt werden.
360°-Drehung	Der ganze Kopf kann auf einem eingebauten Standfuß gedreht werden (derzeit eingeschränkt durch das Stromkabel).
Menschen mit Kopf/Kamera verfolgen	Die Kamera und der Kopf können so bewegt werden, dass Menschen automatisch verfolgt werden.
LEDs schalten	Die LEDs auf der Oberseite können ein- und ausgeschaltet werden.
Farbe der LEDs einstellen	Die Farbe der LEDs kann festgelegt werden.

Mit Hilfe dieser Grundfunktionen sind eine Reihe übergeordneter Funktionen umsetzbar. Über das Mikrofon und die Sprachausgabe besitzt ROS-E die Voraussetzungen für die Entwicklung zum Sprach-

assistenten. Damit sind Funktionen wie Erinnerungen, Terminverwaltung oder das Erstellen einer Einkaufsliste möglich. Über die Abspielfunktion für Ton könnten Inhalte von Musikdiensten wie z.B. Spotify oder eigene Geräusche und die Stimme von ROS-E ausgegeben werden. Die Lieder könnten dabei entweder über die Spracheingabe ausgewählt werden oder ROS-E könnte selbst entscheiden, welches Lied oder welche Playlist gespielt wird.

Um ein besseres Bild davon zu haben, welche Situationen mit und um ROS-E und andere humanoide Roboter auftreten können, sollen im nächsten Abschnitt Personae erstellt werden. Mit ihnen werden anschließend verschiedene Situationen konstruiert. In den Personae werden dabei verschiedene Zielgruppen, Persönlichkeiten und Vorlieben abgebildet.

5.2 Personas/Personae

Das Erstellen einer sogenannten Persona ist eine Methode aus der Produktentwicklung und dem Marketing, die auch für die Software-Entwicklung übernommen wurde. Das Ziel dieser Methode ist es, eine konkrete, fiktive Person als Repräsentant einer Zielgruppe zu erstellen. Dieser Persona werden Hintergrundinformationen wie Familie und Persönlichkeit gegeben, um besser einschätzen zu können, welches Kaufverhalten oder Nutzungsverhalten einer Anwendung die Persona hätte.

Im Rahmen dieser Arbeit wurden fünf Personae entwickelt, die eine Auswahl der Altersgruppen und Lebenssituationen von späteren Besitzern eines Tisch-Roboters abbilden.

Volker Oppel repräsentiert die Gruppe der älteren, fitten Menschen, die sich für Technik begeistern (siehe Abbildung 11). Diese Menschen sind durchaus aufgeschlossen, mit technischen Geräten zu arbeiten. Sie können grundlegende Aufgaben lösen, sind aber von komplexen Anwendungen oder vielen neuen Informationen häufig überfordert. Von ihnen werden Funktionen benutzt, die sie bereits kennen und wissen, wie sie diese bedienen müssen. Sobald sich Abläufe verändern, können sie schnell frustriert sein.

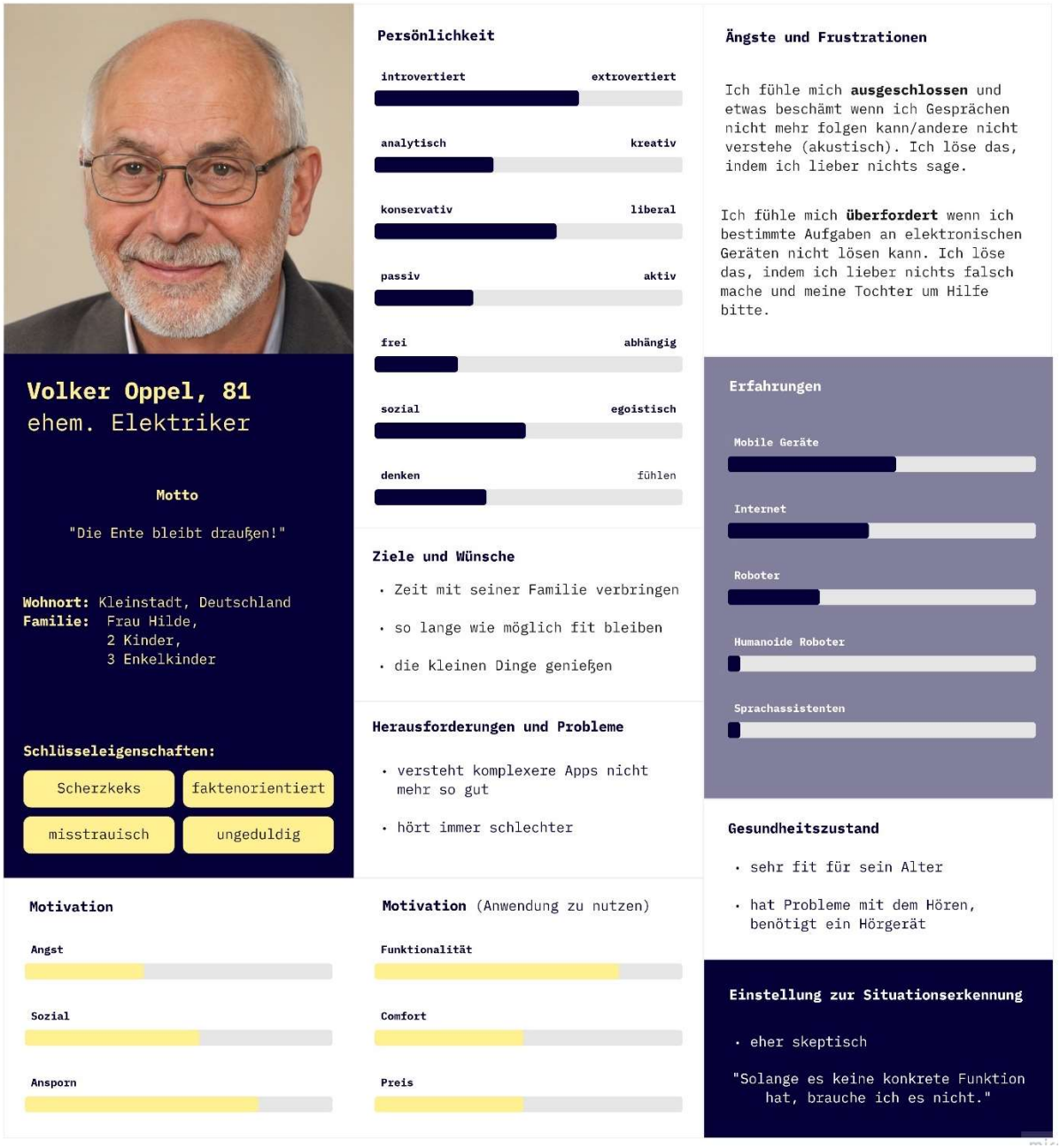

Abbildung 11: Persona Volker Oppel (Quelle: eigene Abbildung, Portrait-Quelle: https://thispersondoesnotexist.com/)

Elfriede Schnoor stellt die Gruppe der älteren, gesundheitlich eingeschränkten Menschen dar, die sich nicht mit Technik beschäftigen (siehe Abbildung 12). Moderne Telefone können diese Menschen bereits überfordern, vor allem, da die Bedienungsanleitungen nicht gut erklärt sind oder es zu anstrengend ist, diese bei jeder Bedienung zu benötigen. Sich Bedienungsabläufe zu merken, ist sehr schwer für diese Personen. Sie sind häufig allein zu Hause und verlassen das Haus nur selten.

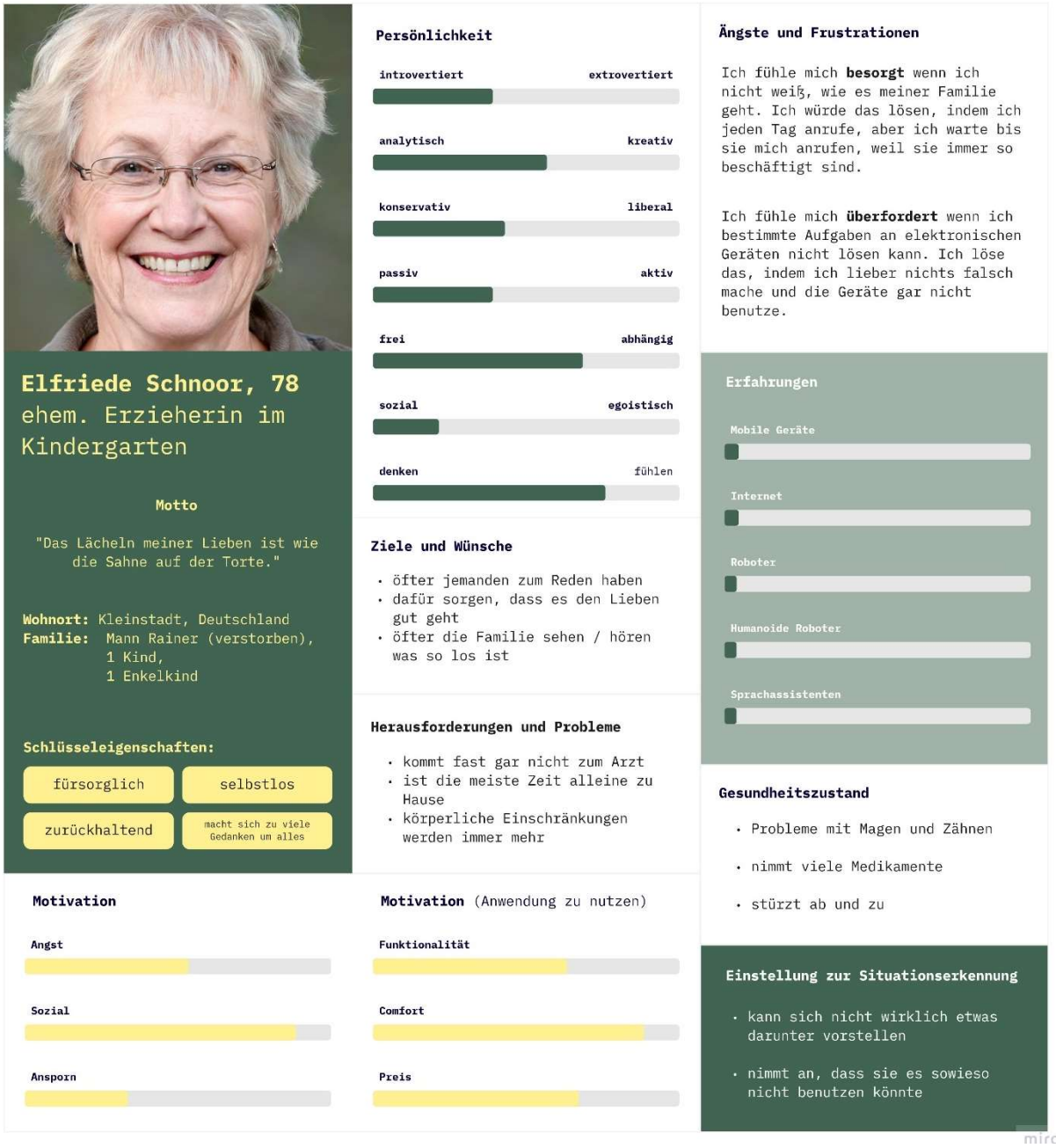

Abbildung 12: Persona Elfride Schnoor (Quelle: eigene Abbildung, Portrait-Quelle: https://thispersondoesnotexist.com/)

Andrea Heuberger bildet einerseits die Gruppe der Eltern ab, die ihre Kinder mit dem Einsatz eines Roboters vor allem beim Lernen unterstützen möchte (siehe Abbildung 13). Andererseits vertritt sie die Gruppe der Betreuungspersonen von gesundheitlich eingeschränkten Menschen. Diese sind oft erfreut über Hilfsmittel, mit denen sie den Patienten das Leben erleichtern oder diese für etwas begeistern können. Sie sind aber auch kritisch gegenüber der Funktionalität und dem Datenschutz.

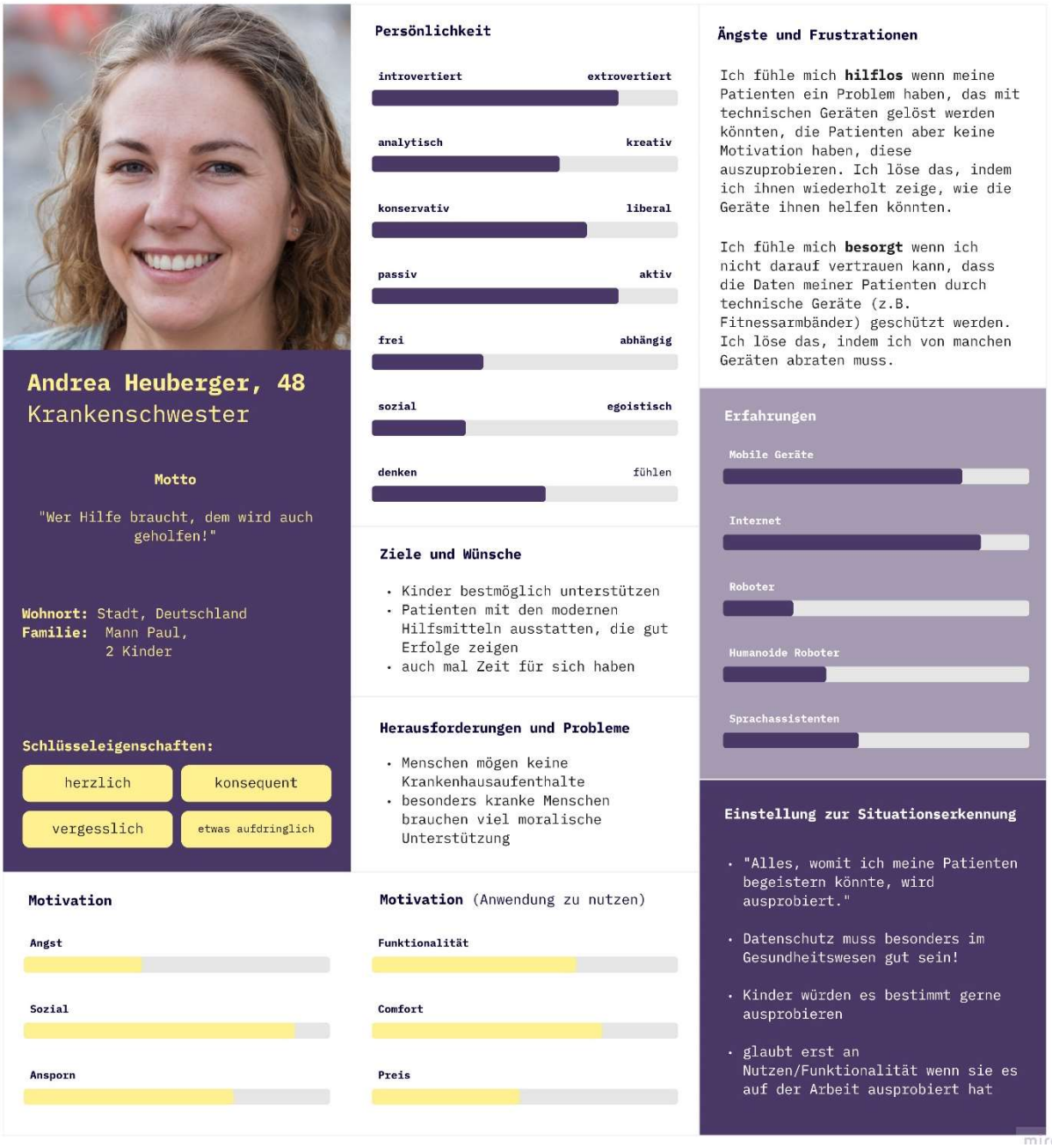

Abbildung 13: Persona Andrea Heuberger (Quelle: eigene Abbildung, Portrait-Quelle: https://thispersondoesnotexist.com/)

Thore Heuberger beschreibt die Gruppe der Kinder, die von Robotern begeistert sind und mit diesen für das Lernen begeistert werden können (siehe Abbildung 14). In der Schule kommen die Kinder dieser Gruppe nicht mit den angewendeten Lehrmethoden zurecht. Die Lehrer haben oft nicht die Zeit, um auf alle Kinder einzugehen. Roboter sind geduldige Lernpartner, vor denen die Kinder dieser Gruppe keine Angst haben, Fehler zu machen.

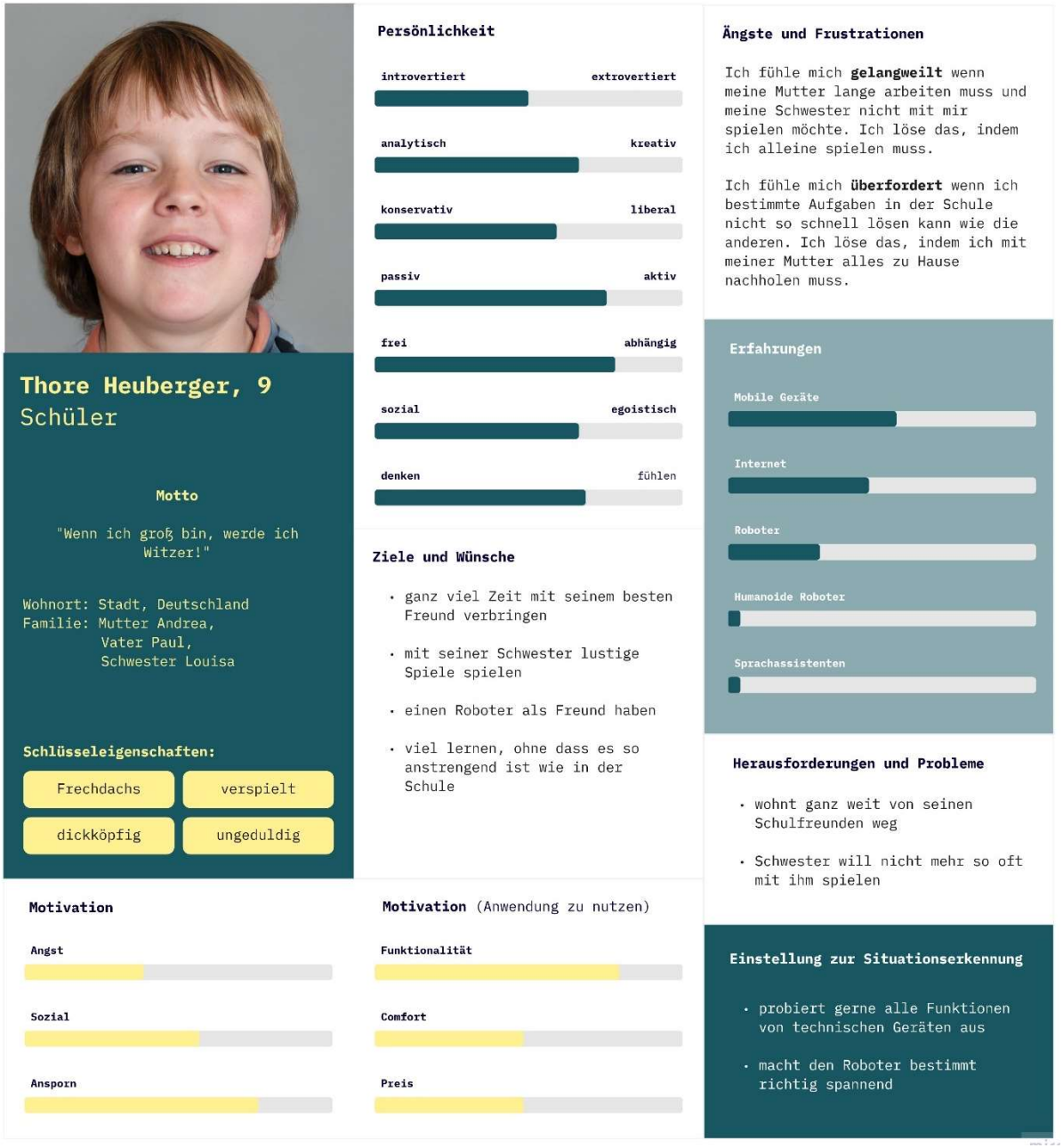

Abbildung 14: Persona Thore Heuberger (Quelle: eigene Abbildung, Portrait-Quelle: https://thispersondoesnotexist.com/)

Elias Flemming repräsentiert die Gruppe der Entwickler für Anwendungen auf humanoiden Robotern (siehe Abbildung 15). Die Entwickler kennen sich gut mit technischen Geräten aus, haben aber häufig noch keine direkte Erfahrung mit humanoiden Robotern. Sie müssen nicht genau wissen, wie die Hardware im Einzelnen funktioniert, benötigen aber eine gute Dokumentation der Schnittstellen.

Abbildung 15: Persona Elias Flemming (Quelle: eigene Abbildung, Portrait-Quelle: https://thispersondoesnotexist.com/)

Anhand dieser Personae werden im folgenden Kapitel mögliche Situationen im Umgang mit ROS-E gesammelt.

5.3 Erfassen von Situationen durch ROS-E

Um die Frage beantworten zu können, welche Situationen erkannt werden sollen, müssten zunächst mögliche Situationen identifiziert werden. Die folgenden Beispiele stellen eine Auswahl möglicher Situationen dar, die in dieser Arbeit betrachtet werden sollen und haben keinen Anspruch auf Vollständigkeit. Die erste Version der Situationserkennung richtet sich vor allem an den Einsatz von ROS-E auf einem Schreibtisch, Küchentisch oder im Wohnzimmer und für Veranstaltungen ähnlich einer Messe.

Die nachfolgende Tabelle enthält Situationen, die in direkter Interaktion zwischen Mensch und Roboter stattfinden.

Tabelle 2: Beispiele für Situationen in der direkten Interaktion zwischen Mensch und Roboter

ID	Situation	Aktionen der Persona
SI1	Jemand begrüßt den Roboter.	• Volker bleibt ernst, aber überwindet sich, eine Begrüßung zu versuchen. • Elfriede lächelt freundlich und grüßt höflich, aber etwas verunsichert. • Thore winkt und grinst begeistert. • Andrea ruft im Vorbeigehen eine Begrüßung zu. • Elias spricht ganz laut und deutlich seine Begrüßung.
SI2	Jemand interagiert zum ersten Mal mit einem Roboter.	• Volker möchte nichts sagen und nicht von einer Kamera erfasst werden, weil ihm das das Gefühl gibt, abgehört zu werden und er nicht weiß, was mit den Daten passiert. • Elfriede traut sich nicht, mit dem Roboter zu sprechen, weil sie es seltsam findet. • Thore springt aufgeregt hin und her und steht ganz dicht vor dem Roboter, weil er sofort ausprobieren möchte, was er alles kann. • Andrea sitzt schweigend vor dem Roboter, weil sie nicht weiß, wo sie anfangen soll. • Elias testet sofort die Funktion, nach der Uhrzeit zu fragen.
SI3	Jemand möchte die aktuelle Funktion abbrechen.	• Volker schüttelt den Kopf und entfernt sich vom Roboter, während er „Nein“ sagt. • Elfriede sagt ganz aufgeregt „Abbrechen, bitte“. • Thore ruft „Aufhören, Ich will nicht mehr!“. • Andrea sagt „Stopp“ und macht dabei eine wegwischende Handbewegung.
SI4	Jemand möchte einer gestellten Frage zustimmen oder sie ablehnen.	• Volker schüttelt den Kopf und entfernt sich vom Roboter, während er „Nein“ sagt. (beabsichtigte Dopplung zu SR3) • Elfriede lächelt und sagt „Ja“. • Thore nickt begeistert. • Andrea ruft aus dem nächsten Zimmer „NEIN!“.

ID	Situation	Aktionen der Persona
		• Elias zuckt mit den Schultern.
SI5	Jemand hat die letzte Aussage des Roboters nicht verstanden.	• Volker sitzt vor dem Roboter und rührt sich nicht mehr, weil er nicht weiß, was dann passiert. • Elfriede fragt „Wie bitte?“. • Thore fordert „Sag nochmal!“. • Andrea fragt „Kannst Du das nochmal wiederholen?“. • Elias fragt „Was?“.
SI6	Der Akku des Roboters ist fast leer (und eine Benachrichtigung wird ausgegeben).	• Elfriede greift sofort nach dem Roboter und möchte ihn zur Ladestation bringen. • Thore ist gerade mitten in einem Spiel und ignoriert die Nachricht. • Andrea antwortet „Danke für die Erinnerung“.
SI7	Jemand verabschiedet sich vom Roboter.	• Volker sagt erleichtert „Auf Wiedersehen“ und macht sich schnell aus dem Staub. • Elfriede ist erstaunt vom Roboter, aber auch erleichtert, dass sie jetzt nicht mehr interagieren muss. Sie verabschiedet sich höflich. • Thore klingt ganz traurig, weil er sich gar nicht verabschieden möchte.

Um die Autonomie eines Roboters zu erhöhen, ist für diesen nicht nur relevant, welche Situationen in der direkten Interaktion auftreten. Es könnte ebenso Situationen ohne eine direkte Interaktion geben, in denen der Roboter seine Reaktion anpassen könnte. Diese sollen in der folgenden Tabelle beispielhaft aufgeführt werden.

Tabelle 3: Beispiele für Situationen in der Umgebung von Robotern

ID	Situation	Aktionen der Person
SU1	Die Rauchmelder schlagen Alarm.	• Volker beschwert sich über die Technik, die ständig Fehler macht. • Elfriede läuft aufgeregt umher und weiß nicht, was sie tun soll. • Thore versteckt sich unter dem Tisch und hält sich die Ohren zu. • Andrea sucht nach dem Telefon und möchte die Feuerwehr rufen. • Elias schließt alle Fenster und verlässt den Raum.
SU2	Jemand sucht einen Gegenstand.	• Volker sucht sein Hörgerät. • Elfriede sucht ihre Brille, die sie auf die Stirn geschoben hat. • Thore sucht seine Lieblings-Legofigur.

ID	Situation	Aktionen der Person
		• Andrea such ihren Autoschlüssel. • Elias sucht sein Ladekabel.
SU3	Jemand gießt Tee auf.	• Elfriede gießt ihren geliebten grünen Tee auf, der genau 30 Sekunden ziehen muss. • Andrea beginnt, den Tisch zu decken, wobei sie den Tee ganz vergisst. • Elias setzt sich an seinen Computer und ist sofort wieder vertieft.
SU4	Der Herd ist noch an, obwohl niemand mehr kocht.	• Volker sitzt gemütlich am Tisch und genießt sein zur Abwechslung sehr Fleisch-lastiges Essen. • Elfriede ist damit beschäftigt, den Tisch zu decken. • Thore hat sich zum ersten Mal selber Eierkuchen gemacht und vergessen, den Herd aus zu machen. • Elias sitzt vor seinem Computer und verschlingt sein Mittag.
SU5	Jemand erhält einen Anruf.	• Volker sieht gerade Skispringen im Fernsehen und greift sofort nach der Fernbedienung. • Andrea sitzt auf dem Sofa und hört Musik.
SU6	Jemand setzt sich auf ein Sofa.	• Volker greift nach der Fernsehzeitschrift. • Elfriede greift nach einem Buch. • Thore spielt mit seinen Superhelden-Figuren. • Andrea möchte das Radio mit der Fernbedienung einschalten.
SU7	Jemand ist eingeschlafen, während der Fernseher, Musik oder ein Hörbuch noch läuft.	• Elfriede ist bei ihrer Mittagsruhe vor dem Fernseher eingeschlafen. • Thore ist beim Hören seines Lieblingshörbuches endlich eingeschlafen. • Andrea hat auf dem Sofa gemütlich Musik gehört und ist dabei eingeschlafen. • Elias liegt im Bett und neben ihm läuft noch ein YouTube-Video auf dem Laptop.
SU8	Jemand hat Schmerzen.	• Volker hat eine krumme Haltung und hält sich den Rücken. • Elfriede stößt sich den Ellenbogen und stöhnt. • Thore hat sich das Knie aufgeschlagen, läuft blutend umher und heult. • Andrea sitzt am Schreibtisch und reibt ihr Handgelenk, weil ihr Mausarm wieder wehtut. • Elias massiert sich die Schläfen, weil er Kopfschmerzen hat.

Anhand dieser wenigen Beispiele wird bereits deutlich, wie unterschiedlich die Eingangsdaten (Video- und Audiodaten) für eine bestimmte Situation sein können. Die Situationserkennung für

humanoide Roboter könnte ohne das Ableiten einer Reaktion als eine Art Protokoll verwendet werden, mit dem bestimmte Ereignisse wie Nahrungsaufnahme, Körperpflege oder auch Arbeitszeiten, Sport und Unterhaltung dokumentiert werden. Aber auch diese Dokumentation hat keine weitere Funktion, wenn die erkannte Situation nicht weiter ausgewertet wird. Im folgenden Abschnitt sollen daher mögliche Reaktionen des Roboters für einige Beispiele dargestellt werden.

5.4 Reaktionen von ROS-E auf Situationen

Bei der Aufgabe, Reaktionen für die im letzten Abschnitt aufgeführten Situationen zu konstruieren, traten zwei Fälle auf: entweder führt eine Situation zu mehreren verschiedenen Reaktionen oder eine Reaktion könnte in mehreren Situationen hilfreich sein. Die folgende Tabelle enthält die interessantesten Ergebnisse der konstruierten Situationen und Reaktionen.

Tabelle 4: Beispiele für Reaktionen eines Roboters auf bestimmte Situationen

ID	Situation	Reaktion des Roboters
SI1	• Jemand begrüßt den Roboter.	Der Roboter grüßt zurück.
SI2	• Volker möchte nichts sagen und nicht von einer Kamera erfasst werden, weil ihm so etwas das Gefühl gibt, abgehört zu werden und er nicht weiß, was mit den Daten passiert. • Elfriede traut sich nicht, mit dem Roboter zu sprechen, weil sie es seltsam findet. • Thore springt aufgeregt hin und her und steht ganz dicht vor dem Roboter, weil er sofort ausprobieren möchte, was er alles kann. • Andrea sitzt schweigend vor dem Roboter, weil sie nicht weiß, wo sie anfangen soll.	Der Roboter schlägt ein Kennlern-Spiel vor.
SI3	• Volker schüttelt den Kopf und entfernt sich vom Roboter, während er „Nein“ sagt. • Elfriede sagt ganz aufgeregt „Abbrechen, bitte“. • Thore ruft „Aufhören, Ich will nicht mehr!“. • Andrea sagt „Stopp“ und macht dabei eine wegwischende Handbewegung.	Der Roboter bricht den aktuellen Befehl ab.
SI4	• Volker schüttelt den Kopf und entfernt sich vom Roboter, während er „Nein“ sagt. (mit Absicht doppelt)	Der Roboter führt den aktuellen Befehl nicht aus.

ID	Situation	Reaktion des Roboters
	• Andrea ruft aus dem nächsten Zimmer „NEIN!“. • Elias zuckt mit den Schultern.	
	• Elfriede lächelt und sagt „Ja“. • Thore nickt begeistert.	Der Roboter führt den aktuellen Befehl aus.
SI5	• Volker sitzt vor dem Roboter und rührt sich nicht mehr, weil er nicht weiß, was dann passiert. • Elfriede fragt: „Wie bitte?“. • Thore fordert: „Sag nochmal!“. • Andrea fragt: „Kannst Du das nochmal wiederholen?“. • Elias fragt „Was?“.	Der Roboter wiederholt den letzten Satz.
SI6	• Elfriede greift sofort nach dem Roboter und möchte ihn zur Ladestation bringen.	Der Roboter bedankt sich.
	• Thore ist gerade mitten in einem Spiel und ignoriert die Nachricht.	Der Roboter macht einen müden Gesichtsausdruck.
	• Andrea antwortet: „Danke für die Erinnerung“.	Der Roboter lächelt.
SI7	• Volker sagt erleichtert „Auf Wiedersehen“ und macht sich schnell aus dem Staub.	Der Roboter erwidert die Verabschiedung und zeigt einen verwunderten Gesichtsausdruck.
	• Elfriede ist erstaunt vom Roboter, aber auch erleichtert, dass sie jetzt nicht mehr interagieren muss. Sie verabschiedet sich höflich.	Der Roboter erwidert die Verabschiedung und lächelt.
	• Thore klingt ganz traurig, weil er sich gar nicht verabschieden möchte.	Der Roboter erwidert die Verabschiedung und zeigt auch einen traurigen Gesichtsausdruck.
SU9	• Volker hat eine krumme Haltung und hält sich den Rücken. • Elfriede stößt sich den Ellenbogen und stöhnt. • Andrea sitzt am Schreibtisch und reibt ihr Handgelenk, weil ihr Mausarm wieder wehtut. • Elias massiert sich die Schläfen, weil er Kopfschmerzen hat.	Der Roboter schaut mitleidig/besorgt und fragt, wie schlimm die Schmerzen sind.
	• Thore hat sich das Knie aufgeschlagen, läuft blutend umher und heult.	Der Roboter sendet eine Benachrichtigung an den eingestellten Notfall-Kontakt.

Nachdem einige Beispiele für Situationen vorgestellt wurden, wird deutlich, dass es eine sehr aufwändige, wenn nicht unmögliche Aufgabe wäre, alle möglichen Situationen zu identifizieren, um diesen eine (oder mehrere) Reaktionen zuordnen zu können. Der Ansatz des *Machine Learning* soll genau dieses Problem lösen: anstatt als Entwickler ein Modell zu finden und einen Algorithmus festlegen zu müssen, sollen die Zusammenhänge zwischen Eingangsdaten (Sinneseindrücken) und Reaktionen durch ein Künstliches Neuronales Netz gelernt werden. Damit wird die Frage, auf welche Weise ein Roboter auf die Situationen reagieren soll, nicht mehr nur von den Entwicklern, sondern vom Roboter beantwortet. Auch der Nutzer könnte Einfluss auf die vom Roboter gewählte Reaktion haben, wenn das Netz kontinuierlich weiter lernt und sich an die Gewohnheiten des Nutzers anpasst.

An dieser Stelle entscheidet sich auch, wie hoch der Grad der Autonomie des Roboters sein soll. Darf er lediglich auf direkte Befehle reagieren? Kann er von selbst Reaktionen vorschlagen, benötigt aber jedes Mal die Zustimmung der Nutzer? Oder darf er selbstständig Entscheidungen treffen?

Im folgenden Abschnitt sollen die verschiedenen Möglichkeiten der Umsetzung einer Situationserkennung diskutiert und das Konzept für die Umsetzung der Situationserkennung im Rahmen dieser Arbeit vorgestellt werden.

5.5 Möglichkeiten der Auswertung des Ergebnisses

Eine Situationserkennung kann für einen Roboter ein wichtiger Baustein sein, um eine seiner grundlegenden Aufgaben zu lösen: den Menschen zu unterstützten. Um einen Menschen zu unterstützen, stehen Robotern eine Anzahl an Funktionen zur Verfügung. Eine Situationserkennung sollte dem Roboter ermöglichen, die bestmögliche Reaktion auf die aktuelle Situation auszuwählen. Wer legt aber fest, was die beste Reaktion ist? Einige mögliche Antworten auf diese Frage sollen in diesem Abschnitt diskutiert werden.

5.5.1 Klassifizierung einer festgelegten Anzahl an Situationen

Der „klassische“ Ansatz würde darin bestehen, eine bestimmte Anzahl an Situationen zu definieren, die das Netz erkennen kann. Solche Situationen könnten „Begrüßung“, „Verabschiedung“, „Überforderung“ oder „Mensch ist aufmerksam“ sein. Diese Art der Aufgabe wird im *Machine Learning* als Klassifizierung bezeichnet. Das KNN bekommt eine Eingabe und ordnet diese einer der festgelegten Situationen (sogenannte Klassen) zu. Diese Ergebnisse würden anschließend allen Anwendungen auf dem Roboter zur Verfügung gestellt werden.

Dieser Ansatz eröffnet allerdings eine Reihe an Fragen, die beantwortet werden müssen. Die erste Frage besteht darin, wer die Auswahl der Situationen festlegt. Wenn diese im Rahmen dieser Arbeit vorgegeben werden, ohne zu wissen, welche Situationen später wirklich auftreten oder welche die Reaktion des Roboters oder der Anwendungen überhaupt beeinflussen, würde die Situationserkennung am Ende vermutlich nicht genutzt werden. Angefangen beim Beispiel der Begrüßung stellt sich die Frage, ob der Roboter für seine Reaktion nicht eine zusätzliche Unterteilung in „gut gelaunte Begrüßung“ oder „unsichere Begrüßung“ benötigen würde.

Angenommen, es würde eine Sammlung an Situationen definiert werden können. Im nächsten Schritt würde dann die Frage auftreten, wer die Einteilung der Beispieldaten für das Training vornimmt. Diese Einteilung gibt dem Netz vor, ob es eine Situation „richtig“ erkannt hat? Die Einschätzung einer Situation kann von Kultur zu Kultur und sogar von Mensch zu Mensch unterschiedlich sein. Besonders wenn der Roboter im Zusammenhang mit Patienten eingesetzt wird, könnte deren Einschätzung einer Situation sich stark von der eines „normalen“ Alltags unterscheiden. Welche Partei diese Einteilung der Trainingsdaten vornimmt, wird in den meisten Fällen organisatorisch festgelegt werden müssen. Die Entwickler haben vielleicht keine Zeit, um tausende Datensätze einzuteilen, also werden Außenstehende dafür gefunden. Damit ein Netz jedoch wirklich lernen kann, ob die Situation richtig erkannt wurde, müsste theoretisch die Person, die in der jeweiligen Situation auch anwesend war, das *Label* festlegen. Dieser Ansatz der Klassifizierung wird in der Praxis häufig von Firmen eingesetzt, die damit feste Regeln und Abläufe unterstützten. Zum Beispiel die korrekte Belegung von Pizza bewerten, die immer sieben gleichmäßig verteilte Salami-Scheiben besitzen muss. Eine Situationserkennung schränkt dieser Ansatz jedoch sehr stark ein, da die Regeln der menschlichen Kommunikation sehr viel komplexer sind. Daher sollen im Folgenden alternative Ansätze untersucht werden.

5.5.2 Bereitstellen eines Vektors für weitere Verarbeitung

In Kapitel 4 wurden die einzelnen Komponenten für die Verarbeitung mehrerer Eingangskanäle zu einer internen Repräsentation der Situation im menschlichen Gehirn und mit Künstlichen neuronalen Netzen beschrieben. Ein Künstliches Neuronales Netz nimmt dabei zum Beispiel die einzelnen Bilder (Frames) eines Videos und die codierten Daten einer Audiodatei auf, erkennt zuerst einzelne Linien und Muster und verarbeitet die Daten zu einem Vektor, der die aktuelle Situation abbildet. Dieser Vektor wird nachfolgend als *Representation-Vector* (RV) bezeichnet. Ähnlich wie im menschlichen Gehirn das *FPAN* diese abstrakte, komprimierte Abbildung der aktuellen Situation verwendet, um eine übergeordnete Aufgabe (z.B. ein Buch lesen) zu erfüllen, könnte dieser Vektor von anderen Anwendungen auf dem Roboter verwendet werden.

Mit diesem Vektor könnten verschiedene Aufgaben durch andere Netze oder Algorithmen gelöst werden, die auf der „Wahrnehmung“ der Umwelt basieren. Ein Beispiel dafür wäre die Auswertung von Sprachbefehlen, die derzeit bereits auf ROS-E verfügbar ist. Diese arbeitet mit einem Netz, das einen solchen Vektor aus dem Text des gesprochenen Befehls erstellt. Dazu wird die Sprache des Nutzers aufgezeichnet und mit der Speech-to-Text Funktion umgewandelt. Aus diesem Text wird ein Vektor erstellt, der anschließend über einen *k-Nächste Nachbarn Algorithmus* dem Befehl zugeordnet, dem der Vektor am ähnlichsten ist. Für die weitere Verarbeitung eines solchen Vektors ist also nicht unbedingt ein weiteres Künstliches Neuronales Netz notwendig. Auch die Auswertung des Sprachbefehls könnte mit dem neuen Vektor der Situationserkennung arbeiten. So wäre es denkbar, dass zum Beispiel in Zukunft auch Nicken und Kopfschütteln für die Steuerung von Befehlen ausgewertet werden könnten.

Ein wichtiger Punkt, der für diesen Ansatz spricht, ist die Beschaffung von Trainingsdaten. Der erste Teil des Netzes könnte anhand „fremder“ Videos, die nicht zwangsläufig eine Interaktion mit Robotern enthalten, vortrainiert werden. In diesem sogenannten *Pre-Training* werden das implizite *Alignment* und die Erkennung von grundlegenden Strukturen, wie Menschen oder Objekten gelernt. Später kann das Netz mit Videos, die direkt mit ROS-E aufgenommen wurden, weiter trainiert werden, um das Netz auf den Anwendungsfall anzupassen und die Genauigkeit zu verbessern. Für das *Pre-Training* könnten also Videos aus Datensets verwendet werden, die über das Internet beschafft werden können.

Dieser Ansatz wurde als minimale Variante gewählt, falls die Zeit für einen weiteren Ansatz nicht ausgereicht hätte. Allerdings muss dieser Ansatz noch erweitert werden, da ohne eine Verarbeitung des *Representation-Vectors* keine Aussage getroffen werden könnte, ob dieser Ansatz erfolgreich war. Die Qualität des Vektors für sich ist schwer einzuschätzen.

5.5.3 Ableiten einer Reaktion des Roboters

Eine Alternative zur Klassifikation mit Künstlichen Neuronalen Netzen stellt das sogenannte *Reinforcement Learning* dar. Dabei wird die Aufgabe des Netzes anders definiert, als bei der Klassifizierung aus dem ersten Ansatz. Beim *Reinforcement Learning* gibt es eine Auswahl möglicher Aktionen als Ergebnis der Vorhersage des Netzes. Die Aufgabe des Netzes ist es, aufgrund der aktuellen Eingabedaten mit Hilfe einer Verhaltensvorschrift die bestmögliche Aktion auszuwählen. Dieser Ansatz besitzt im Vergleich zur Klassifikation zwei Eigenschaften, die für eine Situationserkennung sehr wichtig sein können. Die erste Eigenschaft ist die Art auf die die „bestmögliche“ Reaktion vorgegeben wird. Dies geschieht während der Trainingsphase durch ein Belohnungssystem. Dadurch wird es möglich, dass die Benutzer dem Roboter selbst vorgeben können was

„richtig“ und „falsch“ für sie bedeutet. Die zweite Eigenschaft besteht darin, dass das Netz nicht immer dieselbe Reaktion auf eine Situation zeigen muss. Die Auswahl der nächsten Aktion kann mit einer kleinen Wahrscheinlichkeit eine zufällige Aktion auswählen, um zu ermitteln, ob es nicht eine Aktion gibt, die noch besser geeignet wäre als die derzeit bestbewertete Aktion. Dadurch besitzt der Roboter eine Art Neugier und kann einmal gelernte Verhaltensweisen auch wieder „umlernen“. [37]

Dieser Ansatz verwendet den *Representation-Vector* aus dem vorherigen Ansatz als Grundlage für die Entscheidung, welche Aktion als nächstes ausgeführt werden soll. Mit diesem Ansatz könnte die Vorverarbeitung der Daten mit einer übergeordneten Aufgabe verknüpft werden, ähnlich wie es im menschlichen Gehirn durch die Struktur des *FPAN* geschieht.

Für das Training müsste eine Trainings- und Testumgebung eigerichtet werden, mit der die Benutzer die Belohnungen für die Reaktionen des Roboters vergeben können. Dabei könnten auch zeitgleich die Trainingsdaten der direkten Interaktion mit ROS-E gesammelt werden, die für ein vollständiges Training des gesamten Netzes benötigt werden. Das Konzept für die Situationserkennung und das Künstliche Neuronale Netz wird im nächsten Abschnitt vorgestellt.

5.6 Die Grundstruktur des Netzes

Die Situationserkennung, die im Rahmen dieser Arbeit entwickelt werden soll, kann in zwei Teilaufgaben geteilt werden (siehe Abbildung 16). Die erste Teilaufgabe besteht darin, die Eingangsdaten (Video- und Audiodaten) zu einer internen Repräsentation, dem *Representation-Vector* zu verarbeiten. Dieser Vektor kann von anderen Anwendungen auf dem Roboter verwendet werden, beispielsweise um darauf selbst eine Klassifizierung von selbst definierten Situationen durchzuführen. Oder mit anderen Algorithmen oder Künstlichen Neuronalen Netzen Aufgaben zu definieren, die auf den gewählten Eingangsdaten basieren.

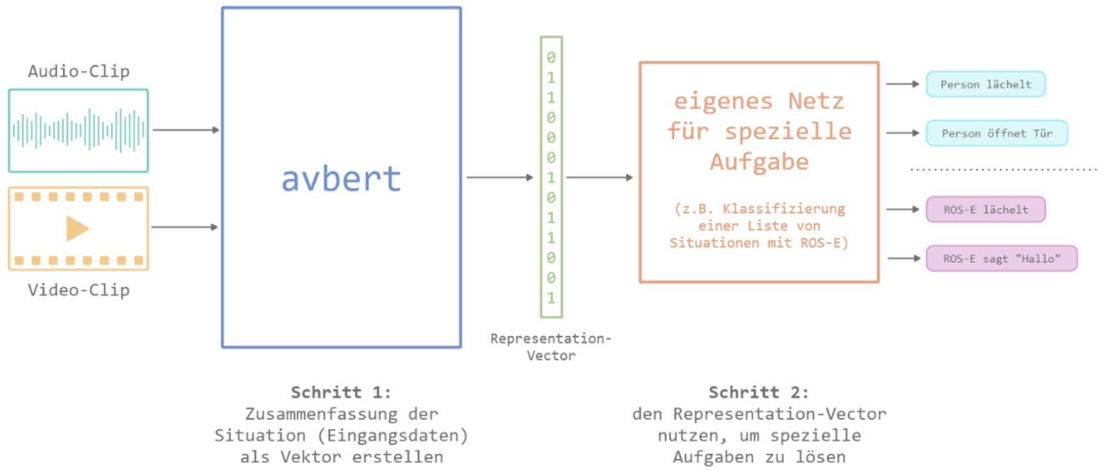

Abbildung 16: Die Schritte der Situationserkennung (Quelle: eigene Abbildung)

Um das Ergebnis des ersten Schrittes testen zu können und einen „autonomen Modus“ für die Situationserkennung unabhängig von laufenden Anwendungen zu haben, wird ein Beispiel für eine solche Weiterverarbeitung des *Representation-Vectors* umgesetzt. Die Aufgabe dieses zweiten Teilschrittes ist es, eine Aktion aus einer definierten Liste mit Hilfe des *Reinforcement Learning* zu wählen.

Zusammenfassung des Kapitels:

Das Ziel, das im Rahmen dieser Arbeit mit einer Situationserkennung erfüllt werden soll, wurde in zwei große Aufgaben geteilt. Die erste besteht darin, eine interne Abbildung der Eingangsdaten in Form eines Vektors zur Verfügung zu stellen. Mit diesem Vektor ist es anderen Anwendungen möglich, die zweite Aufgabe der Situationserkennung zu lösen: die Auswahl einer geeigneten Reaktion auf die aktuelle Situation. Um den ersten Teil der Situationserkennung im Rahmen dieser Arbeit testen zu können, wird für den zweiten Teil ebenfalls ein einfaches Beispiel umgesetzt. Die Eingangsdaten wurden auf Video- und Audiodaten festgelegt, die mit der Kamera und dem Mikrofon von ROS-E aufgenommen werden sollen.

6 Architektur

In Abschnitt 5.6 wurde die Gesamtarchitektur vorgestellt, die grundlegend aus zwei Schritten besteht. In diesem Kapitel sollen mögliche Architekturen für den ersten Teil des Netzes vorgestellt und eine dieser Architekturen für die Umsetzung ausgewählt werden. Anschließend wird das Konzept für den zweiten Teil des Netzes verfeinert.

6.1 Kriterien für die Auswahl einer Architektur

Zunächst hängt die Auswahl einer Architektur für ein Künstliches Neuronales Netz von der Art der Eingabedaten ab. Aus der historischen Entwicklung haben sich für bestimmte Modalitäten einzelne Architekturen als besonders geeignet erwiesen. So wurden lange Zeit CNNs für die Verarbeitung von Bildern verwendet. In den letzten Jahren wurden jedoch auch Ansätze für generelle Netze untersucht, die mehrere Modalitäten verarbeiten können. So wird beispielsweise die *Transformer*-Architektur nicht nur für *NLP*, sondern auch für *Computer Vision* verwendet. Die Art der Eingangsdaten kann die Architektur auch eingrenzen, je nachdem, ob z.B. einzelne Bilder oder ein Video verarbeitet werden soll. Entweder müssen die Frames als einzelne Bilder in das Netz gegeben werden, wodurch Informationen der Bewegung und des zeitlichen Ablaufs verloren gehen. Oder es muss eine Schleife wie bei den *RNNs* eingebaut werden, die aufeinander folgende Wörter eines Satzes verarbeiten. [6]

Mit den heutzutage verfügbaren Rechenkapazitäten könnten für die Erfüllung dieser Bedingungen sehr große Netze entwickelt werden. Da das Netz jedoch auf dem Tisch-Roboter ROS-E eingesetzt werden soll, gibt es eine weitere wichtige Rahmenbedingung für die Architektur des Netzes: es darf eine bestimmte Größe nicht überschreiten. Die Größe des Netzes bezieht sich dabei auf den Platz, den es im Arbeitsspeicher benötigt. Der Roboter ROS-E besitzt derzeit 8GB Arbeitsspeicher, wobei dieser bis zu diesem Zeitpunkt ebenfalls von den Netzen für die Verarbeitung von Sprachbefehlen und die Speech-to-Text-Funktion verwendet wird, die jeweils ca. 2 GB benötigen. In Zukunft könnte es auch noch weitere Netze geben, daher könnte es notwendig sein, die Netze auf einen alternativen Speicher auszulagern. Für die vorliegende Arbeit wird festgelegt, dass das Netz weniger als 4 GB beanspruchen darf.

6.2 Vergleich möglicher Architekturen für den Representation-Vector

Das im Rahmen dieser Arbeit erstellte Netz soll in der Lage sein, Video- und Audiodaten zu verarbeiten. Um die Auswahl der Architekturen besser zu verstehen, soll kurz vorgestellt werden, wie

diese Eingangsdaten an das Netz übergeben werden können. Ein Bild wird häufig als Matrix an ein Netz gegeben, die in zwei Dimensionen die Pixel des Bildes mit Helligkeitswerten enthält. Ist das Bild nicht nur schwarz-weiß, werden in einer dritten Dimension die drei Farbkanäle hinzugefügt. Soll ein Video bearbeitet werden, wird meist eine zusätzliche Dimension mit weiteren Bildern konstruiert. Audiodaten hingegen codieren ihre Information anhand von Wellenformen, die sich von Bildern in einer wichtigen Eigenschaft unterscheiden. Ein Bild ist die Aufnahme genau eines Punktes in der Zeit. Aus einer Momentaufnahme einer Wellenform könnte allerdings keine Information über die Frequenzen gewonnen werden, aus denen sich die Geräusche zusammensetzen. Eine Audiodatei muss daher auf geeignete Weise umgewandelt werden. In einer Audiodatei werden die Frequenzen, deren Amplituden und die Zeit abgebildet. In einem zwei-dimensionalen Diagramm (wie einem Bild) lassen sich üblicherweise jeweils zwei dieser Größen abbilden. Dabei wird mit Hilfe der FFT (Fast Fourier Transform) die Abbildung der Amplitude über die Zeit in die Abbildung der Frequenz über die Zeit umgewandelt. Um alle drei Informationen mit einem KNN gleichzeitig bearbeiten zu können, wird ein sogenanntes *Spektrogramm* erstellt. Dazu wird die FFT jeweils auf einer Anzahl an Samples (Datenpunkte der Audiodatei) durchgeführt und entlang der Zeitachse verschoben. Diese Abwandlung der Fourier-Transformation wird Kurzzeit-Fourier-Transformation (short-time Fourier transform, STFT) genannt. Anschließend enthält das *Spektrogramm* die Information über Frequenz pro Zeitfenster auf den Achsen und die Information über die Amplitude (die Lautstärke der einzelnen Frequenzen) als Werte von z.B. 0 bis 255. Dieses Diagramm kann von einem KNN dann wie ein Bild ausgewertet werden. Diese beiden Bilder werden jeweils in einen Vektor umgewandelt, um ihn an die Input-Schicht des Netzes zu übergeben. Da das Netz die beiden Modalitäten bearbeiten soll, müssen als nächstes die Probleme des *Alignments* und der *Fusion* gelöst werden. Anschließend soll dabei ein *Representation-Vector* gebildet werden, der mit möglichst wenigen Einträgen die aktuelle Situation genau genug abbilden kann, sodass damit die eigentliche Aufgabe der Situationserkennung gelöst werden kann. [38]

Im ersten Schritt werden daraus, ähnlich wie im visuellen Cortex des Menschen, sogenannte Features (Merkmale) extrahiert. Dabei wird der Input-Vektor meist komprimiert und bestimmte Merkmale aus dem Bild zusammengefasst. Ein als Vektor dargestelltes Bild ist meist sehr rechenintensiv, enthält aber viele redundante Pixel, die zum Beispiel zusammen eine Fläche oder ein Muster bilden. Ein Bild mit nur 28x28 Pixeln und einem Farbkanal bildet bereits einen Vektor mit 784 Werten. Dies wird verdreifacht, wenn das Bild farbig ist und nochmals verzehnfacht, wenn zehn Frames eines Videos betrachtet werden sollen. Durch diese Vorverarbeitung wird die Größe dieses Vektors stark verringert und die erste Abstraktion der Daten vorgenommen. Im Falle der Verarbeitung von zeitabhängigen Daten wie Videos kann in diesem Vektor auch eine Bewegung über mehrere Frames abgebildet werden.

Bei der Verarbeitung von zeitabhängigen Daten besteht generell das Problem, dass es einerseits Zusammenhänge in kurzen und langen Zeiträumen in den Daten gibt. Das Heben eines Armes kann mit 10 aufeinanderfolgenden Frames abgebildet werden. Diese Aktion kann aber möglicherweise nur über einen längeren Zeitraum der Aktivität des Armkreisens als Sportübung oder einer Wischbewegung beim Hausputz zugeordnet werden. Der Mensch könnte auch nach dem Armheben wieder eine andere Aktivität verfolgen. Ein einzelner Vektor könnte zum Beispiel über die Zeit von zwei Sekunden gebildet werden. In der Verarbeitung der Aktivität über einen längeren Zeitraum könnten mehrere dieser Vektoren betrachtet werden.

In diesem Abschnitt werden drei konkrete Architekturen vorgestellt, mit denen die Erstellung des *Representation-Vectors* umgesetzt werden könnte. Alle drei stammen aus Veröffentlichungen, die sich auf eine bestimmte Neuheit oder Besonderheit ihrer Architektur konzentrieren und diese gegen bisherige Ergebnisse vergleichen. Daher werden die Netze für den wirklichen Einsatz auf einem Roboter angepasst werden müssen. In solchen Veröffentlichungen werden meist Methoden und Tricks angewendet, für die viel Erfahrung oder Tests notwendig sind. Daher bieten sie eine gute Grundlage für die Auswahl einer Architektur.

6.2.1 Der MMAE Autoencoder

Das Forschungsthema dieses Ansatzes bestand darin, einen *Autoencoder* für verschiedene Modalitäten zu entwickeln, der weiterhin eine ähnliche Genauigkeit erreicht, obwohl die eingegebenen Sensordaten für längere Zeiträume fehlen. Dabei wurde das SNAPSHOT Datenset verwendet, das verschiedene Daten über den Zustand eines Menschen sammelt. Dazu wurden Daten von Sensoren am Handgelenk, Umfragen und eine App verwendet. Mit dem Sensor wurden Hautleitfähigkeit, Temperatur und Beschleunigung gemessen. Jeden Morgen und Abend haben die Teilnehmer ihre Stimmung als traurig oder fröhlich, ihr Stresslevel als hoch oder niedrig und ihren Gesundheitszustand als krank oder gesund eingeschätzt. Zusätzlich wurden mit der App die GPS-Position, Anrufe, SMS und die Aktivität des Screens gesammelt. Insgesamt wurden 11 Modalitäten betrachtet. Diese wurden mit dem *Autoencoder* zu einem komprimierten Vektor verrechnet. Anschließend wurde mit diesem Vektor die Aufgabe gelöst, die Stimmung, den Stress und den Gesundheitszustand zu klassifizieren.

Anschließend wurde ein Test durchgeführt, bei dem nach und nach eine Modalität entfernt wurde. Dabei sank die Genauigkeit der Vorhersage von 63,5 % mit allen Modalitäten auf 58,5 % mit 8 fehlenden Modalitäten. Dabei muss allerdings beachtet werden, dass die verwendeten Modalitäten unterschiedlich relevant für die Einschätzung sein können und die Verschlechterung der Genauigkeit daher auch davon abhängt, welche Modalitäten fehlen. Besitzen zwei Modalitäten viele Redundanzen, hat es keine große Auswirkung auf die Vorhersage, wenn eine davon fehlt. Eine Modalität könnte auch

vollständig irrelevant sein oder sogar die statistischen Zusammenhänge der anderen Modalitäten stören. In diesem Fall kann sich die Genauigkeit sogar verbessern, wenn diese Modalität fehlt.

Dieser Ansatz wurde betrachtet, da es im Falle der Situationserkennung ebenfalls vorkommen kann, dass ein Sensor ausfällt. Der Benutzer könnte auch die Kamera oder das Mikrofon für eine längere Zeit ausschalten. Da in dieser Arbeit nur zwei Modalitäten verwendet werden, würde die Verschlechterung der Genauigkeit vermutlich stärker ausfallen. In Zukunft wäre es aber denkbar, mehr Sensoren oder auch externe Daten wie das Wetter oder die Raumtemperatur zu verwenden.

Der Ansatz löst das Problem der *Fusion* und kann fehlende Modalitäten ausgleichen. Zudem ist ein Training mit ungelabelten Daten möglich. Das Problem des *Alignments* wird allerdings nicht implizit gelöst, was bedeutet, dass eine zusätzliche Methode dafür gefunden werden müsste. Der gesamte *Autoencoder* müsste in ein *RNN* eingefügt werden, sodass die zuvor eingegebenen Daten (Video-Frames und Audio-Abschnitte) berücksichtigt werden können. Es könnte auch der zuvor gebildete Vektor der Features über die Zeit als Input verwendet werden. Der *Autoencoder* hat jedoch keine Möglichkeit, ein zeitliches *Alignment* zu lernen und würde immer den gesamten Zeitraum als Ganzes betrachten.

Das Paper wurde 2017 veröffentlicht, im selben Jahr wie die *Transformer*-Architektur. Seitdem wurden die unterschiedlichsten Variationen des *Transformers* in vielen Veröffentlichungen untersucht, um das offene Problem des *Alignment* zu lösen. Zwei dieser *Transformer* sollen in den folgenden Abschnitten vorgestellt werden. [29]

6.2.2 Der UniT Transformer

Die Grundidee dieses in einem Paper vom August 2021 veröffentlichten Ansatzes besteht darin, einen einzelnen *Transformer* zu erstellen, der viele verschiedene Aufgaben lösen kann. In der Praxis wird eine Architektur meist auf eine bestimmte Aufgabe spezialisiert und die Architektur danach ausgewählt. In dieser Veröffentlichung sollte gezeigt werden, dass ein *Transformer* für mehrere Aufgaben trainiert und somit universeller eingesetzt werden kann. Der *Transformer* wurde für Objekterkennung, Visual Question Answering, Visual Entailment (ob das Bild zu einem Text passt), Question Answering (nur mit Texten), Textual Entailment, Fragen mit selbem Inhalt identifizieren und Sentiment Analysis (Einschätzung des Tonfalls eines Textes) trainiert. Für diese Aufgaben wurden teilweise nur Texte oder Bilder und teilweise beides gemeinsam als Inputs benötigt (siehe Abbildung 17). Es wäre möglich, diese Eingaben mit anderen Modalitäten auszutauschen oder weitere hinzuzufügen. In diesen Fällen müssten auch die sogenannten Output Heads angepasst werden, die jeweils eine bestimmte Aufgabe lösen.

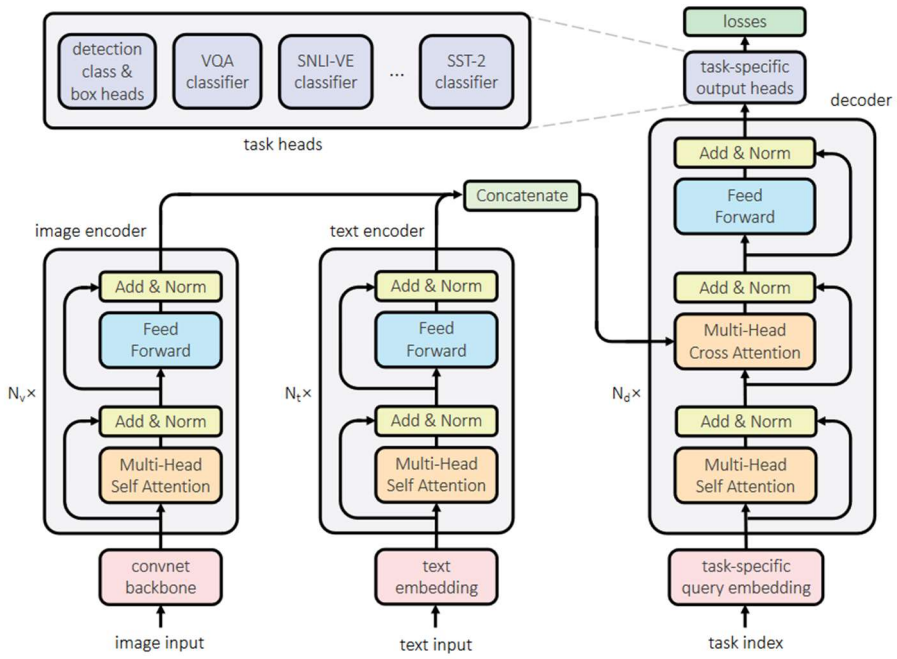

Abbildung 17: Der Aufbau des UniT-Modells (Quelle: https://arxiv.org/pdf/2102.10772.pdf)

Dieser *Transformer* löst die Probleme der *Fusion* und des *Alignments* implizit. Es liegen jedoch keine Daten dafür vor, wie gut er Video- und Audiodaten verarbeiten kann. Die beiden Modalitäten werden in diesem Netz über das Aneinanderfügen der Attention-Matrizen zusammengeführt und diese anschließend für die Einschätzung der Wichtigkeit für die aktuelle Aufgabe verwendet. Die verwendeten Modaltäten sind jedoch Text und Bild, die beide keine zeitliche Komponente besitzen. Es könnte daher das Problem auftreten, dass dieses einfache Aneinanderfügen als *Fusion*-Mechanismus für andere Modalitäten keine guten Ergebnisse erzielen würde.

Ein Vorteil dieses Ansatzes wäre es, dass eine Situationserkennung selbst aus verschiedenen Aufgaben bestehen könnte. Für eine Anwendung könnte die Aufmerksamkeit des Benutzers relevant sein, für eine andere das aktuelle Ziel. Diese Aufgaben könnten parallel mit Hilfe der Output Heads bearbeitet werden. [24]

6.2.3 Der AVBERT Transformer

Dieses Paper wurde im Mai 2021 veröffentlicht und untersucht zwei offene Themen bei der Arbeit mit *Transformern*. Das erste besteht darin, dass einige *Transformer*-Architekturen zwar schon mehr als eine Modalität bearbeiten, diese aber häufig auf Text und Bild begrenzt sind (siehe UniT). Die häufige Verwendung der Modalität Text könnte damit zusammenhängen, dass der *Transformer* ursprünglich für die Übersetzung von Texten entwickelt wurde. Dieser Ansatz stellt einen multimodalen

Transformer für Video Representation Learning (das Lernen eines *Representation-Vectors*) vor. Damit kann die Architektur genau den ersten Schritt der Verarbeitung durchführen, den das im Rahmen dieser Arbeit erstellte Netz ebenfalls ausführen soll. Das zweite in diesem Ansatz untersuchte Thema ist eine Methode für die Reduktion der Parameter, die gelernt werden müssen, um die Architektur effizient zu trainieren.

Der Code für diese Architektur wurde auf *GitHub* veröffentlicht. Es können sehr viele Anpassungen in einer Konfigurationsdatei vorgenommen werden, um die Architektur an die gewünschte Aufgabe oder die vorhandenen Ressourcen anzupassen. Das gesamte Netz wird aus vier Teilen zusammengefügt. Die ersten beiden Teile sind für die Vorverarbeitung der Video- und Audiodaten zuständig. Um die sogenannte Feature-Extraktion, also das Abstrahieren bestimmter Muster (z.B. die Kanten im Bild) vorzunehmen, wird ein *ResNet* (Residual Neural Network) verwendet. Dieses arbeitet ähnlich wie ein CNN (Convolutional Neural Network), besitzt aber zusätzliche Verbindungen, die bestimmte Schichten überspringen. Das *ResNet* liefert die Vektoren für jeweils eine festgelegte Anzahl an Frames bzw. Länge einer Audiodatei. Mehrere dieser kleinen Abschnitte werden an die *Transformer* übergeben. Es gibt drei verschiedene *Transformer* im Netz (siehe Abbildung 18). Zunächst werden die aus den Video- und Audiodaten erstellten Feature-Vektoren an einen *Transformer* der jeweiligen Modalität übergeben, um einen längeren Zeitraum betrachten und mit dem *Attention-Mechanismus* die wichtigen Features auswählen zu können. Anschließend wird mit dem dritten *Transformer* die *Fusion* zwischen den Modalitäten durchgeführt und der Repräsentations-Vektor erstellt. [39]

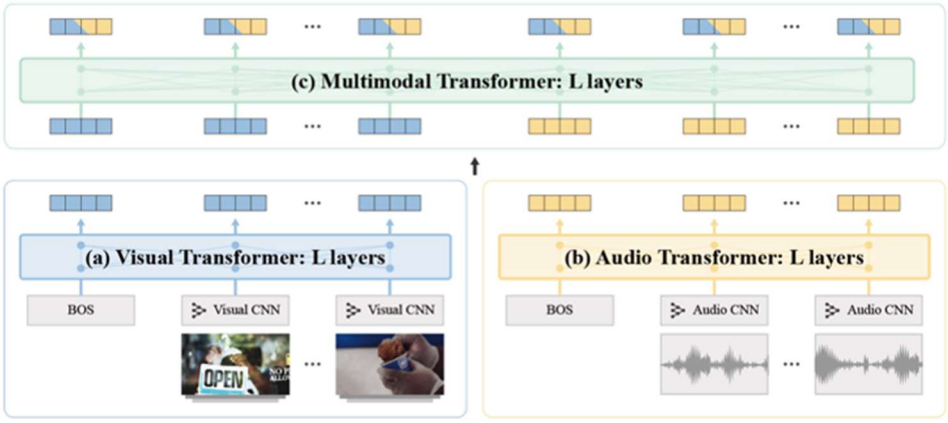

Abbildung 18: Der Aufbau des AVBERT Transformers (Quelle: https://sangho-vision.github.io/assets/poster/iclr2021_lee_poster.png)

Im vierten Teil des Netzes wird die eigentliche Aufgabe mit Hilfe des Representation-Vektors bearbeitet. Im veröffentlichten Code zu diesem Paper wurden drei verschiedene Netze für unterschiedliche Aufgaben erstellt. Das Netz kann jeweils eine Klassifikation von Audio- oder Videodaten alleine

oder gemeinsam ausführen. Es können aber auch andere Aufgaben definiert und unabhängig vom gesamten Netz trainiert werden.

In diesem Aufbau gibt es einige veränderbare Parameter, mit denen die Größe des gesamten Netzes und der Rechenaufwand verringert werden könnten. Dazu zählen die Anzahl der Schichten des CNNs für die Feature-Extraktion, die Anzahl der Schichten und Heads (wie viele verschiedene Attention-Maps erstellt werden) für die *Transformer*, die Größe der internen Vektoren und des Repräsentation-Vektors (wie stark die Informationen komprimiert werden) und die Art und Größe des Netzes, das für die eigentliche Aufgabe der Situationserkennung gewählt wird. Auch die Größe der verarbeiteten Bilder, die Anzahl der Frames und die Länge der Audiodateien kann verändert werden, um Ressourcen zu sparen. Durch diese Anpassungen kann die Genauigkeit der Vorhersage allerdings abnehmen. [40]

6.3 Auswahl der Architektur für den Representation-Vector

Die grundlegende Aufgabe der ausgewählten Architektur ist es, die Eingangsdaten zu einer komprimierten Abbildung zu verarbeiten. Dabei hat jede der im letzten Abschnitt vorgestellten Architekturen eigene Vor- und Nachteile.

Die Eigenschaft eines *Autoencoders*, fehlende Modalitäten auszugleichen ist besonders beim Einsatz auf ROS-E interessant, da häufig die Situation auftreten könnte, in der die Kamera und/oder das Mikrofon nicht immer eingeschaltet sind. Ob die anderen beiden Architekturen diese Eigenschaft ebenfalls besitzen, wurde in den jeweiligen Papers nicht betrachtet. Der AVBERT besitzt alle benötigten Eigenschaften: *Alignment* und *Fusion* über den *Attention-Mechanismus*, die Verarbeitung von Audio- und Video-Clips über viele kurze Abschnitte, die aber auch über einen längeren Zeitraum betrachtet werden können und das Teilen der Parameter, das dafür sorgt, dass die *Transformer* möglichst wenig RAM benötigen. Der *Autoencoder* und der UniT *Transformer* haben jedoch einen entscheidenden Nachteil: sie müssten erweitert werden, sodass sie zeitkontinuierliche Daten verarbeiten können. Daher wird für die Erstellung des *Representation-Vectors* im Rahmen dieser Arbeit der AVBERT verwendet.

Im nächsten Abschnitt wird das Konzept für den zweiten Teil des Netzes vorgestellt, das diesen Vektor verwendet, um eine passende Aktion auszuwählen, die ROS-E anschließend ausführt.

6.4 Der Ansatz des Reinforcement Learning

Als erstes werden die grundlegenden Begriffe und Vorgehensweisen des *Reinforcement Learning* vorgestellt. Ein sogenannter *Agent* (z.B. ein Roboter) bewegt sich in einem *Environment* (einer simulierten oder echten Umgebung).

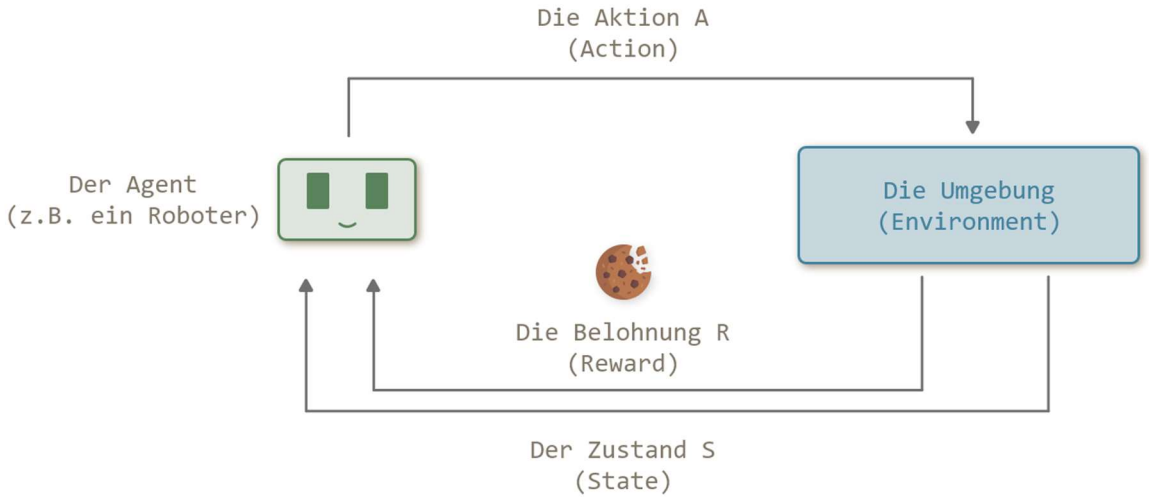

Abbildung 19: Grundbegriffe des Reinforcement Learning (Quelle: eigene Abbildung)

Der *Agent* nimmt in jedem Schritt einen *State* (den aktuellen Zustand) der Umgebung auf und wählt eine Aktion aus einer Liste möglicher Aktionen aus, die am besten zum jeweiligen Zustand passt. Welche die optimale Aktion ist, wird durch den Reward (eine mehr oder weniger große Belohnung) festgelegt. Diese Belohnung soll maximiert werden. Das Lernen geschieht anhand einer *Policy* (die Verhaltensstrategie). Darin wird gelernt, welche Aktionen die größte Belohnung für einen bestimmten Zustand erzielt haben, damit diese in Zukunft wahrscheinlicher beim selben Zustand ausgewählt werden. Das Lernen der *Policy* muss nicht mit Hilfe eines Künstlichen Neuronalen Netzes durchgeführt werden. Es gibt eine Vielzahl verschiedener Algorithmen für diesen Ansatz des Lernens. Sie unterschieden sich zum Beispiel in der Bewertungsmethode der Aktionen, dem Ablauf des Lernens der Art der Eingangs- und Ausgangsdaten. Für die verschiedensten Anwendungsfelder wurden Erweiterungen, wie zum Beispiel das offline *Reinforcement Learning* entwickelt, bei dem der *Agent* die Trainingsbeispiele nicht selbst sammeln muss, sondern auch von den Beispielen anderer *Agents* lernen kann. [13]

Um einen Algorithmus oder eine Methode auswählen zu können, muss zunächst festgelegt werden, welche Eigenschaften die Daten aus dem *Environment* und die Aktionen besitzen. Die Verhaltensstrategie (*Policy*) soll später abbilden, welche Eingangsdaten aus dem *Environment* zu welcher Aktion

führen. Um eine passende Methode für das Lernen dieser Strategie auszuwählen, müssen alle möglichen Aktionen, die Zustände des *Environments* und des *Agents* selbst festgelegt werden. [41]

6.4.1 Beschreibung der Aktionen und Zustände des Agents und Environments

Die Aktionen, die im Rahmen dieser Arbeit verwendet werden, dienen dazu, das Konzept testen und dessen Qualität einschätzen zu können. Es werden Aktionen gewählt, die auf ROS-E bereits ausgeführt werden können. Diese bilden hauptsächlich den Anwendungsfall als Anschauungsmittel auf einer Messe oder in einem Showroom ab und können später erweitert werden (siehe Tabelle 5).

Tabelle 5: Auswahl der Aktionen für den Reinforcement-Ansatz

ID	**Aktion**
A1	Sprachausgabe: [„Hallo“, „Hi“, „Guten Tag“] (zufällig gewählt für mehr Abwechslung)
A2	Animation abspielen: zwinkern
A3	Starte Befehl: Witz erzählen
A4	Animation abspielen: fast einschlafen
A5	Sound-Clip abspielen: summen
A6	Sprachausgabe: [„Auf Wiedersehen“, „Komm bald wieder“] (zufällig gewählt für mehr Abwechslung)
A7	nichts tun

Die Art der Aktionen beeinflusst die Wahl der Methode vor allem bei der Frage, ob die Aktionen diskret oder kontinuierlich sind. In diesem Fall sind die Aktionen diskret. Eine kontinuierliche Aktion könnte beispielsweise der Drehwinkel eines Motors sein.

Der nächste Schritt der Definition besteht darin, den internen Zustand des *Agents* zu betrachten. Dieser ist relevant, da es vorkommen könnte, dass nicht in jedem Zustand jede Aktion ausgeführt werden darf. Zum Beispiel könnte ROS-E in den Zustand „Augen geschlossen“ übergehen, nachdem die Aktion „Animation abspielen: einschlafen“ ausgeführt wurde. Sollte es in diesem Zustand möglich sein, einen Witz zu erzählen? Mit der oben getroffenen Auswahl ist es aber möglich, alle Aktionen in beliebiger Reihenfolge auszuführen.

Eine wichtige Größe, die starken Einfluss auf die Wahl der Methode haben kann, ist die Art der Daten des *Environments*. Dabei wird besonders in Anwendungen der realen Welt zwischen dem *State* des *Environments* und der *Observation* des *Agents* unterschieden. Der *State* des *Environments* beinhaltet alle Informationen der Umgebung (z.B. der gesamte Plan eines Labyrinths). Da aber der *Agent* nicht

zu jeder Zeit den Zustand der gesamten realen Welt aufnehmen kann, wurde für diese Art der Problemstellung der Begriff der *Observation* (Beobachtung) eingeführt. Diese *Observation* enthält einen meist kleinen Teil des *Environments*, der zum Beispiel über eine Kamera oder ein Mikrofon aufgenommen wurde. Auch hier stellt sich die Frage, ob die Beobachtungen diskrete oder kontinuierliche Werte annehmen. In diesem Fall sind die Beobachtungen jeweils ein *Representation-Vector* mit kontinuierlichen Elementen (Fließkommazahlen). Der zweite wichtige Faktor ist die Größe der Beobachtungs-Menge. In simulierten und stark vereinfachten *Environments* kann diese Anzahl so klein sein, dass eine Tabelle der möglichen Zustände und deren Aktionen per Hand aufgestellt werden kann. Ein Beispiel dafür wäre das Finden eines Weges durch ein Labyrinth, bei dem es nur die Situationen „Wand rechts“, „Wand links“ und „Wand vorne“ gibt. Im Fall der Situationserkennung mit ROS-E gibt die Größe des *Representation-Vectors* die Größe der Zustandsmenge an. Wird dieser zu klein gewählt, können die Situationen möglicherweise nicht eindeutig auseinandergehalten werden. Die Größe dieses Vektors wurde von den Entwicklern auf 768x4 festgelegt. Da diese Tests auf mehreren Datensets durchgeführt und somit mehr Erfahrung haben, wurde dieser Wert nicht verändert. Die Anzahl der möglichen Zustände des *Environments* ist also sehr groß und kann nicht mehr einfach in einer Tabelle verwaltet oder per Hand aufgestellt werden. [13]

Nachdem diese wichtigen Eigenschaften ermittelt wurden, kann im nächsten Abschnitt ein geeigneter Algorithmus ausgewählt werden.

6.4.2 Auswahl der Methode

Für das gegebene Problem der Situationserkennung wird also eine Methode für diskrete Actions, kontinuierliche *States* und eine sehr große Menge möglicher *States* benötigt. Eine der bekanntesten Methoden ist das sogenannte *Q-Learning*. Dabei werden für jeden *State* alle möglichen Aktionen mit Hilfe der sogenannten Q-Funktion mit einer Qualität bewertet, um die größtmögliche Belohnung zu berechnen. Das *Deep Q-Network* (DQN) wurde als eine Erweiterung des *Q-Learning* entwickelt, um den *Q-Learning*-Algorithmus mit *Deep Neural Networks* auf hochdimensionale Eingangsdaten anwenden zu können. Das *Q-Learning* selbst ist ein Algorithmus und benötigt keine Künstlichen Neuronalen Netze. Für kleine Probleme kann die *Policy* beim *Q-Learning* in sogenannten Q-Tables aufgeführt werden. Da das *Environment* in diesem Fall sehr komplexe Daten und sehr viele verschiedene mögliche *States* liefert, kann ein KNN erfolgreich verwendet werden, um die *Policy* zu lernen. Dabei wird die Q-Funktion nicht mehr berechnet, sondern mit Hilfe des KNNs approximiert. Das *DQN* ist mit den Eigenschaften des Systems kompatibel und wird in vielen Veröffentlichungen verwendet. So war es möglich, den Code für ein Beispiel der Umsetzung des *DQN* von der offiziellen *PyTorch*-Seite zu übernehmen. [13]

Das Lernen läuft dabei nach folgendem Schema ab:

1. der *State* des *Environments* wird erfasst: im Fall von ROS-E werden die Video- und Audiodaten der letzten x Sekunden mit dem ersten Netz zu einem Vektor komprimiert und an das zweite Netz übergeben
2. das zweite Netz gibt die Aktion aus, die laut der *Policy* die meiste Belohnung für den aktuellen *State* gibt
3. die Aktion wird von ROS-E ausgeführt
4. die zugehörige Belohnung und der nächste *State* vom *Environment* werden abgefragt
5. diese sogenannte *Experience* (Erfahrung) wird gespeichert: der aktuelle *State*, die ausgewählte Aktion, die Belohnung und der nächste *State*
6. wenn eine bestimmte Anzahl an *Experiences* gesammelt wurde, werden die Gewichte des Netzes angepasst und in Zukunft werden die Aktionen mit der neuen *Policy* ausgewählt

[42] [43] [13]

Dieser Code wurde mit einem zufällig generierten Vektor und zufälligen Belohnungen für jede Aktion direkt auf ROS-E getestet, um zu prüfen, ob die Rechenleistung ausreichen würde. Der Arbeitsspeicher wurde dabei lediglich zu 3 % ausgelastet, die CPU hingegen zu ca. 250 % (von 400 %, da der Raspberry Pi vier Kerne besitzt). Diese Auslastung würde lediglich während der Trainings-Phasen entstehen.

Das Künstliche Neuronale Netz, das die Q-Funktion annähert, benötigt meist keine so komplexe Architektur wie der AVBERT. Es lernt eine Zuordnung der Eingangsdaten (den *Representation-Vector*) zu einer der möglichen Aktionen. Für diese Aufgabe kann ein *Multi-Layer Perceptron* (MLP), die einfachste Form eines KNN, verwendet werden. Der Input-Layer muss genauso viele Knoten enthalten, wie die Länge des Vektors vorgibt. Die bestmögliche Anzahl der *Hidden-Layer* und deren Knoten kann nicht vorhergesagt werden und muss zunächst einfach festgelegt und später im Test angepasst werden. Es wurde eine Schicht mit 128 Knoten als Einstiegswert gewählt. Der Output-Layer muss so viele Knoten besitzen, wie es Aktionen gibt, also die sieben oben gewählten Aktionen. [13]

Zusammenfassung des Kapitels

Für die Architektur des ersten Teilnetzes wurde der AVBERT gewählt. Dieser nimmt einen Video-Clip bestehend aus einzelnen Bildern (Frames) und einem in *Spektrogramme* umgewandelten zugehörigen Audio-Clip entgegen. Die Clips werden zuerst getrennt von zwei *ResNets* (Residual Neural Networks) bearbeitet, um eine Feature-Extraktion durchzuführen, also zum Beispiel bestimmte Muster, Objekte oder Geräusche zu erkennen. Anschließend werden diese Features mit einem *Transformer* und somit unter Verwendung des *Attention-Mechanismus* zusammengeführt. Dabei werden die Aufgaben des *Alignments* und der *Fusion* gelöst. Zum Beispiel könnte eine erkannte Katze mit einem „Miau"-Geräusch verbunden werden. Eine komprimierte und abstrakte Version dieser Situation wird in Form des *Representation-Vectors* ausgegeben.

Dieser Vektor wird verwendet, um den Zustand (*State*) des *Environments* bzw. die *Observation* für den Reinforcement-Ansatz abzubilden und eine Zuordnung der *States* zu den bestmöglichen Aktionen in jeder Situation auszuwählen, die von ROS-E ausgeführt werden sollen. Das zweite Teilnetz lernt, die Funktion anzunähern, die die Qualität der möglichen Aktionen bewertet. Dieses Netz ist ein *Multi-Layer Perceptron* mit drei Schichten.

7 Vorbereitung des Trainings

Um beide Teilnetze trainieren zu können, müssen im nächsten Schritt passende Trainingsarten gefunden werden. Die grundlegenden Trainingsarten für Künstliche Neuronale Netze werden kurz vorgestellt, bevor mögliche Trainingsstrategien diskutiert und ausgewählt werden.

Es gibt drei grundlegende Arten für das Training von Künstlichen Neuronalen Netzen: *Supervised Learning* (Beaufsichtigtes Lernen), *Unsupervised Learning* (Unbeaufsichtigtes Lernen) und *Reinforcement Learning* (Bestärkendes Lernen). Beim *Supervised Learning* werden Zusammenhänge zwischen Inputs und Outputs gelernt, indem für jeden Input der „richtige", gewünschte Output vor dem Training festgelegt wird. Dieser vorgegebene Output wird *Label* genannt. Für jedes gezeigte Beispiel errechnet das Netz eine Vorhersage, die anschließend mit dem *Label* verglichen wird, um den Fehler zu bestimmen, den das Netz gemacht hat. Das bedeutet in den meisten Fällen, dass diese *Label* von einem Menschen für sehr viele Beispiele per Hand vergeben werden müssen.

Das *Unsupervised Learning* verfolgt die Strategie, im Gegensatz zum *Supervised Learning* ohne die per Hand vergebenen *Label* zu arbeiten. Dabei gibt es zwei Ansätze um das Wissen an das Netz zu übergeben. Der erste Ansatz ist dem *Supervised Learning* sehr ähnlich. Die Vorhersage des Netzes wird wieder mit einem *Label* verglichen, um den Fehler des Netzes zu bestimmen. Der entscheidende Unterschied ist aber, dass die *Label* nicht mehr von Menschen per Hand vergeben werden müssen. Das ist möglich, indem der korrekte Output implizit festgelegt oder berechnet wird. Ein gutes Beispiel dafür ist der *Autoencoder*, bei dem der Output dem Input so ähnlich wie möglich sein soll. Daher ist der korrekte Output bereits bekannt und der ursprüngliche Input kann als *Label* verwendet werden. Der zweite Ansatz des *Unsupervised Learning* besteht darin, keine *Label* zu verwenden, sondern Zusammenhänge zwischen den Daten zu lernen. Ein Beispiel dafür wäre die Bildung von Clustern, aber ohne zu wissen wie viele oder welche Cluster es gibt. So kann das Netz z.B. ein Bild von einer Katze nicht als „Katze" identifizieren, aber sagen, welchem anderen Bild (einer Katze) es am ähnlichsten ist. [13] [44]

Das *Reinforcement Learning* wurde im letzten Kapitel bereits vorgestellt. Für jede dieser drei Trainingsarten gibt es zahlreiche Abwandlungen und Verfahren, die auch mehrere der Trainingsarten kombinieren. Im Folgenden sollen einige dieser Trainingsarten vorgestellt werden, um zu ermitteln welche für das Training des hier verwendeten Netzes geeignet sind.

7.1 Die Trainingsphasen

Ein neu erstelltes Netz wird häufig mit zufälligen Werten für die Gewichte initialisiert. Das sogenannte *Pre-Training* kann dazu dienen, die Gewichte mit Allgemeinwissen vorzutrainieren, bevor die eigentliche Aufgabe trainiert wird. Diese Methode wird für die Verarbeitung von Texten und auch Bildern häufig verwendet. Ein Netz, das entscheiden soll, ob ein Text zu einer bestimmten Überschrift passt, kann im *Pre-Training* zunächst die Struktur von Sätzen und grundlegende Eigenschaften der Sprache lernen. Ein anderes Netz, das entscheiden soll, ob eine bestimmte Person auf einem Bild zu sehen ist, kann lernen, Kanten, Muster und Gesichter zu erkennen. Die Trainingsdaten für das *Pre-Training* bilden zwar grundlegende Strukturen in Bildern oder Sätzen ab, müssen jedoch nicht unbedingt die Zusammenhänge enthalten, die später im Einsatz auftreten. In einer zweiten Trainingsphase wird daher die eigentliche Aufgabe des Netzes meist mit selbst gesammelten Daten trainiert. Dieses zweite Training wird *Finetuning* genannt. [13] [45]

7.1.1 Das Pre-Training

Der große Vorteil des *Pre-Trainings* besteht darin, dass vorhandene Datensets verwendet werden können, die nicht genau die spätere Aufgabe abbilden. Im Rahmen dieser Masterarbeit wird es nicht möglich sein, genügend vielfältige Daten für das Training einer Situationserkennung selbst mit ROS-E zu sammeln. Das Netz kann aber trotzdem mit anderen Datensets vortrainiert werden.

Im Paper zum AVBERT wurden zwei verschiedene Aufgaben beschrieben, mit denen das *Pre-Training* durchgeführt wurde. Die erste Aufgabe ist eine sogenannte *Masked Embedding Prediction.* Dabei werden zufällige Frames oder Teile eines Bildes bzw. Audio-*Spektrogramms* maskiert (mit speziellen Werten überschrieben). Das Netz hat die Aufgabe, diese fehlenden Teile im Video oder Audio zu ersetzen. Im Fall der Zeit-kontinuierlichen Daten muss dieser Ansatz leicht verändert werden, da die Daten diskret sein müssten, um das Training mathematisch berechnen zu können. Daher generiert der AVBERT nicht selbst die maskierten Stellen, sondern trifft eine Auswahl, welcher Frame von einer bestimmten Anzahl an vorgegebenen Frames der Richtige ist. Dabei ist entscheidend, welche Negativ-Beispiele gewählt werden, da diese bestimmen können, welche Features für eine Entscheidung relevant sind. Diese Features könnten die Helligkeit des Bildes, die Farbe eines bestimmten Objektes oder der Gesichtsausdruck oder die Stimmlage einer Person sein. Diese Aufgabe wird dazu verwendet, um Zusammenhänge innerhalb der einzelnen Modalitäten (also „hören“ und „sehen“ für sich) zu lernen. Die zweite Aufgabe trainiert gezielt die Zusammenhänge zwischen den Modalitäten. Dazu wird die sogenannte *Correct Pair Prediction* verwendet. Dabei muss das Netz für jeweils einen gezeigten Video-Clip und den dazu gespielten Audio-Clip entscheiden, ob diese zusammenpassen oder nicht. Beide Trainingsaufgaben zählen zum *Unsupervised Learning*. [40]

Dieses *Pre-Training* für das Paper wurde auf 64 *NVIDIA* Tesla V100 GPUs durchgeführt. Nach dem Training werden die gelernten Gewichte eines Netzes in einem sogenannten *Checkpoint* gespeichert, um das Wissen auf diese Weise in jedes Netz mit demselben Aufbau laden zu können. So können Netze effektiv gespeichert und übertragen werden. Ein solcher *Checkpoint* nach dem im Rahmen des Papers durchgeführten *Pre-Trainings* wurde ebenfalls auf *GitHub* bereitgestellt. Dieser *Checkpoint* ist dafür gedacht, es als Grundlage zu verwenden, um das Netz anschließend auf eine bestimmte Aufgabe zu trainieren. Im Rahmen dieser Arbeit kann dieser *Checkpoint* allerdings nicht verwendet werden, da das Netz verkleinert wurde, um es auf ROS-E einzusetzen. Das *Pre-Training* musste für die angepasste Konfiguration des Netzes selbst durchgeführt werden. Ein großes Problem bei diesem Vorhaben stellte die Tatsache dar, dass der Code für das im Paper durchgeführte *Pre-Training* nicht auf *GitHub* veröffentlicht wurde. Die beiden im Paper verwendeten und dort verfeinerten Methoden selbst mathematisch korrekt zu implementieren ist im Rahmen dieser Arbeit jedoch nicht möglich. Daher musste für das *Pre-Training* im Rahmen dieser Arbeit die im Code bereitgestellte Klassifizierungs-Aufgabe verwendet werden, die dort als *Finetuning* verwendet wurde. Die Aufgabe der Klassifizierung erlaubt es, Zusammenhänge in den Daten zu lernen, die aber möglicherweise weniger allgemeingültig sind, als jene mit den *Pre-Training* Methoden des Papers gelernten Zusammenhänge. Da das Netz in diesem Fall speziell für die Situationserkennung auf ROS-E entwickelt werden soll, muss untersucht werden, ob die Ergebnisse trotzdem zufriedenstellend sind. Um die Klassifizierungs-Aufgabe für das Training zu definieren, wird an den AVBERT ein als „Classification-Head“ bezeichnetes Netz angehängt (siehe Abbildung 20). [39]

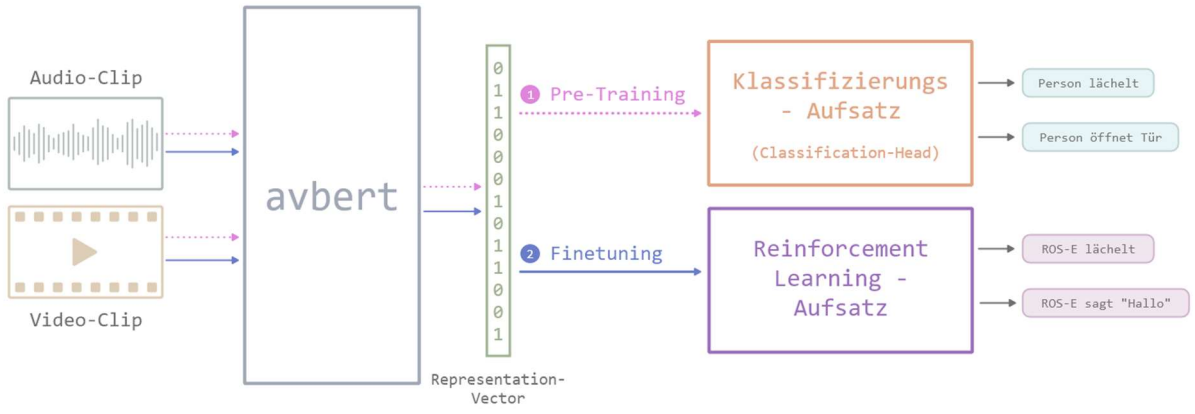

Abbildung 20: Aufbau des gesamten Netzes in verschiedenen Trainings-Phasen (Quelle: eigene Abbildung)

Dieses verwendet den *Representation-Vector* vom AVBERT und teilt diese abstrakte Darstellung der aktuellen Situation aus Audio- und Video-Clip einer bestimmten Klasse (z.B. „nicken“, „winken“, „essen“, „schlafen“, …) zu. Dabei kann entweder nur der *Classification-Head* oder das gesamte Netz trainiert werden. Im ersten Fall verändern sich die Gewichte des AVBERT und somit der

Representation-Vector nicht und nur der letzte Teil des Netzes wird auf die spezielle Aufgabe trainiert. Es ist aber auch möglich, das ganze Netz zu trainieren. In diesem Fall wird der *Representation-Vektor* so angepasst, dass er die Situationen, die der *Classification-Head* vorgibt, mit möglichst stark unterschiedlichen Vektoren darstellt. Auf diese Weise kann mit der Auswahl der Klassen und deren Trainingsdaten bestimmt werden, welche Inhalte besonders wichtig sind, um die gewünschten Situationen im *Representation-Vector* zu abstrahieren. Ein Beispiel dafür wäre, dass das Netz lernen muss, auf den Kopf und die Handbewegung der Personen zu achten, wenn es „nicken" von „essen" unterschieden soll. Diese Klassifizierungs-Aufgabe stellt eine übergeordnete Aufgabe dar, wie sie im menschlichen Gehirn vom *FPAN* verwendet wird, um die verknüpften Sinneseindrücke zu priorisieren. Eine solche Priorisierung geschieht im AVBERT durch den Einsatz des *Attention-Mechanismus* im *Transformer*.

Auf der *GitHub*-Seite zum AVBERT werden drei Datensets vorgeschlagen, für die das Netz von Anderen trainiert und die Ergebnisse mit denen des Papers verglichen werden können. Dabei ist die Aufgabe des Netzes jedes Mal eine Klassifizierung mit *Supervised Learning*. Für das Training des *ResNet* für die Video-Frames wird das „UCF101 - Action Recognition Data Set" verwendet, das 101 Klassen mit menschlichen Tätigkeiten (Instrumente spielen, Sportarten, Zähne putzen, stricken, …) enthält. Für das Training des *ResNet* für die Audio-Daten wird das „ESC-50 - Dataset for Environmental Sound Classification" verwendet, das 50 Klassen mit verschiedenen Geräuschen (Katze, weinendes Baby, Lachen, Schritte, Klopfen an der Tür, …) enthält. Für das Training des gesamten Netzes wird das Kinetics-Sounds Dataset verwendet, das 32 Klassen mit menschlichen Tätigkeiten (Instrumente spielen, singen, lachen, Rasen mähen, …) enthält. Da diese Auswahl der Datensets nicht auf eine Situationserkennung mit Robotern ausgelegt ist, wurde im Rahmen dieser Arbeit ein eigenes Datenset erstellt. Mit diesem Datenset wurde anschließend das *Pre-Training* durchgeführt. Die Erstellung des eigenen Datensets wird im nächsten Kapitel beschrieben. [40]

Der *Classification-Head*, der für das *Pre-Training* entwickelt und trainiert wurde, kann anschließend auch neben dem *Representation-Vector* anderen Anwendungen auf ROS-E zur Verfügung gestellt werden, die die ausgegebenen Klassen verarbeiten möchten.

7.1.2 Das Finetuning

Nachdem das Netz die grundlegenden Zusammenhänge der Bilder und Audiodaten gelernt hat, muss es auf die eigentliche Aufgabe spezialisiert werden. Im *Pre-Training* wurden keine Clips gezeigt, die direkte Interaktionen mit ROS-E enthalten. Außerdem wurden die Clips nicht mit der Kamera und dem Mikrofon von ROS-E aufgenommen. Um die Ergebnisse des Netzes zu verbessern, können weitere Trainingsdurchläufe mit eigenen Daten durchgeführt werden, die die Aufgabe der Situationserkennung mit ROS-E besser abbilden.

Dieses Training kann im Rahmen dieser Arbeit nicht mehr durchgeführt werden, da es zeitlich und aufgrund derzeitiger Kontaktbeschränkungen möglicher Testpersonen nicht möglich sein wird, genügend Trainingsdaten zu sammeln. Es soll jedoch ein Konzept vorgeschlagen werden, mit dem eigene Trainingsdaten direkt mit ROS-E im späteren Einsatz gesammelt werden können. Dieses Konzept wird in Kapitel 8 vorgestellt.

7.1.3 Das Training des Reinforcement-Ansatzes

Für das Training des zweiten Teilnetzes wird der *Classification-Head* mit dem *Multi-Layer Perceptron* ausgetauscht, das lernt, den *Representation-Vector* mit einer ausführbaren Aktion zu verknüpfen (siehe Abbildung 20). Wenn der *Representation-Vector* also ein „Winken“ abbildet, soll dieses Netz lernen, darauf zum Beispiel mit der Aktion „Hallo sagen“ zu antworten.

Das Netz könnte mit dem ursprünglichen *DQN*-Algorithmus direkt trainiert werden. Dabei würden die aktuellen Daten der Kamera und des Mikrofons verarbeitet und anschließend verworfen werden, während die Gewichte des Netzes angepasst werden und so die *Policy* gelernt wird. Alternativ könnte das Training mit Hilfe des sogenannten *Experience Replays* durchgeführt werden. Dabei werden alle sogenannten *Experiences* gespeichert. Eine solche *Experience* besteht aus dem aktuellen *State* der Umgebung (bzw. der *Observation*), der daraufhin gewählten Aktion, deren Belohnung und dem nächsten *State*. Aufgrund dieser Werte ist es möglich, das Netz nicht sofort zu trainieren oder auch mehrere Iterationen über die Trainingsdaten durchzuführen. Dieses Vorgehen hat den Vorteil, dass das Training beispielsweise immer nachts durchgeführt werden kann oder wenn gerade viel Rechenleistung auf ROS-E zur Verfügung steht. Zudem können die *Experiences* auch in einer Datenbank gespeichert und zwischen mehreren Robotern ausgetauscht werden. [46]

Eine andere mögliche Erweiterung des *DQN*-Algorithmus stellt das *Double DQN* dar. Dabei werden zwei Instanzen des Netzes angelegt. Die eine Instanz wird trainiert, während die andere die Vorhersagen für die *Experiences* erstellt. Nach einer festgelegten Anzahl an *Experiences* wird die zweite Instanz mit den neuen Gewichten der Trainings-Instanz aktualisiert. Seit 1993 wurde ein Problem des

DQN-Algorithmus untersucht, bei dem durch verschiedene Faktoren (die zufälligen Gewichte beim Erstellen des Netzes, Ungenauigkeiten oder Störungen in den Eingangsdaten, …) der Wert der Aktion regelmäßig zu hoch eingeschätzt wird. Um dies zu verhindern werden die Aufgaben der Bewertung der Aktion und der Auswahl der Aktion getrennt. [47] [13]

Da beide Erweiterungen des *DQN*-Algorithmus die Ergebnisse des Trainings verbessern sollen, wurden diese für das Training des zweiten Teilnetzes verwendet.

7.2 Datenbeschaffung und -aufbereitung

Nachdem die einzelnen Schritte des Trainings festgelegt wurden, müssen die passenden Daten beschafft werden. Die Qualität der Daten entscheidet, wie gut das Netz die gewünschten Zusammenhänge lernt. Mit einem sogenannten Data Collection Plan soll unter anderem sichergestellt werden, dass die Trainingsdaten auch die Zusammenhänge abbilden, die das Netz lernen soll. Dieser Abschnitt orientiert sich an den Inhalten eines Data Collection Plans, um genau festzuhalten, welche Daten für welche Art von Training benötigt werden und wie diese gesammelt werden sollen.

7.2.1 Zwecke der Datensammlung

Für die beiden Phasen des Trainings des AVBERT werden jeweils Trainings- und Testdaten benötigt. Im *Pre-Training* soll der AVBERT so trainiert werden, dass er *Representation-Vektoren* erstellt, die die Situationen mit möglichst stark unterschiedlichen Vektoren abbilden. Dazu gehört das Lernen von Zusammenhängen in jeder Modalität und zwischen den Modalitäten, um die Situation im *Representation-Vector* zu abstrahieren. Für das *Finetuning* sollen Daten verwendet werden, die direkt mit ROS-E aufgezeichnet wurden und häufig auftretende Situationen im Umgang mit dem Roboter und in dessen Umgebung abbilden.

Mit dem Reinforcement-Ansatz soll eine *Policy* erstellt werden, die eine Verknüpfung von *Representation-Vektoren* mit passenden Aktionen von ROS-E lernt. Das Ziel der ersten Datenerhebung ist es, grundlegende Situationen im Alltag bzw. bei der Präsentation von ROS-E auf Veranstaltungen (z.B. dem Hochschulinformationstag oder im Showroom) zu sammeln.

7.2.2 Aufbau der Daten

Die Daten werden in Form von .mp4-Dateien eingelesen. Darin befinden sich Audio- und Video-Daten, die jeweils kontinuierlich sind. Die Länge der Videos liegt zwischen einer und elf Sekunden. Die Video- Framerate, die Auflösung der Bilder und die Audio-Frequenz werden auf 24 FPS, 112x122 Pixel und 44100 Hz für alle eingelesenen Clips umgerechnet. Daher besteht keine besondere Anforderung an die Auflösung oder Framerate der ursprünglichen Clips. [39]

Die Daten für das *Pre-Training* sollen Situationen im Alltag, menschliche Interaktionen und Aktivitäten enthalten. Die Daten, die während des *Finetunings* gesammelt werden, sollen direkt mit der Kamera von ROS-E aufgezeichnet werden. Dabei sollen ebenfalls alltägliche Situationen mit und ohne direkte Interaktion mit dem Roboter abgebildet werden. Während der Aufzeichnung mit ROS-E wird die Video-Framerate auf 24 FPS festgelegt. Die Auflösung der Videos auf 640x360 Pixel, um Augmentation durchführen zu können (siehe Abschnitt 7.4.2) und die Audio-Frequenz auf 44100 Hz.

Die Daten, die während des Trainings des zweiten Teilnetzes mit dem Reinforcement-Ansatz zwischengespeichert werden, bestehen aus dem Ausgangszustand (als *Represenatetion-Vector* der aktuellen Situation), der gewählten Aktion, deren Belohnung und dem nächsten Zustand (als *Represenatetion-Vector* der danach eingetretenen Situation). Die ursprünglichen Video- und Audio-Clips werden aus zwei Gründen ebenfalls in die Datensammlung eingebunden. Zum einen können auf diese Weise eigene Trainingsdaten für das Training des ersten Teilnetzes gesammelt werden. Zum anderen ist der *Representation-Vector* abhängig von der Version des ersten Teils des Netzes. Wenn dieses also neu trainiert wird, können die Vektoren, die mit dem alten *Checkpoint* des AVBERT erstellt wurden, nicht mehr verwendet werden. Indem die Clips mitgespeichert werden, können die Vektoren mit dem aktuellen *Checkpoint* neu berechnet werden.

7.3 Datenbeschaffung

Die Beschaffung der Trainingsdaten für das *Pre-Training* erfolgte über das Internet. Dazu wurden mögliche Datensets recherchiert und überprüft, ob diese für das Training des Netzes zur Situationserkennung geeignet sind. Die Datensets wurden über die Seite „Papers With Code“ recherchiert, die es ermöglicht, Datensets nach einer bestimmten Aufgabe zu filtern. In diesem Fall wurden Aufgaben wie „Action Recognition“ und „Video Classification“ ausgewählt. Die nachfolgende Tabelle zeigt die Datensets, die für das Training einer Situationserkennung in Frage kommen könnten. [21] [48]

Tabelle 6: Übersicht über mögliche Datensets für das Training mit Video-Clips (Video und Audio)

Name	Anzahl der Klassen	Anzahl der Clips pro Klasse (Train)	Quelle
UCF101	101	13.320 / 101 = 132	[49]
Kinetics 700	700	500 – 900	[50]
HMDB51	51	min. 101	[51]
ActivityNet	200	137	[52]
Something-Something V2	174	220.847 / 174 = 1.269	[53]

Ein Datenset wird als Ordnerstruktur verwaltet, in dem jede Klasse einen eigenen Ordner erhält. In diesem Ordner werden die Beispiele für die Klasse, in diesem Fall als .mp4-Datei abgelegt. Zudem gibt es in jedem Datenset mindestens eine Unterteilung in „train“- und „test“-Ordner. Diese enthalten jeweils die Ordner der einzelnen Klassen mit einer disjunkten Teilmenge der Daten.

Während der ersten Test-Durchläufe des Trainings mit den bereits vorhandenen Datensets (UCF101 und Kinetics-Sounds) wurde deutlich, dass diese nur für das Training des *Classification-Heads* geeignet waren. Wurde das gesamte Netz damit trainiert, konnte nach der zweiten *Epoche* ein starkes *Overfitting* festgestellt werden. Eine *Epoche* bezeichnet das Bearbeiten aller Trainingsbeispiele und Anschließend einer Auswertung, bei der alle Testbeispiele durchlaufen werden und die Genauigkeit und die Größe der Fehler der Vorhersagen zusammengefasst werden. In einem Trainingsdurchlauf werden mehrere dieser *Epochen* durchlaufen, bis das Netz die gewünschte Genauigkeit sowohl im Training, als auch mit unbekannten Testdaten erreicht hat. Durch diese Aufteilung in Trainings- und Testphasen können Probleme, wie das *Overfitting* erkannt werden. *Overfitting* zeigt sich in der Auswertung daran, dass das Netz während des Trainings eine sehr hohe Genauigkeit erreicht, im anschließenden Test allerdings einen sehr großen Fehler bei den Vorhersagen macht. Eine genauere Erklärung des *Overfitting* ist in Anhang B zu finden. In den meisten Fällen ist das Netz zu komplex für die Aufgabe, die Beispiele werden „auswendig gelernt“. Es gibt verschiedene Strategien, um das *Overfitting* zu verhindern. Eine Möglichkeit wäre es, mehr Trainingsdaten zu verwenden. Eine andere besteht darin, das Problem komplexer zu machen, indem mehr Klassen verwendet werden. Diese Erkenntnisse waren entscheidend für die Auswahl des Datensets. [54]

Das Kinetics 700 Datenset besitzt die meisten Klassen und damit die größte Komplexität der Aufgabe. Gleichzeitig bietet es viele Clips pro Klasse. Bevor das Datenset ausgewählt werden konnte, musste überprüft werden, ob es passende Situationen für die Nutzung auf ROS-E enthielt. Ein großer Teil der Klassen besteht aus sportlichen Aktivitäten und Aktivitäten, die nicht in Räumen ausgeführt werden. Da diese Situationen nicht im aktuell geplanten Anwendungsbereich von ROS-E vorkommen, wurde beschlossen, ein eigenes Datenset zu erstellen. Darin sollen die Klassen aus dem Kinetics 700 Datenset ausgewählt werden, die für eine Situationserkennung für ROS-E sinnvoll sind. Die Auswahl der Klassen kann im Rahmen dieser Arbeit nicht im Detail aufgezählt werden. Eine Liste der verwendeten Klassen ist in Anhang A zu finden. Insgesamt wurden 336 Klassen ausgewählt. Die zugehörigen Daten wurden mit Hilfe eines selbst erstellten Skriptes heruntergeladen und zunächst auf einem eigenen Rechner gespeichert.

7.4 Datenaufbereitung

Nach der Datenbeschaffung sollte ein Datenset nicht verwendet werden, ohne einen Überblick über die Daten zu schaffen. Dabei sollte überprüft werden, ob die Daten die gewünschten Zusammenhänge abbilden, die das Netz lernen soll. Das Kinetics 700 Dataset besteht aus Clips, die aus YouTube-Videos ausgeschnitten wurden. Daher enthalten viele Videos eingeblendete Schrift, Effekte, Hintergrundmusik und grafische Elemente. Diese visuellen und akustischen Inhalte wird ROS-E mit der eingebauten Kamera in dieser Form nicht antreffen. In der Datenaufbereitung müssten diese unpassenden Beispiele aussortiert werden. Dies kann im Rahmen dieser Arbeit allerdings nicht durchgeführt werden, da die Anzahl der Videos zu groß ist.

7.4.1 Aufbereitung des eigenen Datasets

Der erste Schritt zum Aufbereiten des eigenen Datensets bestand darin, die zuvor heruntergeladenen Daten zu analysieren. Es wurde ein Skript erstellt, das die maximale und minimale Ausprägung der Eigenschaften Framerate, Länge und Größe der Clips ausgibt. Dabei wurden zunächst beschädigte Dateien aussortiert, die weder Bild- noch Audioinformationen enthielten. Mit Hilfe dieses Skriptes wurde anschließend das vorhandene Kinetics-Sounds Datenset analysiert, um die Grenzwerte für das Netz zu ermitteln. Dabei wurde festgestellt, dass alle Videos maximal 300 Frames besaßen. Da die Framerate auf einen angegebenen, einheitlichen Wert umgerechnet wird, bevor die Videos an das Netz übergeben werden, könnten mit einer größeren Anzahl an Frames unbekannte Probleme auftreten. Daher wurden die wenigen Clips (maximal 6 in einer Klasse), die mehr als 300 Frames besaßen, im eigenen Datenset ebenfalls aussortiert.

Ein weiteres Problem stellten die Dateien dar, denen entweder Audio- oder Bildinformationen komplett fehlten. In diesem Fall wurde jedoch entschieden, diese nicht auszusortieren, sondern mit schwarzen Pixeln bzw. Nullwerten für das Audio zu füllen, um zu simulieren, dass entweder das Mikrofon oder die Kamera ausgefallen ist.

Abschließend wurde ein sogenannter *Split* des Datensets erstellt. In einem *Split* wird eine Untermenge des Datensets zusammengefasst. Dies kann dazu dienen, nicht das gesamte Datenset, das über 600 GB groß ist, bearbeiten zu müssen. In diesem Fall wurde der *Split* einerseits erstellt, um nicht alle Daten auf den Trainingsserver übertragen zu müssen. Andererseits wurde der *Split* so erstellt, dass alle Klassen dieselbe Anzahl an Clips enthalten. Damit soll verhindert werden, dass die Wahrscheinlichkeit für die Vorhersage einer Klasse von der Häufigkeit während des Trainings abhängt. Das auf diese Weise erstellte Datenset wurde „Kinetics-ROSE-x300“ genannt und enthält in jeder Klasse 300 Dateien. Aus jeder der zuvor ausgewählten 336 Klassen des ursprünglichen Kinetics-700 Datensets wurden die ersten 300 Dateien gewählt. An dieser Stelle könnte die Qualität der Ergebnisse deutlich erhöht werden, indem die Dateien einzeln nach ihrem Inhalt ausgewählt werden. Dadurch könnten diejenigen Beispiele übernommen werden, die dem Einsatz auf ROS-E am nächsten kommen. Dazu hätten allerdings zu viele Dateien per Hand sortiert werden müssen als im Rahmen dieser Arbeit möglich war. Die Gesamtgröße des Datensets beträgt nach dieser Auswahl 133 GB.

7.4.2 Augmentation

Unter *Augmentation* wird beim *Machine Learning* das Erzeugen neuer Beispiel-Daten aus bereits vorhandenen Daten verstanden. Dies kann zum Beispiel durch horizontales oder vertikales Spiegeln, Rotieren oder Verzerren eines Bildes geschehen. Das Bild wird dabei kopiert und das neue Bild zusätzlich beim Training verwendet. Einerseits kann dadurch das Datenset vergrößert werden, ohne mehr Daten aufwändig in der echten Welt sammeln zu müssen. Andererseits kann durch das Hinzufügen von Störungen und Rauschen die Fehlertoleranz der Vorhersage verbessert werden. Für das Training im Rahmen dieser Arbeit werden die *Augmentations* des verwendeten Codes für den AVBERT übernommen. Die *Augmentations* bestehen aus dem Erstellen zufälliger Bildausschnitte (Crops) und Vergrößerungen bzw. Verkleinerungen (Resize), dem Spiegeln des Bildes (Flip) und dem Normalisieren der Farben (Color-Normalize). [55] [40]

Zusammenfassung des Kapitels

Das Training des ersten Teilnetzes wird in zwei Phasen geteilt. In der ersten Phase, dem *Pre-Training* werden mit Hilfe eines selbst erstellten Datensets, das an das Einsatzgebiet von ROS-E angepasst wurde, die grundlegenden Zusammenhänge in Video- und Audiodaten gelernt. Die zweite Phase, das *Finetuning*, muss später mit Trainingsdaten durchgeführt werden, die direkt mit ROS-E gesammelt wurden, um die Ergebnisse zu verbessern. Das *Finetuning* kann im Rahmen dieser Arbeit nicht durchgeführt werden.

Das Kinetics-700 Datenset wurde für das *Pre-Training* des AVBERT ausgewählt und beschafft. Anschließend wurde auf dessen Grundlage ein eigenes Datenset erstellt, die Daten aufbereitet und die Strategie der *Augmentation* gewählt.

8 Aufbau der Trainings- und Testumgebung

In diesem Kapitel wird der Aufbau der Trainings- und Testumgebung beschrieben, mit der die Trainingsdaten für das *Finetuning* des ersten Teilnetzes und den Reinforcement-Ansatz direkt mit ROS-E gesammelt werden können. Um diese Trainingsdaten zu sammeln, zu speichern und das Training durchzuführen, werden mehrere Komponenten benötigt. Diese Komponenten und ihre Kommunikation werden in diesem Kapitel beschrieben.

8.1 Mögliche beteiligte Komponenten

Obwohl es möglich wäre, die Situationserkennung nur mit einem Programm auf ROS-E und einer App umzusetzen, gibt es verschiedene Gründe, warum das Gesamtsystem weitere Komponenten benötigt. Diese sollen hier vorgestellt werden.

Nachfolgend wird ein kurzer Überblick über die beteiligten Komponenten, deren Hauptaufgaben und deren Zusammenspiel gegeben (siehe Abbildung 21).

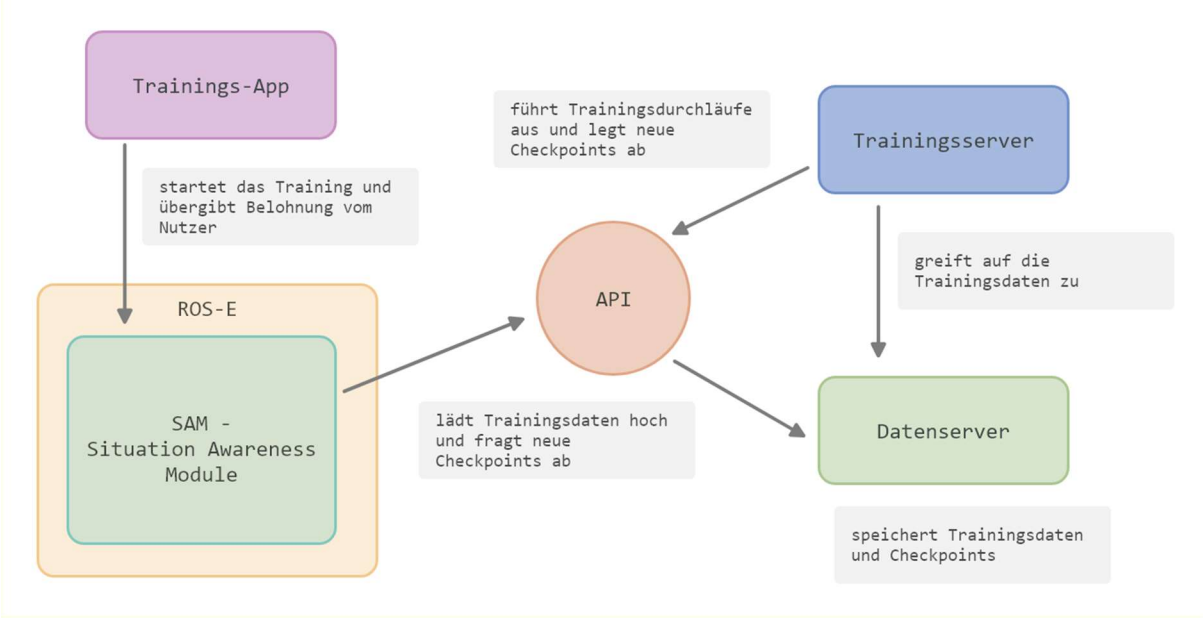

Abbildung 21: Übersicht der Komponenten der Trainings- und Testumgebung (Quelle: eigene Abbildung)

Das Situation Awareness Module (SAM): läuft auf ROS-E und ist für das Training des zweiten Teilnetzes mit dem Reinforcement-Ansatz zuständig. Dazu verwendet es eine Instanz des AVBERT mit einem aktuellen *Checkpoint* vom Training. Es greift auf die Daten der Kamera und des Mikrofons von ROS-E zu und verwaltet diese in kleinen Clips, die an den AVBERT übergeben werden können. Mit dem zurückgegebenen *Representation-Vector* und der *Policy* wählt das Modul die geeignete Aktion für die aktuelle Situation aus. Die ausgewählte Aktion wird anschließend über die anderen Module und Anwendungen auf ROS-E ausgeführt. Dieser Ablauf findet statt, wenn beide Teilnetze trainiert sind und kein weiteres Training durchgeführt wird.

Soll die *Policy* allerdings zu Beginn trainiert oder an veränderte Vorlieben des Benutzers angepasst werden, muss das System mindestens um eine Komponente erweitert werden.

Die Trainings-App: wird verwendet, um dem Benutzer die Möglichkeit zu geben, die von ROS-E ausgeführten Aktionen zu bewerten. Die *Policy*, also das Verhalten, welche Aktion in welcher Situation ausgeführt wird, wird aufgrund dieser Belohnungen angepasst. Damit jeder Nutzer die Belohnung selbst vergeben kann, wird eine Trainings-App benötigt, in der die Belohnung vergeben werden kann.

Diese beiden Komponenten würden ausreichen, um die Situationserkennung zu verwenden. Dabei wird jedoch vorausgesetzt, dass der aktuelle *Checkpoint* des AVBERT mitgeliefert und nicht aktualisiert wird. Weiterhin würde dieses System dazu führen, dass alle Benutzer die Verhaltensweise von ROS-E von einer völlig zufälligen Auswahl der Aktionen selbst trainieren müsste. Zum derzeitigen Entwicklungsstand kann nur vermutet werden, dass dieser Vorgang viel Zeit in Anspruch nehmen würde. Daher wäre es sinnvoll, eine Möglichkeit zu haben, gewisse Grundverhaltensweisen auf alle Roboter zu übertragen. Zudem könnte es sehr vorteilhaft sein, die Erfahrungen mehrerer Roboter zu einer Verhaltensweise zusammenzuführen. Wenn eine ROS-E beispielsweise für eine Zeit in einer Bibliothek trainiert wird und eine andere in einer Pflegeeinrichtung könnten beide unterschiedliche, sinnvolle Verhaltensweisen lernen. Es könnten auch gezielt Verhaltensweisen für bestimmte Einsatzgebiete erstellt werden. Nachfolgend werden daher weitere Komponenten beschrieben, die das Gesamtsystem für den späteren Einsatz erweitern.

Der Trainings-Server: führt weitere Trainingsdurchläufe auf den selbst gesammelten Daten durch, um den AVBERT weiter zu verbessern. Nach jedem Training wird ein *Checkpoint* an einer zentralen Stelle hinterlegt, sodass auf jedem Roboter die neu gelernten Gewichte des Netzes aktualisiert werden können. Die selbst gesammelten Daten bestehen nicht nur aus den Video- und Audio-Clips, die für den AVBERT benötigt werden, sondern auch aus den sogenannten *Experiences* die während des *Reinforcement Learning* gesammelt werden. So ist es möglich, dass der Trainingsserver neue *Policies* lernt, die aus ausgewählten Trainingseinheiten (z.B. von einem bestimmten Tag in der Biblio-

thek) oder auch Erfahrungen von mehreren ROS-Es zusammengestellt werden. Diese *Policies* müssen genau wie die *Checkpoints* an alle oder ausgewählte ROS-Es verteilt werden. So könnten für bestimmte Einsatzgebiete verschiedene *Checkpoints* mit angepassten *Policies* geladen werden.

Die API und die Datenbank: sind für das Sammeln der Trainingsdaten und das Verteilen der *Checkpoints* und *Policies* zuständig. Die API nimmt die Trainingsdaten der ROS-Es entgegen und speichert diese in der Datenbank.

Im Rahmen dieser Arbeit wird das Grundsystem aus dem Situation Awareness Module und der Trainings-App entwickelt, um die Ergebnisse beider Netze mit ROS-E testen zu können. Es wird dabei darauf geachtet, dass die übrigen Komponenten im Nachhinein hinzugefügt werden können. Im nächsten Abschnitt sollen die beiden geplanten Komponenten und deren Kommunikation näher beschrieben werden.

8.2 Konzept der Trainings- und Testumgebung

In diesem Abschnitt wird das Konzept für die nötigen Teile der Situationserkennung und deren Umsetzung auf ROS-E erstellt.

8.2.1 Das Situation Awareness Module

ROS-E verwendet *ROS* (Robot Operation System), um einzelne Programme zu verwalten. Jedes Programm ist dabei ein sogenannter Knoten, der über ein Nachrichtensystem mit anderen Knoten kommunizieren kann. Es gibt beispielsweise einen Knoten, der das Mikrofon ansteuert und die Audiodaten als Stream veröffentlicht. Alle anderen Knoten können auf diesen Stream lauschen und die Daten erhalten. Für die Aufgaben der Situationserkennung wird ein neuer ROS-Knoten geplant, das Situation Awareness Module (SAM). Als erstes muss das Modul auf die Daten der Kamera und des Mikrofons von ROS-E zugreifen und die Frames und Audiodaten in getrennten Arrays abspeichern. Diese Arrays können anschließend an die Instanz des AVBERT übergeben werden, um den zugehörigen *Representation-Vector* zu erhalten. Dieser Vektor wird an das zweite Teilnetz übergeben, um die passende Aktion für die aktuelle Situation zu erhalten. Nachdem eine Aktion vorgeschlagen wurde, verbindet sich der SAM-Knoten zu anderen Knoten und sorgt dafür, dass die Aktion ausgeführt wird. Ein Beispiel hierfür wäre, eine Nachricht an den Facial Expression Manager zu senden, um eine bestimmte Animation abzuspielen.

8.2.2 Die Trainings-App

Für das Training des zweiten Teils des Netzes wird ein Belohnungssystem benötigt. Damit jeder Nutzer die Belohnung selbst vergeben kann, wird eine Trainings-App benötigt, in der die Belohnung vergeben werden kann. Anhand der in Abschnitt 5.2 vorgestellten Personae sollen die wichtigsten Funktionen der App in User Stories festgehalten werden. User Stories sind eine Methode aus der agilen Softwareentwicklung und bieten die Möglichkeit, Anforderungen aus der Perspektive verschiedener Zielgruppen mit einem konkreten Ziel des Benutzers festzuhalten.

Tabelle 7: User Stories für die Trainings-App

ID	Beschreibung
U1	Thore möchte die App benutzen können, ohne viel lesen zu müssen. Andrea möchte nicht jedes Mal ihre Lesebrille holen, um die App zu verwenden.
U2	Volker möchte genau wissen, wann die Kamera und das Mikrofon an sind und ihn aufnehmen.
U3	Elfriede möchte große Buttons für die Interaktion, weil sie Schwierigkeiten hat, kleine Buttons zu treffen.
U4	Elfriede möchte eine Rückmeldung, sobald sie erfolgreich einen Button gedrückt hat, weil sie sich sonst nie sicher ist, ob sie etwas ausgelöst hat.
U5	Elias möchte die Clips sehen, die zur Verarbeitung der aktuellen Situation verwendet wurden, um die Belohnung besser einschätzen zu können.
U6	Andrea möchte möglichst wenig Auswahlmöglichkeiten für die Belohnung, weil sie sich sonst überfordert fühlt.
U7	Elias möchte möglichst viele Auswahlmöglichkeiten für die Belohnung, weil er das Training ganz genau steuern möchte.

Mit der App kann zunächst eine Trainings-Session gestartet werden. Daraufhin wartet ROS-E nach jeder ausgeführten Aktion auf eine Belohnung. Die Belohnung wird durch Tippen auf einen großen Button mit dem Wert und der entsprechenden Anzahl an Keksen gegeben (siehe U1 und U3). Die Buttons können einen Ton, eine Vibration und/oder eine Animation auslösen (siehe U4). In der linken oberen Ecke der Ansicht wird der Status der Kamera und des Mikrofons angezeigt (siehe U2). Nachdem die Trainings-Session gestartet wurde, werden die Clips angezeigt, die sich aktuell im Zwischenspeicher befinden (siehe U5).

Für die Trainings-App wurde das folgende Mockup erstellt (siehe Abbildung 22).

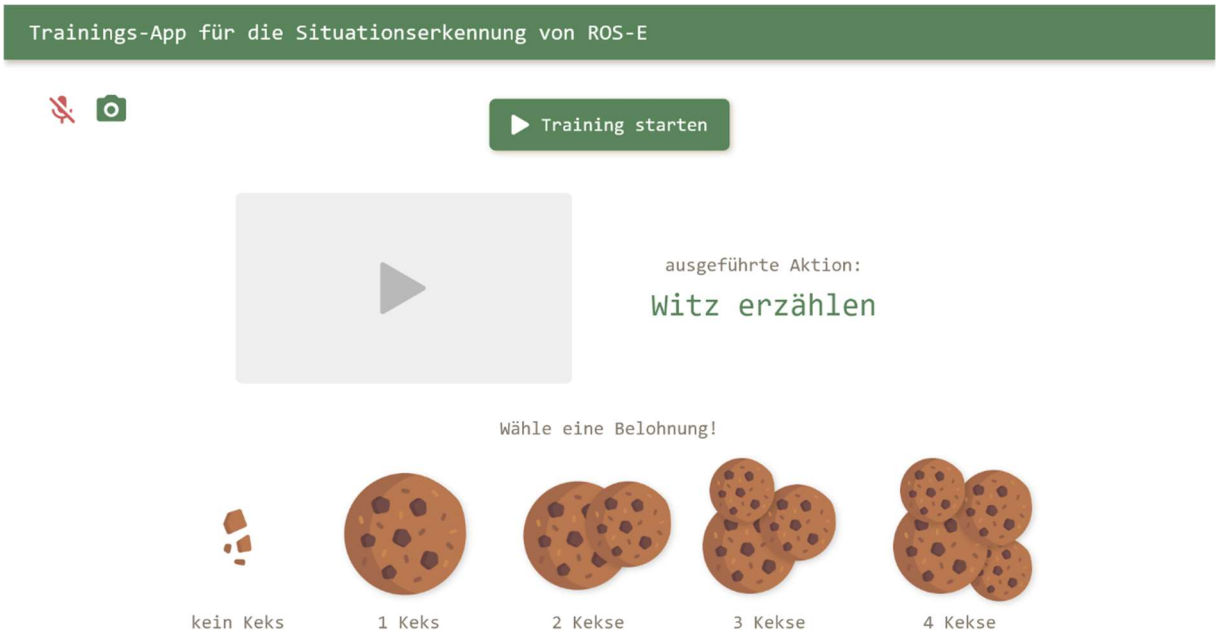

Abbildung 22: Mockup für die Trainings-App (Quelle: eigene Abbildung, Icon created by Freepik)

Die Belohnung wird aus einer festgelegten Auswahl mit den Werten 0, 1, 2, 3 und 4 (Keksen) getroffen. Diese Werte müssen nicht genauso an das Netz weitergegeben werden. Es könnte sich in späteren Tests ergeben, dass Belohnungen von Werten -0,7; -0,3; 0,1; 0,4 und 2 am besten geeignet sind, um die Zuordnung der Aktionen zu lernen. Da diese Zahlen für die Nutzer jedoch schwerer einzuschätzen oder verwirrend sein könnten, wird ein System gewählt, das einfach zu überblicken ist, aber genügend Abstufungen enthält (siehe U6 und U7). Diese fünf Stufen sind heutzutage in Bewertungssystemen z.B. die Sterne bei Amazon oder Fragebögen mit den Optionen „stimme gar nicht zu“, „stimme eher nicht zu“, „unentschlossen“, „stimme eher zu“ und „stimme voll zu“ häufig anzutreffen. Damit der Trainingsprozess aufgelockert wird und auch Kinder die App verwenden können, wurde zusätzlich zur Bewertung anhand von Zahlen die Belohnung in Form von Keksen dargestellt (siehe U1).

Später kann diese App erweitert werden, indem der Nutzer entscheiden kann, welche Daten er zum Server senden möchte, ungewünschte Erfahrungen aussortieren kann und die Trainings-Session zu benennen, sodass sie gezielt für Trainingsdurchläufe verwendet werden kann.

8.2.3 Ablauf des Trainings

Der Ablauf der Kommunikation zwischen den Teilnetzen, dem Situation Awareness Module und der Trainings-App soll in diesem Abschnitt noch einmal genau aufgeführt werden:

1. Der Benutzer startet über die App eine Trainingssession.
2. Der SAM-Knoten bekommt über eine Schnittstelle die Nachricht von der App.
3. Der SAM-Knoten startet den Trainings-Modus:
    - Der SAM-Knoten greift jederzeit den Audio-Stream und die einzelnen Bilder von der Kamera ab und fügt beide zu jeweils einem Array zusammen.
    - Die letzten x Sekunden der Video- und Audioaufzeichnung werden mit Hilfe des AVBERT in einen Vektor umgewandelt, der an das *Environment* übergeben wird.
    - Der *Agent* macht mit Hilfe der aktuellen *Policy* eine Vorhersage für eine Aktion basierend auf dem letzten Vektor.
    - Der SAM-Knoten sorgt dafür, dass die gewählte Aktion ausgeführt wird.
    - Der *Agent* fragt den nächsten Vektor und die Belohnung für die ausgewählte Aktion vom *Environment* ab.
    - Das *Environment* berechnet den nächsten Vektor und wartet bis die Belohnung von der Trainings-App für die Aktion übermittelt wurde.
    - Der Nutzer bewertet die Aktion.
    - Die Trainings-App gibt den Wert an das *Environment* zurück.
    - Das *Environment* gibt den nächsten *State* und die Belohnung zurück an den *Agent*.
    - Der *Agent* speichert die *Experience* in seinem Cache.
    - Der *Agent* führt einen weiteren Schritt im Training der einen Instanz des *MLP* aus.
4. Der Ablauf von Punkt 3 wird wiederholt, bis der Nutzer die Trainingssession beendet.

Dieser Ablauf wird leicht verändert ausgeführt, wenn die Situationserkennung aktiv ist, aber kein Training durchgeführt wird:

1. Der SAM-Knoten greift jederzeit den Audio-Stream und die einzelnen Bilder von der Kamera ab und fügt beide zu jeweils einem Array zusammen.
2. Die letzten x Sekunden der Video- und Audioaufzeichnung werden mit Hilfe des AVBERT in einen Vektor umgewandelt, der an das *Environment* übergeben wird.

3. Der *Agent* macht mit Hilfe der aktuellen *Policy* eine Vorhersage für eine Aktion basierend auf dem letzten Vektor.
4. Der SAM-Knoten sorgt dafür, dass die gewählte Aktion ausgeführt wird.
5. Der *Agent* fragt den nächsten Vektor für die ausgewählte Aktion vom *Environment* ab.
6. Das *Environment* berechnet den nächsten Vektor.
7. Das *Environment* gibt den nächsten *State* zurück an den *Agent*.

Neben den *Experiences*, die eine einzelne Iteration der Auswertung abbilden, sind die sogenannten *Episoden* eine weitere wichtige Größe für das Training. In einer *Episode* werden mehrere aufeinanderfolgende *Experiences* zusammengefasst, die am Ende ein größeres Ziel erreichen. Ein solches Ziel könnte sein, ein Spiel zu gewinnen oder einen Gegenstand mit einem Roboterarm erfolgreich hochzuheben. Die Einteilung in *Episoden* kann entscheidend für das Lernen sein, da die Zwischenschritte häufig keine direkte, große Belohnung erzielen, aber dennoch entscheidend für die Höhe der Belohnung am Ziel sein können. Ein geeignetes Ziel zu definieren, ist für ROS-E nicht einfach. Somit ist es auch keine leichte Aufgabe, die *Experiences* in *Episoden* einzuteilen. Da die Einteilung nicht zwingend notwendig ist, um das Gesamtkonzept zu testen, kann eine Umsetzung für dieses höhere Ziel auch später erfolgen, nachdem Erfahrungen mit dem Training gesammelt werden konnten. Das Problem sollte dennoch festgehalten werden, da es ein entscheidender Faktor für den Erfolg des Trainings darstellen kann. [13]

Zusammenfassung des Kapitels

Für die Verwendung der Situationserkennung auf ROS-E werden die beiden minimal erforderlichen Komponenten der beschriebenen Trainings- und Testumgebung umgesetzt. Diese bestehen aus einem ROS-Knoten, dem Situation Awareness Module (SAM), der auf ROS-E ausgeführt wird. Er ist dafür zuständig, den *Representation-Vector* zu verwenden, um mit Hilfe des Reinforcement-Ansatzes die Aktionen für die aktuelle Situation auszuführen. Zudem ist er für das Training des zweiten Teilnetzes direkt auf ROS-E zuständig. Um dieses Training durchzuführen, können die Benutzer mit Hilfe der zweiten Komponente, einer Trainings-App, selbst die Belohnung für die ausgeführten Aktionen vergeben.

9 Implementierung

In diesem Kapitel wird zunächst beschrieben, wie das erste Teilnetz, der AVBERT konfiguriert, trainiert und erweitert wurde. Anschließend wird die Implementierung der in Abschnitt 6.4.2 gewählten Methode des *Reinforcement Learning*, des *DQN* erläutert. Nachdem die beiden Teilnetze erstellt wurden, wird gezeigt, wie diese auf ROS-E übertragen und getestet wurden.

9.1 Die Umsetzung des AVBERT

Die Architektur des AVBERT wurde 2021 in einem Paper vorgestellt, an dem drei Angehörige der Seoul National University, zwei Mitarbeiter von *NVIDIA* Research und ein Mitarbeiter von Microsoft Research beteiligt waren. Der Quellcode für das Netz und dessen Training wurde bei *GitHub* veröffentlicht, damit andere Forscher (oder Interessierte) die Ergebnisse prüfen können. Für die Programmierung und das Training wurde das *Machine Learning* Framework *PyTorch* verwendet. Dieses Open-Source-Framework verwendet die Programmiersprache Python. [39] [40]

9.1.1 Die Konfiguration

Der Code ist entsprechend der am Projekt beteiligten Einrichtungen sehr komplex und bedurfte einiger Einarbeitungszeit. Es wurde jedoch von den Entwicklern eine umfangreiche Konfigurationsdatei vorbereitet, sodass eigene Tests sehr gut unterstützt werden. In dieser Datei gibt es drei große Kategorien von Einstellungen, die vorgenommen werden können: die Einstellungen für den Aufbau des Netzes selbst, die Parameter des Trainings und die Einstellungen für das Vorverarbeiten und Laden der Trainingsdaten. Anhand dieser Konfigurationsdatei war es als erstes möglich, das Netz auf seine minimal mögliche Größe zu reduzieren.

Der AVBERT ist in drei große Teilnetze geteilt. Die ersten beiden sind für die getrennte Verarbeitung der Modalitäten (Bilder und Audio) zuständig, bevor diese von einem dritten Teilnetz zusammengeführt werden. Die ersten beiden Teilnetze verwenden ein *Residual Neural Network* (ResNet), um die wichtigsten Informationen aus den Bildern bzw. *Spektrogrammen* abzubilden. Das dritte Netz verwendet die *Transformer*-Architektur um die Probleme der *Fusion* und des *Alignments* zu lösen und den *Attention-Mechanismus* um die Eingangsdaten zu priorisieren. Die beiden *ResNets* konnten auf 18 Schichten reduziert werden, womit deren Aufbau dem ResNet-18 aus dem Paper entspricht, in dem diese Architektur vorgestellt wurde. [56]

Der *Transformer* wurde aus dem mitgelieferten Beispiel für das Kinetics-Sounds Datensets (ebenfalls ein *Split* des Kinetics-700 Datensets) übernommen. Es wurden lediglich die Anzahl der sogenannten

Attention-Heads und die Anzahl der *Attention-Layer* reduziert. Die Anzahl der *Attention-Heads* gibt an, wie viel Mal der *Transformer* unterschiedliche Prioritäten für die Input-Daten vergibt. Das heißt, dass der *Transformer* aus mehreren Perspektiven betrachtet, welche Input-Daten am wichtigsten sind. Einer dieser Heads könnte vereinfacht dargestellt, darauf spezialisiert sein, Hunde und Katzen zu unterscheiden. Ein anderer könnte einen bestimmten Menschen erkennen. In der Praxis werden diese Perspektiven selbst gelernt und können abstrakte mathematische Zusammenhänge darstellen, die ein Mensch schwer verstehen könnte. Welche Perspektiven gelernt werden, hängt von der bearbeiteten Aufgabe während des Trainings ab. Lautet die Aufgabe, bestimmte Tierarten zu erkennen, könnte es hilfreich sein, Katzen (und Katzenartige) von Hunden zu unterscheiden. Die Anzahl der *Attention-Heads* hat jedoch einen großen Einfluss auf die Größe des *Transformers*, da dort die Anzahl der durchzuführenden Rechenoperationen vervielfacht wird. Die Anzahl wurde im Rahmen dieser Arbeit auf 8 gesetzt, was der Zahl der Heads aus dem ursprünglichen Paper der *Transformer* entspricht. Die Anzahl der *Attention-Layer* gibt an, wie viele dieser Schichten aus *Attention-Heads* hintereinander durchlaufen werden. Auch dieser Parameter beeinflusst die Größe des *Transformers* und wurde daher auf 4 gesetzt. [31]

Für den späteren Einsatz könnte die beste Auswahl der Parameter durch weitere Trainingsdurchläufe ermittelt werden. Im Rahmen dieser Arbeit war es aus Zeitgründen jedoch nicht möglich, die Auswahl anhand vieler Trainingsdurchläufe zu treffen.

Nachdem die oben beschriebenen Anpassungen vorgenommen wurden, benötigte der *Transformer* zwischen 2 und 3 GB Speicher in der GPU. Dieser Wert erfüllt die Vorgaben für den vorhandenen Speicherplatz auf ROS-E (siehe Abschnitt 6.1). Mit der Anpassung konnte jedoch der *Checkpoint*, der im *GitHub* des AVBERT für die ursprüngliche Konfiguration zur Verfügung gestellt wurde, nicht mehr verwendet werden und das Netz musste somit von Grund auf selbst trainiert werden.

9.1.2 Das Training

In diesem Kapitel wird beschrieben, wie das Training des AVBERT vorbereitet und durchgeführt wurde. Die Ergebnisse des Trainings werden anschließend vorgestellt und ausgewertet.

Um den angepassten AVBERT selbst mit dem eigenen Datenset trainieren zu können, musste zunächst ein DataLoader für das eigene Datenset geschrieben werden. Ein DataLoader ist eine Klasse, die von der gleichnamigen Klasse aus *PyTorch* erbt. Darin wird festgelegt, wie die Daten an das Netz übergeben werden und welche Vorverarbeitungsschritte stattfinden. In diesem Fall wurde der DataLoader für das Kinetics-Sounds Datenset angepasst. Darin war bereits das Einlesen der .mp4-Dateien in Frame-Arrays und *Spektrogramm*-Arrays vorgenommen. An dieser Stelle wurde auch die

Augmentation der Daten vorgenommen. Das heißt, dass aus einem Video-Clip zehn kürzere Clips mit zufälligen Start- und Endpunkten geschnitten wurden. Außerdem wurden die Bilder Zugeschnitten, in der Größe verändert und gedreht. Diese *Augmentation* sorgt neben der Vervielfachung der Trainingsdaten vor allem dafür, dass das Netz unanfälliger gegen Störungen wird. Sollte ROS-E beispielsweise einmal auf der Seite liegen, kann die Situationserkennung trotzdem funktionieren, wenn das Netz zusätzlich mit rotierten Bildern trainiert wurde. [55]

Der DataLoader wurde um die Funktion erweitert, nicht vorhandene Video- oder Audiospuren mit Nullwerten zu ersetzen. Im späteren Einsatz könnte es vorkommen, dass die Kamera oder das Mikrofon von ROS-E kurzzeitig ausfallen oder nicht eingeschaltet werden. Ohne diese Erweiterung würde es zu einem Fehler kommen und die Ausführung des Programms abgebrochen werden. Theoretisch könnte der AVBERT mit dieser Erweiterung ohne Kamera und Mikrofon verwendet werden. Da auf diese Weise jedoch keine sinnvollen Ergebnisse erwartet werden können, muss später geprüft werden, ob beide Eingangskanäle ausgefallen sind und eine Warnung gegeben werden.

Die ersten Trainingsversuche wurden auf einem eigenen Rechner mit 2 GPUs (NVIDIA GeForce GTX 1080 und NVIDIA GeForce GTX 1070), einer Intel Core i7-7800X CPU und 32 GB DDR3 Arbeitsspeicher durchgeführt. Eine *Epoche* konnte nach ca. 36 Stunden abgeschlossen werden. Während des Trainings wird aufgrund der aktuell benötigten Zeit eine Hochrechnung angezeigt, wie viel Zeit für die gesamten 20 *Epochen* benötigt werden würde. Diese schwankte um einen Wert von 22 Tagen. Daraufhin wurde nach einem alternativen Server mit mehr GPU-Leistung gesucht. Die ausführliche Beschreibung aller Trainingsversuche und das genaue Vorgehen ist in Anhang C aufgeführt. Der Server auf dem letztendlich das Training durchgeführt werden konnte, wurde von der TH Wildau zur Verfügung gestellt. Zuerst mussten die 133 GB der Trainingsdaten auf den Server geladen werden. Dieser Vorgang nahm mehr als 24 Stunden in Anspruch. Anschließend wurden alle nötigen Python-Pakete in einer virtuellen Umgebung in einem Docker-Container installiert und das Training mit Hilfe der Linux-Funktion „nohup“ als Job gestartet. So konnte das Training auch nach der Abmeldung vom Server weiter ausgeführt werden. Der Server besaß 6 NVIDIA Tesla V100 GPUs mit je 32 GB RAM, eine Intel Xeon Gold 5218 CPU und 376 GB Arbeitsspeicher. Insgesamt dauerte das Training 6 Tage und 6 Stunden. Es wurden 20 *Epochen* durchlaufen.

Während des Trainings werden mit Hilfe des SummaryWriters des Python-Paketes *TensorBoard* bestimmte Daten über den Zustand des Trainings gesammelt und in eine Datei geschrieben. Die während des Trainings aufzuzeichnenden Daten, die sogenannte Metriken, können festgelegt werden, wurden für das Training aber unverändert übernommen. Zwei der aussagekräftigsten Metriken sind der *Loss* (Verlust) und die *Accuracy* (Genauigkeit). Der *Loss* gibt an, wie groß der Fehler

für ein einzelnes Beispiel war. Besitzt der *Loss* den Wert 0, hat das Netz keinen Fehler gemacht. Dabei können verschiedene Funktionen verwendet werden, um diesen Fehler zu ermitteln. Im Rahmen des Trainings wurde der sogenannte *Cross-Entropy Loss* verwendet. Dieser ist besonders für Klassifizierungsprobleme geeignet und besitzt die Eigenschaft, dass falsche Voraussagen, die aber eine hohe Wahrscheinlichkeit haben, besonders stark bewertet (bestraft) werden. Zum Beispiel sagt das Netz mit 80,6 % Wahrscheinlichkeit voraus, dass die aktuelle Situation „singen" ist, die korrekte Klasse wäre aber „einen Hund bürsten". In diesem Fall würde der *Loss* sehr hoch berechnet werden. Wäre die korrekte Klasse „ein Gedicht vortragen" erkannt worden oder würde „singen" nur mit 40 % Wahrscheinlichkeit erkannt werden, wäre der *Loss* geringer. [13]

Die Datei, in der diese Werte gespeichert werden, kann in einem *TensorBoard* grafisch aufbereitet werden. *TensorBoard* ist ein Toolkit für die Visualisierung dieser Log-Dateien und wurde von *TensorFlow* entwickelt. *TensorFlow* ist genau wie *PyTorch* ein Framework für *Machine Learning*, das auf Python basiert. [57]

Die folgenden Diagramme wurden von *TensorBoard* für das durchgeführte Training erstellt.

Abbildung 23: Verlauf der Loss-Werte (Verlustes) über die 20 Trainingsepochen (Quelle: eigene Abbildung, erstellt mit TensorBoard)

Auf der x-Achse werden die *Epochen* des Trainings dargestellt, während die y-Achse den Wert der *Loss*-Funktion, gemittelt über alle Werte der *Epoche* (für jedes Trainingsbeispiel ein Wert), zeigt. Zu Beginn des Trainings fällt die *Loss*-Kurve stark ab, wobei sich diese Steigung über den Verlauf des Trainings immer weiter verringert. Nach 20 *Epochen* lag der durchschnittliche *Loss* im Training bei 3,8. Die *Loss*-Werte haben keine festen Werte für gute oder schlechte Ergebnisse. Je näher an 0 der Wert liegt, desto weniger Fehler macht das Netz. Darüber hinaus wird dieser Wert verwendet, um Probleme zu erkennen, falls diese Kurve stark von der hier aufgeführten Form abweicht. In diesem Fall kann die Kurve als gutes Ergebnis eingeschätzt werden.

Als nächstes werden die Werte für die *Accuracy*, die Genauigkeit der Vorhersagen, ausgewertet. Das Netz gibt bei einer Klassifizierung für jede Klasse („singen“, „tanzen“, „trinken“, …) eine Wahrscheinlichkeit aus, dass diese Klasse auf das aktuelle Video zutrifft. Für ein Video von einem tanzenden Menschen könnten die Wahrscheinlichkeiten „singen“ = 60 %, „tanzen“ = 80 % und „trinken“ = 25 % ausgegeben werden. Als Vorhersage des Netzes wird meist die Klasse mit der höchsten Wahrscheinlichkeit ausgewählt. Die *Accuracy* gibt an, mit welcher Wahrscheinlichkeit das Netz eine korrekte Antwort gibt, mit anderen Worten wie viele der Vorhersagen bezogen auf alle gemachten Vorhersagen richtig waren. Die *Accuracy* wird besonders bei Klassifizierungsaufgaben mit sehr vielen Klassen, die sich durchaus ähneln (wie z.B. das Spielen ähnlicher Instrumente), in zwei Metriken unterteilt. Die Top-1-Accuracy gibt die Genauigkeit wie eben beschrieben an. Die Top-5-Accuracy gibt an, wie oft unter den 5 Klassen mit der höchsten Wahrscheinlichkeit in einer Vorhersage die richtige Antwort enthalten ist. Wenn das gezeigte Video einen klavierspielenden Menschen enthält und die Vorhersage des Netzes zum Beispiel wie folgt aussieht „Klavier spielen“ = 70 %, „Keyboard spielen“ = 80 %, „Orgel spielen“ = 60 %, „Cembalo spielen“ = 50 %., „Schlagzeug spielen“ = 20 %, zählt diese Antwort ebenfalls als korrekt. Diese Metriken bilden ab, wie genau die Vorhersagen des Netzes sind, aber auch wie oft das Netz „fast richtige“ Vorhersagen macht. Die Diagramme dieser beiden Metriken ist in Abbildung 24 zu sehen. [58] [13]

Abbildung 24: Der Verlauf der Accuracy (Genauigkeit) währen des Trainings (Quelle: eigene Abbildung, erstellt mit TensorBoard)

Auch in diesem Diagramm werden auf der x-Achse die *Epochen* des Trainings dargestellt. Auf der y-Achse ist die durchschnittliche *Accuracy* für jede *Epoche* in Prozent angegeben. Ähnlich wie beim *Loss* steigt die *Accuracy* zu Beginn des Trainings stark an, während mit zunehmender Anzahl der *Epochen* die Steigung abnimmt, bis die Genauigkeit nach einer gewissen Anzahl an *Epochen* kaum noch verbessert wird. An diesem Punkt kann entschieden werden, ob es sich lohnt, das Training weiterzuführen oder zu beenden. Nach 20 *Epochen* lag der Wert der Top-1-Accuracy bei 55.77 % und die Top-

5-Accuracy bei 81.47 %. Da ein eigenes Datenset verwendet wurde, sind keine direkten Vergleichswerte vom Training anderer Netze vorhanden. Um das Ergebnis trotzdem einschätzen zu können, werden die Ergebnisse des AVBERT für das Kinetics-Sounds Datenset aufgeführt. Innerhalb der Bearbeitungszeit für das Paper konnte der AVBERT mit den dort ermittelten Konfigurationen eine Top-1-Accuracy von 75.6 % und eine Top-5-Accuracy von 94.6 % erreichen. Später wurde das Netz weiter optimiert und neuere Ergebnisse auf *GitHub* veröffentlicht. Dort werden eine Top-1-Accuracy von 85.8 % und eine Top-5-Accuracy von 97.8 % angegeben. Bei diesen Werten ist zu beachten, dass das dort verwendete Netz mehr Schichten besitzt, ein anderes *Pre-Training* durchlaufen hat und der verwendete *Classification-Head* (das angehängte Netz, das den *Representation-Vector* verwendet) alleine für die Klassifizierung der (nur) 32 Klassen des Kinetics-Sounds Datensets zuständig ist. Ein direkter Vergleich der Werte ist daher nicht sinnvoll. Das Beispiel soll lediglich zeigen, welche Werte von derzeit entwickelten Netzen als sehr gute Ergebnisse für die gegebene Aufgabe der Klassifizierung von Videos gelten. Die Netze werden in solchen Papers zu dem Zweck entwickelt, die bisherigen Ergebnisse für bestimmte Datensets zu übertreffen. Daher sind diese nicht in ihrer Größe oder Rechenleistung eingeschränkt. Es ist demnach zu erwarten, dass das für ROS-E verkleinerte Netz keine Genauigkeit von 80 % erreichen wird. Die Top-1-Accuracy von 55.77 % ist dennoch verbesserungswürdig, da dies bedeutet, dass das Netz bei ungefähr jeder zweiten Vorhersage falsch liegt. [39] [40]

Ob das Training erfolgreich war, kann vor allem am Vergleich der *Loss*- und *Accuracy*-Metriken zwischen den Trainings- und Testiterationen innerhalb einer *Epoche* erkannt werden. Jede *Epoche* besteht aus einer Trainingsiteration, in der das Netz anhand aller Beispiele aus dem Train-*Split* des Datensets trainiert wird. Anschließend wird eine Testiteration durchgeführt, in der das Netz alle Beispiele aus dem Test-*Split* einschätzen soll, aber die Gewichte nicht angepasst werden. Durch dieses Vorgehen kann erkannt werden, ob das Netz eine Funktion lernt, die auch auf nie zuvor gesehene Testdaten zutrifft oder ob es bei diesen unbekannten Daten, wie sie auch später im Einsatz auftreten werden, plötzlich sehr viel schlechtere Vorhersagen trifft. Für das Training im Rahmen dieser Arbeit sollen die *Accuracy*-Werte zwischen Training und Test verglichen werden. Die folgende Abbildung zeigt den Verlauf der *Accuracy*-Werte während der Testiterationen.

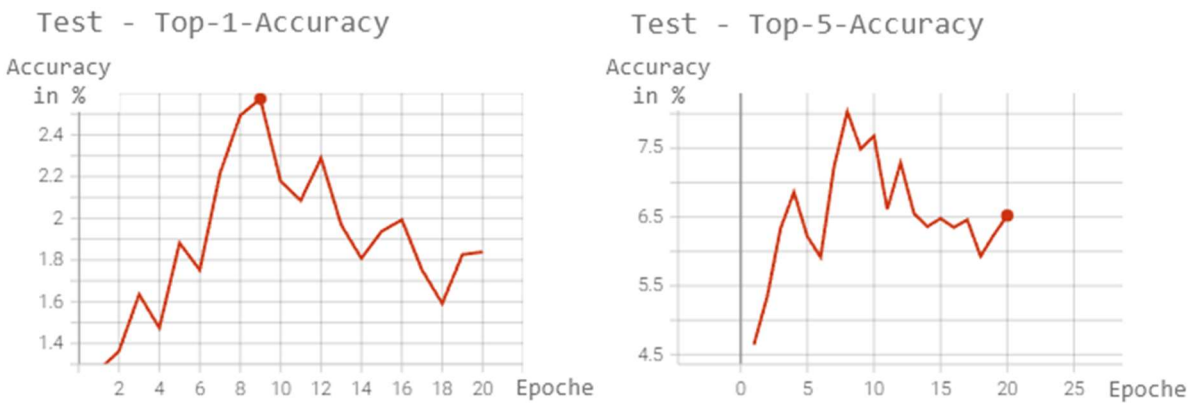

Abbildung 25: Der Verlauf der Accuracy (Genauigkeit) währen des Tests (Quelle: eigene Abbildung, erstellt mit TensorBoard)

Auf der x-Achse werden die *Epochen* des Trainings dargestellt, auf der y-Achse ist die durchschnittliche *Accuracy* für jede *Epoche* in Prozent angegeben. Diese Diagramme folgen nicht dem typischen Verlauf wie während des Tests. Es sind große Sprünge zu sehen und die *Accuracy* fällt mit zunehmender Anzahl *Epochen* wieder ab, nachdem sie einen Höchstwert von 2,6 % der Top-1-Accuracy in *Epoche* 9 und einen Höchstwert von 8 % für die Top-5-Accuracy in *Epoche* 8 erreichte. Dabei sind die absoluten Werte der *Accuracy* entscheidender als der Verlauf der Kurve. Die Werte sind extrem niedrig, was für ein starkes *Overfitting* spricht. Das Konzept des *Overfitting* wird in Anhang B genauer veranschaulicht. Diese Auswertung zeigt, dass das Netz zwar in der Lage ist, die Zusammenhänge abzubilden, diese aber nicht generalisieren, also auf unbekannte Daten übertragen kann. Der nach dem Training vorhandene *Checkpoint* ist für die weitere Verwendung nicht geeignet, da er nur sehr schlechte Ergebnisse im Einsatz auf ROS-E erzielen kann. [13]

Für die Einschätzung der Trainingsergebnisse kann festgehalten werden, dass die Wahl der Aufgabe für das *Pre-Training* nicht geeignet war. Die Klassifizierung ist selbst mit einer Anzahl von 336 Klassen eine zu einfache Aufgabe für den sehr komplexen Aufbau des AVBERT. Diese Klassifizierung wurde im Paper mit einem *Classification-Head* durchgeführt, einem *Multi-Layer Perceptron* mit nur drei Schichten. Da der Code für die im Paper verwendeten Trainingsaufgaben für den AVBERT nicht veröffentlicht wurden und in der gegebenen Zeit nicht selbst implementiert werden konnten, wird es eine Aufgabe für die Zukunft sein, diese zu implementieren und das Training zu wiederholen. Der Vorteil, den die dort verwendeten Aufgaben bringen, ist die Möglichkeit, das Training *unsupervised* durchzuführen. Das bedeutet, dass die Trainingsdaten, die später mit ROS-E gesammelt werden nicht per Hand in Klassen eingeteilt werden müssen, sondern direkt verwendet werden können. Eine weitere Möglichkeit, das *Overfitting* zu verhindern, könnte die Implementierung weiterer Methoden zur *Augmentation* der Daten und die Verwendung des kompletten Kinetics-ROS-E Datensets ohne die Reduzierung auf 300 Beispiele für jede Klassen sein.

Obwohl im Rahmen dieser Arbeit kein guter *Checkpoint* trainiert werden konnte, wurde der Code für das Netz trotzdem weiterverwendet. Das Training kann in Zukunft wiederholt und ein guter *Checkpoint* im Nachhinein hinzugefügt werden. In nächsten Abschnitt wird beschrieben welche Schritte notwendig waren, um den AVBERT auf ROS-E zu übertragen.

9.1.3 Das Umschreiben für die Verwendung auf ROS-E

Ein großer Teil des vorhandenen Codes wurde ausschließlich für das Training des AVBERT geschrieben. Besonders für die effiziente Verteilung der Daten zwischen mehreren GPUs wurden viele zusätzliche Klassen verwendet, die im Einsatz auf ROS-E nicht mehr benötigt werden. Daher wurde der von *GitHub* heruntergeladene Code vereinfacht und umgeschrieben.

Im ersten Schritt wurde eine eigene Klasse (ROSETransformer) angelegt, die eine Instanz des AVBERT erstellt und einen aktuellen *Checkpoint* einliest. Daraufhin konnten anhand der Imports alle Abhängigkeiten zu den benötigten Klassen aufgelöst werden. Alle nicht mehr benötigten Klassen konnten entfernt werden. Die übrig gebliebenen Klassen wurden zu einem eigenen Python-Package zusammengefügt. Die im ersten Schritt erstellte Klasse wurde als Einstiegspunkt für dieses Paket festgelegt. So können später verschiedene ROS-Knoten diese Klasse verwenden, um Video- und Audio-Clips an den AVBERT zu übergeben und einen *Representation-Vector* zu erhalten. Die Clips werden in Form von numpy-Arrays übergeben und müssen ähnlich wie vor dem Training aufbereitet werden, bevor sie an den AVBERT übergeben werden. Dies geschieht ebenfalls in der erstellten Klasse. Es werden jedoch keine zufälligen Ausschnitte mehr erstellt und keine *Augmentation* (z.B. drehen oder spiegeln der Bilder) vorgenommen.

Das fertige Python-Package konnte auf ROS-E übertragen werden.

9.2 Die Umsetzung des DQNs

Wie in Abschnitt 6.4.2 beschrieben, wurde der Code für die Umsetzung des *Double DQN*-Algorithmus von einem Tutorial auf der offiziellen *PyTorch*-Seite verwendet und angepasst. Die wichtigsten Klassen sollen kurz beschrieben werden. [42]

Der ROSEAgent: enthält die Methoden act(), cache(), recall() und learn(), die dafür sorgen, dass Aktionen mit Hilfe der *Policy* gewählt, *Experiences* gespeichert und wieder abgerufen und die beiden Netze nach dem *Double DQN*-Ansatz trainiert werden können. Zusätzlich werden Helfer-Methoden für das Laden von *Checkpoints* und das Synchronisieren der beiden Netze bereitgestellt. Der Code für diese Klasse implementiert den grundlegenden Algorithmus und musste nicht verändert werden.

Der ROSENet: erstellt den Aufbau der beiden Netze mit den korrekten Input- und Output-Dimensionen. Die Input-Dimension muss dem *State* des *Environments*, also dem *Representation-Vector* entsprechen und die Output-Dimension der Anzahl der möglichen Aktionen.

In dieser Klasse wurde die Architektur des Netzes angepasst, sodass dieses der Definition aus Abschnitt 6.4.2 entspricht. Da mit dem Netz keine Bilder mehr ausgewertet werden sollen, werden die Convolutional-Layer gegen einen Linear-Layer mit 128 Knoten ausgetauscht. Dieser Wert muss später im Test angepasst werden, falls die Ergebnisse nicht ausreichen.

Das Environment: verwendet das eigene Package mit der ROSETransformer-Klasse, um einen Vektor vom AVBERT für die aktuellen Video- und Audio-Clips zu erhalten. Dieser Vektor bildet den *State* des *Environments* ab. Zusätzlich liefert das *Environment* die Belohnung für die ausgewählte Aktion und entscheidet, ob eine *Episode* abgeschlossen ist.

Diese Klasse musste selbst hinzugefügt werden. Das ursprünglich verwendete *Environment* war ein Spiel, das über das Toolkit „Gym“ von *OpenAI* verwendet wurde, ohne die Belohnung oder den *State* selbst erzeugen zu müssen. [59] [60]

Das Training: wird in einer Schleife ausgeführt, solange eine Variable gesetzt ist, die später über die App verändert werden soll. Während das Training aktiv ist wird eine Aktion für den aktuellen *State* ausgewählt, diese Aktion später von ROS-E ausgeführt, der nächste Schritt vom *Environment* abgefragt, die *Experience* im Cache hinterlegt und zum Lernen verwendet. Ist die Variable nicht bzw. auf „False“ gesetzt, werden die *Experiences* in einer alternativen Schleife nicht gespeichert und das Netz nicht trainiert.

Dieser Code wurde auf einem eigenen Rechner mit GPU und *CUDA*-Unterstützung und einer funktionsfähigen Installation von *PyTorch* erfolgreich getestet. Dabei wurde das Training allerdings mit zufällig erzeugten Vektoren und Belohnungen durchlaufen, um zu sehen, ob Fehler auftreten. Es konnte bei diesem Test noch nicht nachgewiesen werden, dass das Training sinnvolle Ergebnisse liefert. Um dies zu testen sollte der Code auf ROS-E übertragen und dort das Training durchgeführt werden. Im nächsten Abschnitt wird die Umsetzung auf ROS-E beschrieben.

9.3 Die Trainings- und Testumgebung

Nachdem die beiden Teilnetze auf einem Rechner getestet wurden, der vergleichsweise viel Rechenleistung und eine funktionierende Installation von *PyTorch* mit GPU-Unterstützung besaß, mussten die Netze auf ROS-E übertragen werden. ROS-E verwendet den Raspberry Pi 4 mit 8 GB RAM und einer CPU mit 4 Kernen. Dieser besitzt eine sehr kleine GPU, die nicht für das Laden und Ausführen

von Netzen ausgelegt ist. Daher müssen auf ROS-E die CPU und deren Arbeitsspeicher verwendet werden, um die Netze zu nutzen. Da *PyTorch* darauf spezialisiert ist, den Code auf GPUs mit *CUDA* (Compute Unified Device Architecture), einem von *NVIDIA* entwickelten Standard für GPUs, auszuführen, gibt es einige Funktionen, die nur mit einer *CUDA*-kompatiblen GPU verwendet werden können. Diese Funktionen mussten ausgetauscht werden. [61]

9.3.1 Übertragen der Teilnetze auf ROS-E

Als erster Schritt wurde der Code für beide Netze auf ROS-E übertragen. Anschließend wurden alle Python-Pakete installiert, die für die Ausführung der Netze und die Umwandlung der Video- und Audiodaten benötigt wurden. Die drei wichtigsten Pakte waren torch, torchaudio und torchvision. Diese Pakete wurden über *pip* installiert und anschließend das Test-Skript für die Situationserkennung aufgerufen. Dieses Skript erstellt eine Instanz des AVBERT und verwendet diesen, um zunächst mit Nullwerten gefüllte Video- und Audio-Clips zu übergeben. So kann ohne weiteren Aufwand ein Durchlauf einer Vorhersage getestet werden. Anschließend wird der Vektor vom zweiten Teilnetz verwendet, um eine Aktion auszugeben.

Beim Start dieses Skriptes trat der folgende Fehler ohne weitere Informationen oder die Aufrufhierarchie auf:

```
Illegal Instruction (core dumped)
```

Da ohne weitere Informationen nicht ermittelt werden konnte, wo dieser Fehler auftrat oder wodurch er ausgelöst wurde, musste das Skript mit Hilfe von gdb, einem Debugging-Tool von Linux, analysiert werden. Auf diese Weise konnte eine genauere Fehlermeldung gewonnen werden:

```
Thread 1 "python" received signal SIGILL, Illegal instruction.

0x0000fffff3e40a78 in exec_blas () from /usr/local/lib/python3.8/dist-
packages/torch/lib/libtorch_cpu.so
```

Auf der *GitHub*-Seite von *PyTorch* wurde am 23. Dezember 2021 genau diese Fehlermeldung als Problem (Issue) gemeldet, jedoch bis kurz vor dem Ende der Bearbeitungszeit keine Lösung dafür gefunden. Da das Problem mit der *ARM-Architektur* des Raspberry Pi zusammenhängt, wurde während der Untersuchung dieses Fehlers festgestellt, dass *PyTorch* für den Raspberry speziell gebaut werden muss und es keine offizielle Version für die *ARM-Architektur* gibt. Im Internet sind jedoch einige Quellen für selbst gebaute Versionen von *PyTorch* für den Raspberry zu finden. Einige davon wurden

erfolglos ausprobiert, bis der Versuch aus Zeitgründen abgebrochen werden musste. In Zukunft muss das Problem behoben werden, um den Code auf ROS-E ausführen zu können oder der AVBERT muss auf einen Server ausgelagert werden, falls die Berechnungen zu komplex für den Raspberry Pi sind. [62] [63]

Die weiteren Lösungsversuche dieses Problems sind in Anhang D aufgeführt.

Nachdem sich herausstellte, dass kein Test des Gesamtkonzeptes auf dem Raspberry Pi möglich war, wurde der Code zurück auf einen Rechner übertragen und dort die fehlenden Teile hinzugefügt, um die Durchführung des Trainings zu testen. Im folgenden Abschnitt werden die Funktionen beschrieben, um die das Situation Awareness Module erweitert werden musste.

9.3.2 Das Situation Awareness Module

Neben der Verwendung des AVBERT und der Implementierung des *Double DQN*-Algorithmus hat der SAM-Knoten weitere Aufgaben, um die Situationserkennung vollständig verwendbar zu machen.

Eine der wichtigsten Aufgaben ist das Abgreifen der Bilder von der Kamera und den Audiodaten vom Mikrofon. Auf ROS-E gibt es für beide Bauteile einen ROS-Knoten, der diese Daten über ROS-Nachrichten zur Verfügung stellen soll. Da der Stream der Kamera über diese ROS-Nachrichten allerdings zu langsam war, wurde die Kamera mit Hilfe des Python-Packages OpenCV direkt angesprochen. Die Daten vom Mikrofon wurden auf ROS-E vom dafür vorgesehenen ROS-Knoten abgerufen. Auf dem Rechner wurde ein Python-Package namens pyaudio verwendet, um einen Audiostream vom Mikrofon zu erstellen. In den Klassen VideoClipLoader bzw. AudioClipLoader wurden jeweils eine bestimmte Anzahl an Frames bzw. Audiodaten in einem Array gespeichert. Dieses Array wurde nach dem First In First Out (FIFO) Prinzip kontinuierlich befüllt. Auf diese Weise werden jedes Mal die letzten 10 Sekunden für die Verarbeitung mit dem AVBERT verwendet. Bevor die Daten übergeben werden, müssen die Arrays zunächst in Tensoren umgewandelt werden. Dies geschieht in der ROSEnv-Klasse, die das eigene *Environment* beinhaltet. Diese Klasse ist ebenfalls dafür zuständig, die Belohnung in jedem Schritt von der App abzufragen. In der derzeitigen Umsetzung wurde die Belohnung ersatzweise über die Konsole eingelesen.

Nachdem eine Aktion ausgewählt wurde, muss diese von ROS-E ausgeführt werden. Die zeitintensive Erarbeitung der einzelnen Anbindungen an die verschiedenen Knoten, wie den Facial Expression Manager für die Gesichtsausdrücke und Animationen oder an RoSaSS für die Sprachausgabe konnte bisher nicht umgesetzt werden.

9.3.3 Die Trainings-App

Die Entwicklung von ROS-E umfasste bis zu diesem Zeitpunkt den Aufbau von grundlegenden ROS-Knoten für das Ansteuern der einzelnen Bauteile, wie Motoren, die Kamera und die Displays. Da das Entwicklerteam nicht sehr groß ist, wurden noch keine Apps entwickelt. Die Apps für ROS-E sollen in Form einer Webseite umgesetzt werden, die von einem Webserver auf ROS-E gehostet wird. Die Benutzer können sich über das WLAN mit ROS-E verbinden und diese Webseite auf einem beliebigen Gerät (z.B. einem Tablet) aufrufen. Sobald der Benutzer eine Aktion auf der Webseite ausführt (z.B. eine Belohnung vergeben) kann der Webserver auf diese Information zugreifen. Dieser kann jedoch nicht direkt auf die Funktionen von ROS-E zugreifen. Die Information mit dem Wert der Belohnung muss vom Webserver zum SAM-Knoten übermittelt werden, damit dieser das zweite Teilnetz trainieren und die nächste Aktion ausführen kann.

Eine solche Schnittstelle ist zwar in der Entwicklung, konnte aber im Rahmen der Bearbeitungszeit nicht mehr verwendet werden. Da die Trainings-App ohne diese Schnittstelle nicht getestet werden konnte, wurde die Priorität für die Umsetzung der Trainings-App sehr niedrig eingestuft. Die Trainings-App wurde aus diesen Gründen nicht umgesetzt, könnte aber anhand des Mockups und der Berücksichtigung während der Umsetzung der anderen Komponenten in Zukunft entwickelt werden.

Zusammenfassung des Kapitels

Die für das erste Teilnetz verwendete Architektur des AVBERT wurde konfiguriert, indem das Netz verkleinert wurde. Dadurch ist es möglich, das Netz direkt auf ROS-E zu verwenden. Die angepasste Architektur wurde mit dem selbst erstellten Datenset trainiert. Obwohl beim Training kein verwendbarer *Checkpoint* erreicht werden konnte, wurde der Code für das Netz so angepasst, dass er auf ROS-E übertragen werden konnte.

Anschließend wurde der verwendete Code für den *DQN*-Algorithmus angepasst. Dazu wurde eine eigene *Environment*-Klasse erstellt, die den AVBERT für die Auswertung der aktuellen Situation verwendet und die Belohnung vom Benutzer entgegennimmt.

Bei der Übertragung des Quellcodes für die beiden Teilnetze auf ROS-E trat jedoch ein Problem mit der Installation von *PyTorch* auf *ARM-Architekturen* auf, das bisher nicht gelöst werden konnte. Aus diesem Grund wurde eine Version des SAM-Knotens ohne die speziellen Teile für *ROS* für die Verwendung auf einem Rechner implementiert, um den Code testen zu können. Diese Implementierung konnte erfolgreich getestet werden.

10 Fazit

In diesem Kapitel soll ein Überblick über die im Rahmen dieser Arbeit gewonnenen Erkenntnisse und die umgesetzten und noch offenen Aufgaben für eine Situationserkennung auf humanoiden Robotern gegeben werden. Dazu werden zunächst die gefundenen Antworten auf die Forschungsfragen zusammengefasst. Anschließend wird versucht, das erstellte Gesamtkonzept für die Situationserkennung einzuschätzen. Zum Schluss werden mögliche Lösungen für offene Probleme und Aufgaben vorgestellt, die in Zukunft umgesetzt werden sollen, um die Situationserkennung im Einsatz auf ROS-E nutzen zu können.

10.1 Auswertung der Forschungsfragen

In diesem Abschnitt werden die Antworten auf die Forschungsfragen kurz zusammengetragen:

1. Welche Situationen treten im Umgang mit dem Tisch-Roboter ROS-E auf?

   Zum einen gibt es eine Reihe an Situationen, die sich auf die grundlegenden Funktionen des Roboters beziehen, wie zum Beispiel das Abbrechen einer gerade ausgeführten Funktion oder das Bestätigen oder Ablehnen eines Vorschlags des Roboters. Zum anderen sind Situationen vorstellbar, die von menschlicher Kommunikation übertragen werden, wie zum Beispiel Begrüßungen und Verabschiedungen. Selbst in einem eingeschränkten Szenario, wie der Vorstellung eines Roboters bei einer Veranstaltung können darüber hinaus jedoch zu viele andere Situationen eintreten, als dass sie vollständig erfasst oder von Entwicklern vorhergesehen und berücksichtigt werden könnten. Es ist daher nicht möglich, eine abschließende Antwort auf die Frage zu geben, da es unendlich viele Situationen gibt. Diese Erkenntnis führt direkt zur zweiten Forschungsfrage.

2. Kann mit Hilfe von Künstlichen Neuronalen Netzen eine Situationserkennung so entwickelt werden, dass die zu erkennenden Situationen nicht fest definiert, sondern gelernt werden können?

   Menschen können auf viele verschiedene Situationen eingehen, da sie ein sehr effektives Neuronales Netz besitzen: das Gehirn. Indem die Verarbeitung einer Situation im menschlichen Gehirn untersucht wurde, konnten Grundbausteine für eine Situationserkennung identifiziert werden. Dazu zählt die Wahl mehrerer Eingangskanäle, die Verknüpfung dieser Eingangsdaten durch *Fusion* und *Alignment* und schließlich die Formulierung einer übergeordneten Aufgabe für deren Lösung die Eingangsdaten anhand des *Attention-Mechanismus* priorisiert und aussortiert werden.

Diese übergeordnete Aufgabe kann so definiert werden, dass der Roboter in jeder dieser abstrakten Repräsentationen einer Situation eine Aktion zuordnet. Mit Hilfe des *Reinforcement Learning* ist es möglich, ein Künstliches Neuronales Netz zu trainieren, das jederzeit weiterlernen kann. Dabei kann entweder der Entwickler oder der Benutzer festlegen, welche Situationen der Roboter auswerten soll und welche keinen weiteren Einfluss haben sollen.

Im Rahmen dieser Arbeit wurde zunächst die Vorstellung des Roboters auf einer Veranstaltung als Einsatzgebiet betrachtet. Mit der entwickelten Trainings- und Testumgebung ist es jedoch möglich, ROS-E auch auf einem Schreibtisch oder in einem Kindergarten zu trainieren. Dabei würde sie lernen, die neuen Eingangsdaten ihren vorhandenen Aktionen zuzuordnen, ohne dass ein Mensch jede Kombination dieser Eingangsdaten als eine bestimmte Situation benennen muss.

3. Welche Eingangskanäle können in einem Künstlichen Neuronalen Netz für die Situationserkennung kombiniert werden?

   Die verwendeten Eingangskanäle hängen zunächst von den im humanoiden Roboter verwendeten Sensoren ab. Neben einem Mikrofon und einer Kamera könnten auch Buttons, Touch-Sensoren, GPS-, CO_2- und Temperatursensoren oder viele weitere Sensoren verwendet werden. Dabei ist allerdings zu beachten, dass nicht alle humanoiden Roboter diese Sensoren besitzen und die Situationserkennung mit mehr Sensoren vielleiht nicht auf den meisten Robotern einsetzbar ist. Ein weiterer Aspekt, der die Auswahl der Eingangskanäle einschränkt, ist das Vorhandensein von Trainingsdaten. Je mehr Eingangskanäle verwendet werden, desto unwahrscheinlicher ist es, dass bereits große Datensets mit genau dieser Kombination verfügbar sind. Daher müssten in vielen Fällen alle Trainingsdaten selbst gesammelt werden.

   Ein weiterer wichtiger Aspekt für die Wahl der Eingangskanäle ist der Informationsgehalt der Daten. Bei der Verarbeitung mehrerer Eingangskanäle können drei Fälle auftreten: die Eingangskanäle liefern völlig unabhängige Daten, die Daten widersprechen sich oder sie sind redundant. Wie im menschlichen Gehirn muss ein Mittelweg gefunden werden zwischen der Effektivität von völlig unabhängigen Daten und der erhöhten Sicherheit der Ergebnisse bei redundanten Daten. Um die widersprüchlichen Daten auswerten zu können, müssen geeignete Mechanismen, wie der *Attention-Mechanismus* vorhanden sein, um zu lernen, welche Daten in welchen Fällen öfter korrekt sind.

4. Auf welche Weise können Situationen in einem Künstlichen Neuronalen Netz abgebildet werden?

   Ein Künstliches Neuronales Netz nimmt die Eingangsdaten, die zum Beispiel von einer Kamera oder einem Mikrofon digitalisiert wurden, in Form von Vektoren entgegen. Die Vektoren mehrerer Eingangskanäle werden durch *Fusion* und *Alignment* zu einem gemeinsamen Vektor verknüpft. So kann beispielsweise ein Geräusch zu einem bestimmten Objekt zugeordnet werden. Indem eine übergeordnete Aufgabe definiert wird, können die Eingangsdaten bei der Verknüpfung mit Hilfe des *Attention-Mechanismus* priorisiert werden. Wenn der Roboter das Licht steuern kann, könnte er besonders auf die Helligkeit aber auch auf einen schlafenden Menschen oder einen leeren Raum achten. Eine Situation kann in einem Künstlichen Neuronalen Netz als Vektor abgebildet werden, der die wichtigsten Informationen zur Lösung einer bestimmten Aufgabe zusammenfasst und komprimiert.

5. Wie können die Ergebnisse der Situationserkennung für die Nutzung in anderen Anwendungen bereitgestellt werden?

   Die flexibelste Art, die Ergebnisse bereitzustellen, ist der *Representation-Vector*, der die wichtigsten Informationen der Situation zusammenfasst. Da dieser Vektor im Vergleich zu einem Video- oder Audio-Clip sehr klein ist, kann er direkt auf ROS-E effektiv für Berechnungen verwendet werden. Es ist jedoch notwendig, ein weiteres kleines Künstliches Neuronales Netz oder einen Algorithmus zu verwenden, der diesen Vektor interpretiert oder lernt, einer selbst definierten Anzahl an Situationen zuzuordnen. Eine weitere Möglichkeit besteht darin, die Ergebnisse des Netzes zu verwenden, das für die Klassifizierung während des Trainings verwendet wurde. Dessen Klassen sind jedoch festgelegt und können ohne ein weiteres Training und das Erstellen neuer *Label* nicht verändert werden.

10.2 Auswertung des Gesamtkonzeptes

In diesem Abschnitt soll diskutiert werden, ob der in dieser Arbeit vorgestellte Ansatz für die Umsetzung einer Situationserkennung für humanoide Roboter geeignet ist.

Die für die Umsetzung ausgewählte Architektur des AVBERT erfüllt alle Aufgaben, die als Grundbausteine für eine Situationserkennung erarbeitet wurden. Es wird mehr als ein Eingangskanal verwendet, wobei die Architektur so aufgebaut ist, dass noch zusätzliche Eingangskanäle hinzugefügt werden könnten. Diese Eingangskanäle werden mit dem *Attention-Mechanismus* zusammengeführt, wobei das Problem der *Fusion* und des *Alignment* gelöst werden. Die Architektur konnte auf eine

Größe reduziert werden, mit der sie theoretisch auch auf dem begrenzten Arbeitsspeicher von ROS-E verwendet werden kann. Aus diesen Gründen wird die Wahl der Architektur als passend für eine Situationserkennung auf humanoiden Robotern eingeschätzt.

Der angepasste Quellcode für den AVBERT konnte auf ROS-E jedoch nicht erfolgreich ausgeführt werden. Da zu diesem Zeitpunkt nicht festgestellt werden konnte, was den Abbruch der Ausführung verursacht, kann nicht ausgeschlossen werden, dass vom AVBERT eine Operation ausgeführt wird, die auf der CPU von ROS-E mit der *ARM-Architektur* oder den begrenzten Ressourcen nicht unterstützt wird. In diesem Fall müsste das Netz auf einen externen Server ausgelagert werden und wäre nicht mehr offline verfügbar. Sollte in Zukunft ein anderer Rechner oder sogar eine GPU bei ROS-E eingebaut werden, könnte das Netz darauf erneut getestet werden.

Der Versuch, den AVBERT ausschließlich mit einer Klassifizierungsaufgabe zu trainieren, war nicht erfolgreich. Daher konnte nicht ermittelt werden, wie sehr die Genauigkeit durch die Verkleinerung des Netzes eingeschränkt wurde. In Zukunft müssen die beiden Trainingsaufgaben, die im Paper beschrieben wurden selbst implementiert werden, um das Training wiederholen und einen guten *Checkpoint* erhalten zu können.

Der Reinforcement-Ansatz für das zweite Teilnetz ist theoretisch sehr gut geeignet, den *Representation-Vector* zu verwenden, um Aktionen zu lernen. Auf diese Weise kann den Benutzern nicht nur ermöglicht werden, die „richtigen“ Aktionen selbst zu bestimmen, sondern das Training jederzeit weitergeführt werden. So kann ROS-E auch auf veränderte Vorlieben oder Abläufe eingehen. Eine sehr interessante Eigenschaft des *DQN*-Algorithmus stellt außerdem die Auswahl-Funktion dar, die mit einer bestimmten Wahrscheinlichkeit nicht die am besten bewertete Aktion, sondern eine zufällige Aktion auswählt. Auf diese Weise kann ein Roboter eine Art Neugier besitzen, die auf Benutzer menschlich wirken könnte.

Ob das Training insgesamt zu einem sinnvollen Ergebnis führt, konnte im Rahmen dieser Arbeit nicht getestet werden. Es war möglich, den grundlegenden Ablauf des Trainings beider Teilnetze zu durchlaufen, da jedoch kein *Checkpoint* für den AVBERT erstellt werden konnte, kann das Training des zweiten Teilnetzes nicht bewertet werden. Auf einem Rechner mit einer GPU und einer funktionierenden Version von *PyTorch* konnte das Training des zweiten Teilnetzes getestet werden. Dazu wurden eine Kamera und ein Mikrofon angeschlossen und die Bewertung zunächst über die Konsole, statt über die Trainings-App entgegengenommen. Da das Training aber mehrere hundert *Episoden* benötigen würde und die Vektoren vom AVBERT die Situation nicht gut zusammenfassen, wurde kein vollständiger Trainingsdurchlauf ausgeführt. Auf diese Weise konnte aber gezeigt werden, dass die Situationserkennung nicht nur auf ROS-E, sondern auch auf anderen humanoiden Robotern oder

auch anderen Geräten funktionieren würde, vorausgesetzt diese besitzen eine funktionierende Installation von *PyTorch* und den übrigen benötigen Python-Packages.

An dieser Stelle soll festgehalten werden, dass das Training - auch ohne die ausgeführte Reaktion von ROS-E - nur über den Text auf der Konsole eine interessante Aufgabe war, die den Benutzern mit einer App und den Reaktionen von ROS-E später viel Spaß machen könnte.

10.3 Ausblick

In diesem Abschnitt sollen die Aufgaben festgehalten werden, die für eine erfolgreiche Umsetzung des Gesamtkonzeptes der Situationserkennung ausgeführt werden müssten und wie man dieses Konzept noch erweitern könnte.

Zunächst müsste ein ausreichend guter *Checkpoint* für den AVBERT erstellt werden, indem die beiden Trainingsaufgaben aus dem Paper für den AVBERT selbst implementiert werden und das Training wiederholt wird. Anschließend müsste das Problem mit PyTroch auf der *ARM-Architektur* gelöst oder der AVBERT auf einen Server ausgelagert werden. Sobald das Netz einen sinnvollen Vektor liefert, kann der Reinforcement-Ansatz getestet werden. Sollte dieser gute Ergebnisse erzielen, könnte im nächsten Schritt die Belohnung nicht mehr über die Konsole, sondern über die App eingegeben werden. Dazu muss allerdings die Schnittstelle zwischen ROS-Knoten und Webservern fertiggestellt sein.

Die einzelnen *Experiences* während des Trainings werden zum Lernen in einem Cache zwischengespeichert. Wenn diese zusätzlich an einen Datenserver gesendet werden würden, könnten Erfahrungen von verschiedenen Einsatzorten in die Erstellung eines neuen *Checkpoints* einbezogen werden. Die *Experiences* könnten in Form einzelner Trainingssessions gezielt zusammengestellt werden, sodass auch verschiedene *Checkpoints* für verschiedene Einsatzgebiete erstellt werden können. Dazu wäre es nötig, das gesamte Konzept für die Trainings- und Testumgebung umzusetzen, wie es in Kapitel 8 beschrieben wurde.

In der Trainings-App könnten zusätzlich auch die Clips angezeigt werden, die zur Erstellung des Vektors verwendet wurden. So hätten die Benutzer ein besseres Bild, welche Zusammenhänge das Netz lernt. Außerdem sollte eine Trainingssession benannt werden können bevor die Daten zum Datenserver gesendet werden, sodass die Session später gezielt für die Erstellung eines *Checkpoints* verwendet werden kann.

Sollte das Gesamtkonzept später auf ROS-E vollständig umgesetzt und funktionsfähig sein, ist eine wichtige Überlegung, ob mehr Eingangskanäle verwendet werden könnten. Da es viele verschiedene

Kombinationen von Sensoren und Bauteilen in humanoiden Robotern gibt, ist es schwer, ein Datenset zu finden, das genau die Daten enthält, die ROS-E verarbeiten könnte. Es wird daher notwendig sein, die Daten selbst zu sammeln. Anschließend kann untersucht werden, welche Daten die meisten Informationen für die Situationserkennung enthalten und somit das Ergebnis am meisten verbessern.

Im besten Fall kann ROS-E mit der Situationserkennung irgendwann für viele Situationen eine passende Reaktion auswählen, mit der sie ihre verschiedenen Benutzer an den unterschiedlichen Einsatzorten bei ihren täglichen Aufgaben unterstützen kann.

ROS-E könnte für Andrea, die gerade laut Musik hört, als sie einen Anruf erhält, die Musik ausschalten und den Anruf annehmen.

Sie könnte fragen, ob sie einen Notfallkontakt anrufen soll, als Elfriede gestürzt ist.

Und sie könnte passende Musik oder Geräusche abspielen, je nachdem welche Geschichte Thore mit seinen Action-Figuren gerade spielt.

Abkürzungsverzeichnis

API ..... *Application Programming Interface, Schnittstelle*

CNN..... *Convolutional Neural Network*

CPU ..... *Central Processing Unit, Zentrale Verarbeitungseinheit*

CUDA ..... *Compute Unified Device Architecture*

DQN ..... *Deep Q-Network*

FFT ..... *Fast Fourier Transform*

FPAN ..... *Frontoparietal Attention Network, (laterales) frontoparietales Netzwerk*

FPS ..... *Frames Per Second, Bilder pro Sekunde*

GB ..... *Gigabyte*

KNN..... *Künstliches Neuronales Netz*

MLP..... *Multi-Layer Perceptron*

NLP..... *Natural Language Processing*

RAM ..... *Random-Access Memory, Arbeitsspeicher*

ResNet ..... *Residual Neural Network*

ROS ..... *Robot Operating System*

Glossar

Accuracy	Eine Größe für die Bewertung von Trainingsergebnissen von KNNs. Sie gibt an, wie viele Vorhersagen in Bezug auf die Gesamtanzahl der Vorhersagen korrekt waren. [13]
Agent	Als Agent wird beim Reinforcement Learning das Programm bezeichnet, das Informationen aus der Umwelt aufnimmt und anhand einer gelernten Verhaltensstrategie eine Aktion passend zum aktuellen Zustand der Umgebung auswählt. [13]
Alignment	Im Deutschen etwa: Ausrichtung bezeichnet den Vorgang, Daten aus verschiedenen Eingangskanälen meist zeitlich oder verschiedene Datenraten aneinander auszurichten. [25]
ARM-Architektur	Eine bestimmte Architektur für Prozessoren, die seit den 1980er Jahren entwickelt wird. Zunächst stand die Abkürzung für Acorn Risc Architecture, wird aber derzeit von der Firma Advanced Risc Machines Ltd. weiterentwickelt, wofür die Abkürzung ebenfalls gebraucht wird. [64]
Artificial Intelligence	kurz: AI, siehe Künstliche Intelligenz.
Artificial Neural Network	kurz: ANN, siehe Künstliches Neuronales Netz.
Attention-Heads	Ein bestimmter Teil der Transformer-Architektur für KNNs, die den Attention-Mechanismus parallelisiert, sodass gleichzeitig unterschiedliche Priorisierungskarten (sogenannte Attention-Maps) erstellt werden können, die unterschiedliche Zusammenhänge in den Daten priorisieren. [31]
Attention-Layer	Eine Bezeichnung für die Umsetzung des Attention-Mechanismus in einem KNN. Die beteiligten Schichten, Knoten und Operationen, die für den Mechanismus verwendet werden, können zur Vereinfachung der Beschreibung zu einer Schicht zusammengefasst werden. [31] [40]
Attention-Mechanismus	Ein Mechanismus, mit dem KNNs nicht nur einzelne Inputs gewichten, sondern Prioritäten über den gesamten Input erstellen können. [33] [31]

Augmentation	Eine Methode des Machine Learning bei der neue Trainingsdaten aus bereits vorhandenen Daten generiert werden, indem diese leicht verändert (z.B. gespiegelt oder rotiert) werden. [55]
Autoencoder	Eine bestimmte Architektur für KNNs, die aus einem sogenannten Encoder besteht, der die hochdimensionalen Eingangsdaten auf einen kleineren Vektor reduziert und einem Decoder, der die ursprünglichen Daten aus dem komprimierten Vektor wiederherstellt und so dafür sorgt, dass möglichst wenig Informationen bei der Reduzierung verloren gehen. [29]
Checkpoint	Die Bezeichnung für einen Speicherpunkt aller Gewichte eines KNNs während oder zum Abschluss eines Trainingsdurchlaufes. Mit diesem Speicherpunkt kann das Wissen des Netzes gespeichert und wieder geladen werden. [65]
Classification-Head	Ein Teilnetz, das für das Training des AVBERT mit einer Klassifizierungsaufgabe an den AVBERT angehängt wird und den Representation-Vector weiterverarbeitet. [40]
Computer Vision	Ein Teilgebiet der Forschung zu Künstlicher Intelligenz, das sich mit der Verarbeitung von Bildern, Videos und allgemein visuellen Informationen befasst. [66]
Correct Pair Prediction	Eine Trainingsaufgabe für KNNs, bei der das Netz eine Vorhersage treffen soll, ob zwei gezeigte Inputs zusammengehören oder nicht. Im Fall des AVBERT wurde das Training verwendet, um zu lernen, ob der gezeigte Video-Clip zum gezeigten Audio-Clip passt. [40]
Cross-Entropy Loss	Eine Art der Berechnung für den Fehler, den ein KNN bei einer Vorhersage gemacht hat. Der Cross-Entropy Loss ist besonders für Klassifizierungsaufgaben geeignet und hat die Eigenschaft, falsche Vorhersagen, die das Netz jedoch mit einer hohen Wahrscheinlichkeit belegt hat, stärker zu bewerten. [58] [13]
CUDA	Eine Abkürzung für Compute Unified Device Architecture ist eine von der Firma NVIDIA entwickelte Schnittstelle, um Programme auf GPUs (Grafikkarten) ausführen zu können. [61]

Deep Neural Networks	kurz: DNN. Eine Bezeichnung für Künstliche Neuronale Netze, die aus sehr vielen Schichten bestehen, wobei die Anzahl der Schichten, mit denen ein KNN als „deep“ gilt, nicht fest definiert ist. Die Bezeichnung wird verwendet, wenn ein komplexes Netz gemeint ist. [13]
Deep Q-Network	kurz: DQN. Eine Erweiterung des sogenannten Q-Learning, einem Algorithmus des Reinforcement Learning, bei dem ein DNN verwendet wird, um die verwendeten Funktionen des Algorithmus anzunähern. [13] [47]
Double DQN	kurz für Double Deep Q-Network ist eine Erweiterung des DQN-Algorithmus, bei dem die Berechnung der Q-Funktion und die Auswahl der Aktion auf zwei Netze verteilt werden, die in bestimmten Abständen synchronisiert werden. [47]
Environment	Ein Begriff aus dem Reinforcement Learning, der die Umgebung bezeichnet, in dem der Agent sich bewegt. Diese Umgebung kann die reale Welt oder eine simulierte Umgebung, wie ein Spiel sein. [13]
Episode	Ein Begriff aus dem Reinforcement Learning, der mehrere Durchläufe, bestehend aus der Aufnahme einer Beobachtung aus der Umgebung, über die Auswahl einer passenden Aktion bis zur Aufnahme der zugehörigen Belohnung und der nächsten Beobachtung, umfasst. Am Ende einer Episode ist ein höheres Ziel erreicht, wie das Ende einer Runde eines Spiels oder das erfolgreiche Anheben eines Gegenstandes. [13]
Epoche	Die Bezeichnung für das Verarbeiten des gesamten Datensets während des Trainings eines KNNs. Eine Epoche besteht aus einer Trainings- und einer Testphase. Ein Training kann mehrere Epochen umfassen, also mehrmals die gesamten Daten des Datensets verarbeiten.
Experience Replay	Eine Methode des Reinforcement Learning, bei der die einzelnen Experiences gespeichert werden, um sie zu einem späteren Zeitpunkt (erneut) für das Training des KNNs oder des Algorithmus zu verwenden. [13]
Experience	Beinhaltet den aktuellen State (Zustand) oder die aktuelle Observation (Beobachtung) des Environments (Umgebung), die daraufhin gewählte Aktion, die zugehörige Belohnung und den daraufhin eingetretenen nächsten State bzw. Observation. Damit ist eine Experience die kleinste Einheit der Trainingsdaten beim Reinforcement Learning. [13]

Finetuning	Das Training eines KNNs kann in zwei Phasen eingeteilt werden: das Pre-Training und das Finetuning. Im Finetuning wird das Netz auf die Aufgabe abgestimmt, die es im Einsatz lösen soll. Dazu werden Daten verwendet, die den im Einsatz verwendeten Daten möglichst ähnlich sind. [13]
FPAN	Das Frontoparietal Attention Network ist ein Teil des menschlichen Gehirns, das für die Priorisierung von Sinneseindrücken zuständig ist, um die Aufmerksamkeit des Menschen zu lenken. [30]
Fusion	Im Deutschen etwa: Verknüpfung bezeichnet den Vorgang, mehrere Modalitäten (Eingangskanäle) in einem KNN zu einer internen Darstellung zusammenzubringen. [26]
GitHub	Eine Plattform, auf der Quellcode mit Hilfe von Git, einer Software zur Versionskontrolle, verwaltet und veröffentlicht werden kann. [67]
Hidden-Layer	Ein KNN besteht aus drei Arten von Schichten (Layers): die Input-Layer nehmen die Eingabedaten entgegen, die Hidden-Layer verarbeiten die Daten, ohne dass die Zwischenergebnisse bekannt sein müssen (hidden, versteckt) und der Output-Layer gibt das Ergebnis aus. [13]
Interozeption	Im Englischen Interoception bezeichnet die Sinneseindrücke, die innerhalb des menschlichen Körpers bei den inneren Organen und Geweben aufgenommen werden. Beispiele dafür sind die Wahrnehmung des Blutdrucks, des Blutzuckerspiegels oder des Dehnungsgrades des Magens. [4]
k-Nächste Nachbarn Algorithmus	Ein Algorithmus zur Einteilung von Daten in eine Gruppe (Cluster), wobei die Cluster gelernt, also anhand der zuvor gegebenen Beispiele gebildet werden. [15]
k-Nearest Neighbor Algorithm	siehe k-Nächste Nachbarn Algorithmus.
Künstliche Intelligenz	kurz: KI. Ein Oberbegriff für Algorithmen und Methoden, die menschliche Intelligenz nachahmen, wobei Intelligenz nicht fest definiert ist und sich auf unterschiedliche Aspekte wie Lernfähigkeit, Problemlösung oder Verhalten beziehen kann. [14]

Künstliches Neuronales Netz	kurz: KNN. Eine Bezeichnung für eine netzartige Struktur aus einzelnen Knoten und Verbindungen, die im Machine Learning dazu verwendet werden, um Funktionen durch das Training anhand vieler Beispieldaten anzunähern. [13]
Label	Bezeichnet den gewünschten („korrekten“) Wert für einen Output eines KNNs beim Supervised Learning. Dieses Label muss in den meisten Fällen von einem Menschen festgelegt werden. [13]
Loss	Eine Größe für das Messen der Höhe des Fehlers, den ein KNN bei einer Vorhersage in Bezug auf den gewünschten („richtigen“) Wert macht. [13]
Machine Learning	Im Deutschen Maschinelles Lernen ist ein Oberbegriff für Algorithmen und andere Methoden, bei denen mit einem Rechner aus Beispieldaten bestimmte Zusammenhänge gelernt werden, ohne dass ein Mensch diese im Einzelnen vorgeben muss. [17]
Maschinelles Lernen	siehe Machine Learning.
Masked Embedding Prediction	Eine Trainingsaufgabe für KNNs, bei der ein Teil der Daten maskiert wird und das Netz diese fehlenden Daten vorhersagen soll. Im Fall des AVBERT wurde das Training verwendet, um Zusammenhänge innerhalb einer Modalität zu lernen. [40]
Multi-Layer Perceptron	kurz MLP ist die einfachste Form eines KNNs bestehend aus mehreren hintereinander geordneten Knoten (Perceptrons). [13]
Natural Language Processing	kurz NLP ist ein Teilgebiet der Künstlichen Intelligenz, das sich mit der Verarbeitung von Texten mit Rechnern beschäftigt, wobei das Ziel ist, dass Rechner die Texte oder auch gesprochene Sprache ähnlich wie ein Mensch verstehen kann. [32]
NVIDIA	Ein Unternehmen, das Grafikkarten für Gaming aber auch spezialisierte GPUs für Machine Learning mit Servern entwickelt. [68]
Observation	Bezeichnet die Daten, die ein Agent beim Reinforcement Learning aufnimmt. Im Gegensatz zum State wird dabei nicht der gesamte Zustand der Umgebung aufgenommen, sondern nur der für den Agent sichtbare Bereich. [13]

OpenAI	Ein Unternehmen, das Forschung in vielen Bereichen der Künstlichen Intelligenz unterstützt. [59]
Overfitting	Tritt beim Training von KNNs auf, wenn dieses im Training eine hohe Genauigkeit erreicht, jedoch im Test auf nie zuvor gesehenen Daten sehr schlechte Ergebnisse erzielt. Das Netz wurde in diesem Fall zu stark an die Trainingsdaten angepasst, kann das Wissen aber nicht auf neue Daten generalisieren. [54] [13]
Perceptron	Das von Frank Rosenblatt entwickelte Modell einer menschlichen Gehirnzelle für einen Rechner, das den Grundbaustein (Knoten) für KNNs darstellt. [13]
pip	Ein Tool, um Python-Pakete auf verschiedenen Betriebssystemen zu installieren. [69]
Policy	Bezeichnet die Verhaltensstrategie, die der Agent beim Reinforcement Learning lernt, um eine passende Aktion auszuwählen. [13]
Pre-Training	Das Training eines KNNs kann in zwei Phasen eingeteilt werden: das Pre-Training und das Finetuning. Beim Pre-Training werden grundlegende Zusammenhänge, wie Strukturen von Sätzen oder Muster und Objekte, der Datentypen (Texte, Bilder, …) gelernt, ohne dass dabei die spätere Aufgabe verwendet werden muss. [13]
PyTorch	Ein Framework für Machine Learning basierend auf der Programmiersprache Python. [70]
Q-Learning	Ein häufig verwendeter Algorithmus für Reinforcement Learning. [13]
Recurrent Neural Network	kurz: RNN bezeichnet eine Architektur für KNNs, die für die Verarbeitung von Sequenzen, wie aufeinanderfolgende Wörter, verwendet werden. Dabei gibt es besondere Verbindungen zwischen den Knoten, die anders als in einem MLP auch auf vorhergehende Schichten zeigen können. [13]
Reinforcement Learning	Bezeichnet einen Teil des Machine Leaarning bei dem die Aufgabe des Lernens anhand einer Verhaltensstrategie und eines Belohnungssystems anstatt mit dem Festlegen von „korrekten“ Ergebnissen mit Labels gelöst wird. [13]

Representation-Vector	Ein Vektor, der durch den AVBERT erstellt wird, indem die wichtigsten Informationen aus den Video- und Audio-Clips mit Hilfe des Attention-Mechanismus zusammengefasst werden.
ResNet	Residual Neural Networks stellen eine bestimmte Architektur für KNNs dar, bei denen es Verbindungen gibt, die mehrere Schichten überspringen können. [56]
ROS	kurz für Robot Operation System. Eine Sammlung von Bibliotheken und Tools für die Programmierung von Robotern. [71]
Spektrogramm	Eine Darstellung von Audiodaten in der Amplitude und Frequenz über die Zeit aufgetragen werden. [38]
Split	Eine Teilmenge eines Datensets, das einen bestimmten Zweck erfüllt (Training, Test oder Validation) oder bestimmte Zusammenhänge der Daten abbildet. [13]
State	Bezeichnet im Reinforcement Learning den Zustand der Environments (Umgebung) zum Beispiel in Form der aktuellen Position des Spielfeldes und der Spielsteine in einem Spiel. [13]
Supervised Learning	Im Deutschen Beaufsichtigtes Lernen ist eine Trainingsmethode für KNNs, bei der die Vorhersage des Netzes mit einem von einem Menschen festgelegten korrekten Ergebnis (Label) verglichen wird, um zu lernen. [44]
TensorBoard	Ein Tool zum Aufzeichnen und Visualisieren von Metriken während des Trainings eines KNNs. [57]
TensorFlow	Ein Framework für Machine Learning basierend auf der Programmiersprache Python. [72]
Transformer	Eine Architektur für KNNs, die den Attention-Mechanismus umsetzt. [31]
Unsupervised Learning	Im Deutschen Unbeaufsichtigtes Lernen ist eine Trainingsmethode für KNNs, bei der das korrekte Ergebnis nicht für jedes Trainingsbeispiel von einem Menschen vorgegeben werden muss. [44]
Visual Question Answering	kurz VQA ist eine Aufgabenstellung für KNNs bei der das Netz Fragen zu einem Bild beantworten soll. [24]

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur- und Quellenverzeichnis

[1] T. Lüthe, „Konzeption und prototypische Implementierung eines hybriden Algorithmus zur Emotionserkennung mit humanoiden Robotern," 2020.

[2] Microsoft, „Erkennung von erkannten Emotionen mithilfe der Gesichtserkennungs-API," 2018. [Online]. Available: https://docs.microsoft.com/de-de/xamarin/xamarin-forms/data-cloud/azure-cognitive-services/emotion-recognition . [Zugriff am 15 Juli 2021].

[3] IBM, „Endpunkt für allgemeine Zwecke verwenden," 2019. [Online]. Available: https://cloud.ibm.com/docs/tone-analyzer?topic=tone-analyzer-utgpe. [Zugriff am 15 Juli 2021].

[4] L. Feldman Barrett, How Emotions Are Made, New York: Houghton Mifflin Harcourt Publishing Company, 2017.

[5] L. Feldman Barrett, Seven And A Half Lessons About The Brain, New York: Houghton Mifflin Harcourt Publishing Company, 2020.

[6] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel und Y. Bengio, „Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," *CoRR,* Bd. abs/1502.03044, 2016.

[7] O. Rudovic, J. Lee, M. Dai, B. Schuller und R. Picard, „Personalized machine learning for robot perception of affect and engagement in autism therapy," *Science Robotics,* Bd. 3, 2018.

[8] M. Feichter, „Demenz," 2018. [Online]. Available: https://www.netdoktor.de/krankheiten/demenz/. [Zugriff am 19 Juli 2021].

[9] J. Dobmeier und C. Fux, „Depression," 2018. [Online]. Available: https://www.netdoktor.de/krankheiten/depression/. [Zugriff am 19 Juli 2021].

[10] Dudenredaktion, „Situation," 2021. [Online]. Available: https://www.duden.de/node/166752/revision/166788. [Zugriff am 16 Juli 2021].

[11] C. Gu, C. Sun, D. Ross, C. Vondrick, C. Pantofaru, Y. Li, S. Vijayanarasimhan, G. Toderici, S. Ricco, R. Sukthankar, C. Schmid und J. Malik, „AVA: A Video Dataset of Spatio-temporally Localized Atomic Visual Actions," *CoRR,* Bd. abs/1705.08421, 2018.

[12] A. Turing, „COMPUTING MACHINERY AND INTELLIGENCE," *Mind,* Bd. 236, 1950.

[13] H. Dong, Z. Ding und S. Zhang, Deep Reinforcement Learning: Fundamentals, Research and Applications, Singapore: Springer Singapore, 2020.

[14] IBM, „Artificial Intelligence (AI)," 2021. [Online]. Available: https://www.ibm.com/cloud/learn/what-is-artificial-intelligence. [Zugriff am 3 August 2021].

[15] M. von der Hude, Predictive Analytics und Data Mining, Wiesbaden: Springer Vieweg, 2020.

[16] W. Stangl, „Lernen,“ 2021. [Online]. Available: https://lexikon.stangl.eu/551/lernen. [Zugriff am 10 August 2021].

[17] IBM Cloud Education, „Maschinelles Lernen,“ 2020. [Online]. Available: https://www.ibm.com/de-de/cloud/learn/machine-learning. [Zugriff am 3 August 2021].

[18] A. Wichert, „Künstliche Intelligenz,“ 2021. [Online]. Available: https://www.spektrum.de/lexikon/neurowissenschaft/kuenstliche-intelligenz/6810. [Zugriff am 10 August 2021].

[19] W. Müller, Gibt es einen "7. Sinn"?, Berlin: Springer, 2016.

[20] N. Bruno und F. Pavani, Perception: A multisensory perspective, Oxford: Oxford University Press, 2018.

[21] Facebook AI, „Datasets,“ 2021. [Online]. Available: https://paperswithcode.com/datasets. [Zugriff am 20 August 2021].

[22] L.-P. Morency, Autor, *Lecture 1.1: Introduction.* [Performance]. Carnegie Mellon University, 2020.

[23] B. Stein, T. Stanford und B. Rowland, „Multisensory Integration and the Society for Neuroscience: Then and Now,“ *Journal of Neuroscience,* Bd. 40, 2020.

[24] R. Hu und A. Singh, „Transformer is All You Need: Multimodal Multitask Learning with a Unified Transformer,“ *CoRR,* Bd. abs/2102.10772, 2017.

[25] L.-P. Morency, Autor, *Lecture 5.1: Multimodal Alignment.* [Performance]. Carnegie Mellon University, 2020.

[26] L.-P. Morency, Autor, *Lecture 8.1: Discriminative Graphical Models.* [Performance]. Carnegie Mellon University, 2020.

[27] S. Wang, J. Zhang und C. Zong, „Associative Multichannel Autoencoder for Multimodal Word Representation,“ in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brüssel, Association for Computational Linguistics, 2018.

[28] Z. Li, Z. Li, J. Zhang, Y. Feng und J. Zhou, „Bridging Text and Video: A Universal Multimodal Transformer for Audio-Visual Scene-Aware Dialog,“ *IEEE/ACM Transactions on Audio, Speech, and Language Processing,* Bd. 29, 2021.

[29] N. Jaques, S. Taylor, A. Sano und R. Picard, „Multimodal autoencoder: A deep learning approach to filling in missing sensor data and enabling better mood prediction,“ *2017 Seventh International Conference on Affective Computing and Intelligent Interaction (ACII),* Bd. 1, 2017.

[30] R. Ptak, „The Frontoparietal Attention Network of the Human Brain: Action, Saliency, and a Priority Map of the Environment,“ *The Neuroscientist,* Bd. 18, 2012.

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser und I. Polosukhin, „Attention Is All You Need,“ *CoRR,* Bd. abs/1706.03762, 2017.

[32] IBM Cloud Education, „Natural Language Processing (NLP),“ 2020. [Online]. Available: https://www.ibm.com/cloud/learn/natural-language-processing. [Zugriff am 3 August 2021].

[33] A. Kosiorek, „Attention in Neural Networks and How to Use It,“ 2017. [Online]. Available: http://akosiorek.github.io/ml/2017/10/14/visual-attention.html. [Zugriff am 6 August 2021].

[34] N. Adaloglou, „How Attention works in Deep Learning: understanding the attention mechanism in sequence models,“ 2020. [Online]. Available: https://theaisummer.com/attention/. [Zugriff am 6 August 2021].

[35] S. Mohla, „Cross-Attention is what you need!,“ 2020. [Online]. Available: https://towardsdatascience.com/cross-attention-is-what-you-need-fusatnet-fusion-network-b8e6f673491 . [Zugriff am 5 Juli 2021].

[36] O. Wiles, S. Ehrhardt und A. Zisserman, „Co-Attention for Conditioned Image Matching,“ *CoRR,* Bd. abs/2007.08480, 2020.

[37] D. Silver, S. Singh, D. Precup und R. Sutton, „Reward is enough,“ *Artificial Intelligence,* Bd. 299, 2021.

[38] K. Chaudhary, „Understanding Audio data, Fourier Transform, FFT and Spectrogram features for a Speech Recognition System,“ 2021. [Online]. Available: https://towardsdatascience.com/understanding-audio-data-fourier-transform-fft-spectrogram-and-speech-recognition-a4072d228520 . [Zugriff am 16 September 2021].

[39] S. Lee, „avbert,“ 2021. [Online]. Available: https://github.com/sangho-vision/avbert. [Zugriff am 5 Jannuar 2022].

[40] S. Lee, Y. Yu , G. Kim, T. Breuel, J. Kautz und Y. Song, „Parameter Efficient Multimodal Transformers for Video Representation Learning,“ *CoRR,* Bd. abs/2012.04124, 2020.

[41] OpenAI, „Part 1: Key Concepts in RL,“ 2021. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html. [Zugriff am 23 Oktober 2021].

[42] Y. Feng, S. Subramanian, H. Wang und S. Guo, „Train a Mario-playing RL Agent,“ 2021. [Online]. Available: https://pytorch.org/tutorials/intermediate/mario_rl_tutorial.html. [Zugriff am 23 Oktober 2021].

[43] M. Volodymyr, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg und D. Hassabis, „Human-level control through deep reinforcement learning,“ *Nature,* Bd. 518, 2015.

[44] J. Delua, „Supervised vs. Unsupervised Learning: What’s the Difference?,“ 2021. [Online]. Available: https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning. [Zugriff am 8 Juni 2021].

[45] J. Devlin, M.-W. Chang, K. Lee und K. Toutanova, „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,“ *CoRR,* Bd. abs/1810.04805, 2018.

[46] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Gheshlaghi Azar und D. Silver, „Rainbow: Combining Improvements in Deep Reinforcement Learning," *CoRR,* Bd. abs/1710.02298, 2017.

[47] A. Guez, H. van Hasselt und D. Silver, „Deep Reinforcement Learning with Double Q-learning," *CoRR,* Bd. abs/1509.06461, 2015.

[48] Facebook AI, „Datasets for Action Recognition," 2021. [Online]. Available: https://paperswithcode.com/datasets?task=action-recognition-in-videos. [Zugriff am 2 September 2021].

[49] Facebook AI, „UCF101 (UCF101 Human Actions dataset)," 2021. [Online]. Available: https://paperswithcode.com/dataset/ucf101. [Zugriff am 2 September 2021].

[50] Google DeepMind, „Kinetics," 2021. [Online]. Available: https://deepmind.com/research/open-source/kinetics. [Zugriff am 2 September 2021].

[51] Facebook AI, „HMDB51," 2021. [Online]. Available: https://paperswithcode.com/dataset/hmdb51. [Zugriff am 2 September 2021].

[52] Facebook AI, „ActivityNet," 2021. [Online]. Available: https://paperswithcode.com/dataset/activitynet. [Zugriff am 2 September 2021].

[53] Facebook AI, „Something-Something V2 (20BN-Something-Something Dataset V2)," 2021. [Online]. Available: https://paperswithcode.com/dataset/something-something-v2. [Zugriff am 2 September 2021].

[54] IBM, „Overfitting," 2021. [Online]. Available: https://www.ibm.com/cloud/learn/overfitting. [Zugriff am 8 Dezember 2021].

[55] C. Shorten und T. Khoshgoftaar, „A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data,* Bd. 6, 2019.

[56] K. He, X. Zhang, S. Ren und J. Sun, „Deep Residual Learning for Image Recognition," *CoRR,* Bd. abs/1512.03385, 2015.

[57] TensorFlow, „TensorBoard: TensorFlow's visualization toolkit," 2021. [Online]. Available: https://www.tensorflow.org/tensorboard. [Zugriff am 19 November 2021].

[58] ML Glossary, „Loss Functions," 2021. [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html. [Zugriff am 5 Oktober 2021].

[59] OpenAI, „OpenAI," 2021. [Online]. Available: https://openai.com/about/. [Zugriff am 12 September 2021].

[60] OpenAI, „Gym," 2021. [Online]. Available: https://gym.openai.com/. [Zugriff am 12 September 2021].

[61] M. Harris, „An Even Easier Introduction to CUDA," 2017. [Online]. Available: https://developer.nvidia.com/blog/even-easier-introduction-cuda/. [Zugriff am 12 September 2021].

[62] GitHub, „Core dumped with large matmul on aarch64,“ 2022. [Online]. Available: https://github.com/pytorch/pytorch/issues/70350. [Zugriff am 2 Jannuar 2022].

[63] Unbekannt, „pytorch-aarch64,“ 2022. [Online]. Available: https://torch.kmtea.eu/. [Zugriff am 2 Jannuar 2022].

[64] J. Richling und A. Busse, „ARM gestern, heute, morgen,“ *Linux-Magazin,* Bd. 06/13, 2013.

[65] TensorFlow, „Training checkpoints,“ 2021. [Online]. Available: https://www.tensorflow.org/guide/checkpoint. [Zugriff am 20 August 2021].

[66] IBM, „What is computer vision?,“ 2021. [Online]. Available: https://www.ibm.com/topics/computer-vision. [Zugriff am 20 Juni 2021].

[67] GitHub, „About,“ 2021. [Online]. Available: https://github.com/about. [Zugriff am 16 Dezember 2021].

[68] NVIDIA, „Die Geschichte von NVIDIA,“ 2021. [Online]. Available: https://www.nvidia.com/de-de/about-nvidia/corporate-timeline/. [Zugriff am 12 September 2021].

[69] PyPI, „Project description,“ 2021. [Online]. Available: https://pypi.org/project/pip/. [Zugriff am 12 September 2021].

[70] PyTorch, „End-to-end Machine Learning Framework,“ 2021. [Online]. Available: https://pytorch.org/features/. [Zugriff am 16 Dezember 2021].

[71] ROS, „ROS - Robot Operating System,“ 2021. [Online]. Available: https://www.ros.org/. [Zugriff am 9 Juli 2021].

[72] TensorFlow, „About,“ 2021. [Online]. Available: https://www.tensorflow.org/about. [Zugriff am 16 Dezember 2021].

Anhang

Im Anhang werden wichtige zusätzliche Informationen für einzelne Teile der Arbeit gegeben. Dazu zählt die genaue Auswahl der Klassen im selbst erstellten Kinetics-ROSE Datenset, die Beschreibung der durchgeführten Trainingsversuche und deren Fehler und zusätzliche Erklärungen zu den Grundlagen des Machine Learning.

Anhang A – Das Kinetics-ROSE Datenset

In diesem Abschnitt wird die Liste der Klassen aufgeführt, die in das eigene Datenset Kinetics-ROSE aus dem Kinetics-700 übernommen wurden.

Nummer im Kinetics-700	**Bezeichnung im Kinetics-700**	**Nummer im Kinetics-700**	**Bezeichnung im Kinetics-700**
003	"adjusting glasses"	347	"opening door"
006	"answering questions"	348	"opening present"
007	"applauding"	349	"opening refrigerator"
008	"applying cream"	350	"opening wine bottle"
011	"arguing"	351	"packing"
012	"arm wrestling"	358	"peeling apples"
017	"attending conference"	359	"peeling banana"
021	"baking cookies"	360	"peeling potatoes"
022	"bandaging"	362	"petting animal (not cat)"
023	"barbequing"	363	"petting cat"
026	"bathing dog"	366	"photocopying"
028	"beatboxing"	369	"pillow fight"
030	"being excited"	370	"pinching"
033	"bench pressing"	371	"pirouetting"
034	"bending back"	372	"planing wood"
035	"bending metal"	375	"playing accordion"
037	"blasting sand"	378	"playing bagpipes"
038	"blending fruit"	380	"playing bass guitar"
039	"blowdrying hair"	384	"playing cards"
043	"blowing nose"	385	"playing cello"
044	"blowing out candles"	386	"playing checkers"
048	"bottling"	387	"playing chess"

Nummer im Kinetics-700	Bezeichnung im Kinetics-700	Nummer im Kinetics-700	Bezeichnung im Kinetics-700
049	"bouncing ball (not juggling)"	388	"playing clarinet"
050	"bouncing on bouncy castle"	389	"playing controller"
051	"bouncing on trampoline"	391	"playing cymbals"
053	"braiding hair"	393	"playing didgeridoo"
054	"breading or breadcrumbing"	394	"playing dominoes"
057	"breaking glass"	395	"playing drums"
059	"brush painting"	397	"playing flute"
060	"brushing floor"	398	"playing gong"
061	"brushing hair"	399	"playing guitar"
062	"brushing teeth"	400	"playing hand clapping games"
064	"building lego"	401	"playing harmonica"
071	"calculating"	402	"playing harp"
072	"calligraphy"	404	"playing keyboard"
076	"card stacking"	407	"playing lute"
079	"carrying weight"	408	"playing mahjong"
080	"cartwheeling"	409	"playing maracas"
083	"carving pumpkin"	410	"playing marbles"
084	"carving wood with a knife"	411	"playing monopoly"
087	"catching or throwing baseball"	413	"playing nose flute"
088	"catching or throwing frisbee"	414	"playing oboe"
089	"catching or throwing softball"	415	"playing ocarina"
090	"celebrating"	416	"playing organ"
094	"chasing"	418	"playing pan pipes"
096	"checking watch"	419	"playing piano"
101	"chopping meat"	420	"playing piccolo"
104	"clapping"	422	"playing ping pong"

Nummer im Kinetics-700	Bezeichnung im Kinetics-700	Nummer im Kinetics-700	Bezeichnung im Kinetics-700
109	"cleaning shoes"	425	"playing recorder"
110	"cleaning toilet"	428	"playing rubiks cube"
111	"cleaning windows"	429	"playing saxophone"
115	"closing door"	430	"playing scrabble"
116	"coloring in"	435	"playing trombone"
117	"combing hair"	436	"playing trumpet"
120	"cooking chicken"	437	"playing ukulele"
121	"cooking egg"	438	"playing violin"
123	"cooking sausages (not on barbeque)"	440	"playing with trains"
124	"cooking scallops"	441	"playing xylophone"
126	"coughing"	442	"poaching eggs"
128	"country line dancing"	443	"poking bellybutton"
129	"cracking back"	445	"polishing furniture"
130	"cracking knuckles"	446	"polishing metal"
131	"cracking neck"	447	"popping balloons"
133	"crocheting"	448	"pouring beer"
134	"crossing eyes"	449	"pouring milk"
136	"crying"	450	"pouring wine"
139	"curling eyelashes"	451	"preparing salad"
140	"curling hair"	454	"pull ups"
141	"cutting apple"	457	"pumping fist"
142	"cutting cake"	459	"punching bag"
143	"cutting nails"	460	"punching person (boxing)"
144	"cutting orange"	461	"push up"
145	"cutting pineapple"	465	"pushing wheelchair"
146	"cutting watermelon"	467	"putting on eyeliner"
148	"dancing charleston"	468	"putting on foundation"
149	"dancing gangnam style"	469	"putting on lipstick"
150	"dancing macarena"	470	"putting on mascara"
152	"dealing cards"	474	"raising eyebrows"

Nummer im Kinetics-700	Bezeichnung im Kinetics-700	Nummer im Kinetics-700	Bezeichnung im Kinetics-700
153	"decorating the christmas tree"	475	"reading book"
154	"decoupage"	476	"reading newspaper"
157	"dining"	491	"robot dancing"
163	"doing aerobics"	493	"rock scissors paper"
164	"doing jigsaw puzzle"	495	"rolling eyes"
165	"doing laundry"	496	"rolling pastry"
166	"doing nails"	500	"salsa dancing"
167	"doing sudoku"	503	"sanding wood"
168	"drawing"	505	"sawing wood"
169	"dribbling basketball"	506	"scrambling eggs"
175	"drumming fingers"	507	"scrapbooking"
178	"dyeing eyebrows"	508	"scrubbing face"
179	"dyeing hair"	510	"seasoning food"
180	"eating burger"	511	"separating eggs"
181	"eating cake"	512	"setting table"
182	"eating carrots"	513	"sewing"
183	"eating chips"	514	"shaking hands"
184	"eating doughnuts"	515	"shaking head"
185	"eating hotdog"	516	"shaping bread dough"
186	"eating ice cream"	517	"sharpening knives"
187	"eating nachos"	518	"sharpening pencil"
188	"eating spaghetti"	519	"shaving head"
189	"eating watermelon"	520	"shaving legs"
191	"embroidering"	524	"shoot dance"
193	"exercising arm"	530	"shouting"
194	"exercising with an exercise ball"	534	"shuffling cards"
195	"extinguishing fire"	538	"sign language interpreting"
196	"faceplanting"	540	"singing"
198	"falling off chair"	541	"sipping cup"

Nummer im Kinetics-700	Bezeichnung im Kinetics-700	Nummer im Kinetics-700	Bezeichnung im Kinetics-700
199	"feeding birds"	542	"situp"
200	"feeding fish"	553	"slapping"
203	"fidgeting"	555	"sleeping"
204	"filling cake"	556	"slicing onion"
205	"filling eyebrows"	557	"smashing"
206	"finger snapping"	563	"sneezing"
208	"fixing hair"	572	"splashing water"
210	"flipping bottle"	576	"square dancing"
211	"flipping pancake"	577	"squat"
214	"folding clothes"	578	"squeezing orange"
215	"folding napkins"	579	"stacking cups"
216	"folding paper"	580	"stacking dice"
219	"gargling"	581	"standing on hands"
221	"getting a haircut"	582	"staring"
230	"grinding meat"	585	"sticking tongue out"
231	"grooming cat"	587	"stretching arm"
232	"grooming dog"	588	"stretching leg"
236	"hand washing clothes"	593	"sweeping floor"
237	"head stand"	600	"swing dancing"
242	"high fiving"	608	"taking photo"
252	"huddling"	609	"talking on cell phone"
253	"hugging (not baby)"	610	"tango dancing"
255	"hula hooping"	611	"tap dancing"
262	"inflating balloons"	615	"tasting food"
264	"ironing"	618	"texting"
265	"ironing hair"	625	"throwing tantrum"
270	"juggling balls"	627	"tickling"
272	"juggling soccer ball"	632	"tossing coin"
275	"jumping jacks"	633	"tossing salad"
276	"jumping sofa"	634	"training dog"
277	"jumpstyle dancing"	637	"trimming or shaving beard"

Nummer im Kinetics-700	Bezeichnung im Kinetics-700	Nummer im Kinetics-700	Bezeichnung im Kinetics-700
278	"karaoke"	641	"twiddling fingers"
281	"kissing"	645	"tying shoe laces"
283	"knitting"	646	"unboxing"
286	"laughing"	651	"using a power drill"
293	"leatherworking"	652	"using a sledge hammer"
297	"lighting candle"	653	"using a wrench"
298	"lighting fire"	656	"using circular saw"
299	"listening with headphones"	657	"using inhaler"
303	"looking at phone"	660	"using remote controller (not gaming)"
304	"looking in mirror"	663	"vacuuming floor"
306	"lunge"	668	"waking up"
307	"making a cake"	672	"walking with crutches"
308	"making a sandwich"	673	"washing dishes"
315	"making paper aeroplanes"	674	"washing feet"
316	"making pizza"	675	"washing hair"
319	"making sushi"	676	"washing hands"
320	"making tea"	677	"watching tv"
321	"making the bed"	681	"waving hand"
324	"massaging back"	682	"waxing armpits"
325	"massaging feet"	683	"waxing back"
326	"massaging legs"	684	"waxing chest"
327	"massaging neck"	685	"waxing eyebrows"
328	"massaging person's head"	686	"waxing legs"
332	"mixing colours"	690	"whistling"
333	"moon walking"	692	"winking"
334	"mopping floor"	694	"wrapping present"
340	"moving furniture"	696	"writing"
343	"needle felting"	698	"yawning"
345	"opening bottle (not wine)"	699	"yoga"
346	"opening coconuts"	700	"zumba"

Anhang B – Overfitting

Künstliche Neuronale Netze lernen anhand von vielen Beispieldaten eine mathematische Funktion. Je nachdem wie komplex das Netz ist, also wie viele Schichten, Knoten und Gewichte es besitzt, kann es unterschiedlich komplexe Funktionen lernen (siehe Abbildung 26).

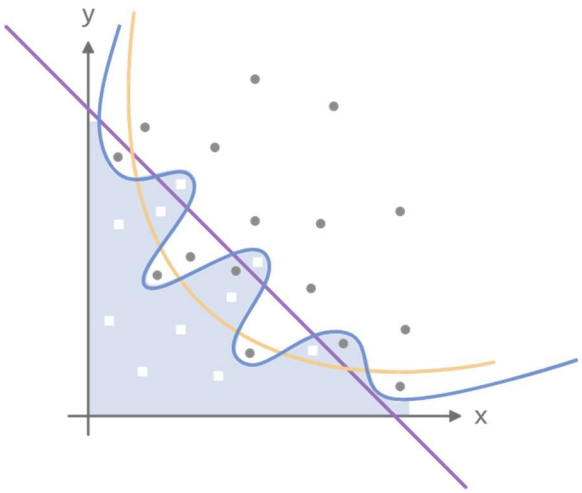

Abbildung 26: Annähern verschieden komplexer Funktionen mit KNNs (Quelle: eigene Abbildung)

Ein einfaches *Multi-Layer Perceptron* mit sechs Schichten könnte aus den gegebenen Daten eine Lineare Funktion lernen (lila), während eines mit sechs Schichten eine Parabel-ähnliche Funktion lernen könnte (gelb). Der AVBERT mit seiner noch komplexeren Architektur würde eine Funktion lernen, die noch stärker an die Beispieldaten angepasst ist (blau). Während des Trainings werden die Funktionen so gut wie möglich an die Trainingsdaten angepasst. Da das Netz aber später in der Lage sein soll, auch unbekannte Daten zu bewerten, werden in der Test-Phase neue Beispiele gezeigt, ohne dass das Netz die Funktion daran anpassen kann (siehe Abbildung 27).

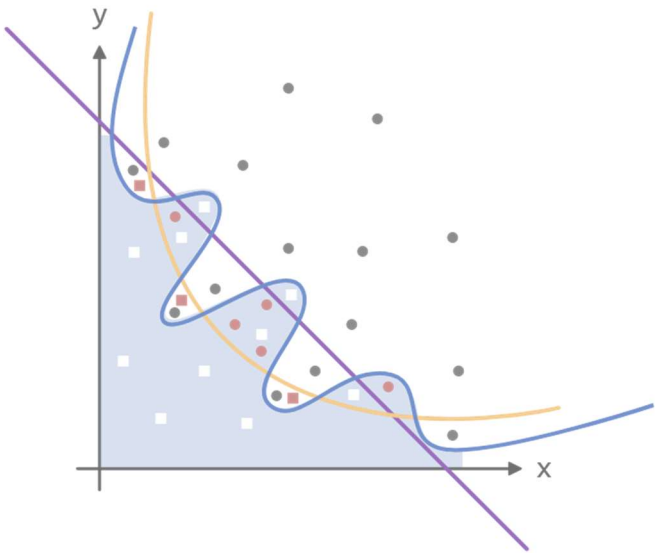

Abbildung 27: Ein weniger komplexes KNN (gelb) kann im Test weniger Fehler machen, als ein komplexes KNN (blau) (Quelle: eigene Abbildung)

Obwohl der AVBERT im Training am besten an die Daten angepasst war, ist seine gelernte Funktion im Test nicht mehr die mit den wenigsten Fehleinschätzungen. Im Bild wurden alle Beispiele, die der AVBERT im Test falsch vorhersagt, das *MLP* mit sechs Schichten aber korrekt bestimmt. Die gelernte Funktion des AVBERT generalisiert nicht gut. Die Generalisierung des Netzes ist aber das Ziel, da es später im Einsatz Daten verarbeiten soll, die es noch nie zuvor bearbeitet hat. Da das Problem der Überanpassung (*Overfitting*) an die Trainingsdaten ein grundlegendes Problem für jedes Netz darstellen kann, gibt es viele Strategien, um dem entgegen zu wirken. Eine einfache Strategie besteht darin, das Netz zu verkleinern, bis es den passenden Grad an Komplexität besitzt. Da die Trainingsdaten die zu lernende Funktion möglicherweise nicht gut genug abbilden, könnte das Netz dadurch allerdings nicht die gewünschte Funktion lernen. Eine andere häufig verwendete Lösung besteht daher in der Vergrößerung des Trainings-Datensets. Diese kann zum Beispiel durch zusätzliche Augmentation-Methoden erreicht werden, ohne mehr Daten beschaffen zu müssen. [13] [54]

Anhang C – Die Trainingsversuche

In diesem Abschnitt werden die Versuche beschrieben, mit denen das Training des AVBERT auf verschiedenen Rechnern erfolgreich und nicht erfolgreich durchgeführt wurde.

Versuchsaufbau 1

Betriebssystem: Windows 10 Enterprise (eigener Rechner)

CPU: Intel Core i7-7800X @ 3.50GHz

RAM: 32 GB

GPUs: NVIDIA GeForce GTX 1080 8GB und NVIDIA GeForce GTX 1070 8GB

Der Code wurde mit dem folgenden Befehl gestartet:

```
python run_net.py \
--cfg_file configs/kinetics-rose/config.yaml \
--configuration kinetics-rose \
--pretrain_checkpoint_path checkpoints/checkpoint.pyth
```

Nach über 36 Stunden war die erste Epoche durchlaufen und der erste Checkpoint gespeichert. Danach wurde das Training abgebrochen, da der Rechner nicht weiterhin dauerhaft laufen konnte und das Training viel zu lange dauerte.

Versuchsaufbau 2

Betriebssystem: Ubuntu 20.04.3 LTS (Focal Fossa) (GPU-Server vom RoboticLab der TH-Wildau)

CPU: Intel Xeon E5-2670 v2 @ 2.50GHz

RAM: 252 GB

GPUs: 4 x NVIDIA K20m Tesla 5 GB

Schritte der Durchführung:

1. Das Kinetics-ROSE-x300 Dataset und den Quellcode mit Hilfe von FileZilla auf den Server übertragen (Dauer: ca. 24 Stunden)
2. Über Miniconda und *pip* alle benötigten Pakete aus der requirements.txt-Datei installieren

3. den Code wie in Versuch 1 ausführen

Nach dem Start wurde in der Konsole die folgende Fehlermeldung angezeigt:

```
/home/l       /miniconda3/lib/python3.8/site-packages/torch/cuda/__i
    Found GPU%d %s which is of cuda capability %d.%d.
    PyTorch no longer supports this GPU because it is too old.
    The minimum cuda capability supported by this library is %d.%d.

  warnings.warn(old_gpu_warn.format(d, name, major, minor, min_arch
```

Der Versuch wurde allerdings nicht sofort abgebrochen, da auf einer GitHub-Seite eine mögliche Lösung für diesen Fehler gefunden wurde. Diese Möglichkeit bestand darin, *PyTorch* für die vorhandene *CUDA*-Version 3.5 selbst zu bauen. Dabei trat jedoch der folgende Fehler auf:

```
CMake Error at cmake/public/cuda.cmake:49 (message):
  PyTorch requires CUDA 10.2 or above.
```

Danach musste der Versuch abgebrochen und ein alternativer GPU-Server gesucht werden.

Versuchsaufbau 3

Betriebssystem: Windows Server 2016

CPUs: 2 x Intel Xeon E5-2603v4 @ 1.70GHz

RAM: 2 x 16 GB

GPUs: NVIDIA GeForce GTX 1080 8GB und NVIDIA GeForce GTX 1070 8GB

Während nach einem anderen GPU-Server gesucht wurde, konnte ein weiterer Versuch auf einem eigenen Server gestartet werden. In diesen Server wurden die beiden GPUs aus dem Rechner aus Versuch 1 eingebaut. Die Daten wurden übertragen, die Pakete installiert und das Training gestartet.

Nach einer unbestimmten Zeit brach das Training mehrmals mit folgendem Fehler ab:

```
DefaultCPUAllocator: not enough memory
```

Daraufhin wurde zunächst die Batch-Size verringert, um den gleichzeitig benötigten Speicher zu reduzieren. Beim nächsten Trainingsversuch trat der Fehler erst nach einigen Stunden auf. Es konnte jedoch keine Epoche erfolgreich trainiert werden.

Versuchsaufbau 4

Betriebssystem: Debian GNU/Linux 10 (GPU-Server der TH-Wildau)

CPU: Intel Xeon Gold 5218 @ 2.30GHz

RAM: 376 GB

GPUs: 6 Tesla V100-SXM2 32GB

Schritte der Durchführung:

1. Das Kinetics-ROSE-x300 Dataset und den Quellcode mit Hilfe von scp auf den Server übertragen (Dauer: ca. 45 Minuten, da sich beide Server im Netzwerk der TH Wildau befinden)
2. Proxy-Einstellungen für die Installation von Python-Packages vorgenommen
    a. Für conda den Proxy in die .arcconda-Datei eintragen
    b. *pip* mit --proxy aufrufen:

```
pip install --proxy=http://proxy.th-wildau.de:8080/
```

3. über conda die folgenden Pakete installieren

```
conda install python=3.8
conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch
conda install -c fvcore -c iopath -c conda-forge fvcore
```

4. über *pip* das folgende Paket installieren

```
pip install --proxy=http://proxy.th-wildau.de:8080/ install av
```

5. das Training als Linux-Job mit nohup starten

```
nohup python run_net.py \
--cfg_file configs/kinetics-rose/config.yaml \
--configuration kinetics-rose \
--pretrain_checkpoint_path checkpoints/checkpoint.pyth &
```

Mit dieser Konfiguration konnte das Training erfolgreich durchlaufen werden. Nach 6 Tagen und 6 Stunden waren 20 Epochen beendet.

Anhang D – PyTorch auf ROS-E

In diesem Abschnitt werden die Probleme und Lösungsansätze beschrieben, die bei der Installation von *PyTorch* und der Ausführung des Codes des AVBERT auf dem Raspberry Pi aufgetreten sind.

Ansatz 1 – Installation über pip

Für das Training des AVBERT wurden die Versionen 1.10.0 von torch, 0.10.0 von torchaudio und 0.11.1 von torchvision mit *CUDA*-Unterstützung verwendet. Da mit diesen Paketen keine Probleme auftraten, wurden diese Versionen auf dem Raspberry Pi ebenfalls über *pip* installiert. Damit konnte der Code für den Reinforcement-Ansatz mit dem DQN-Algorithmus und den zufällig generierten Belohnungen und dem mit Nullwerten gefüllten *Representation-Vector* ohne Fehler ausgeführt werden. Als der *Representation-Vector* im nächsten Schritt jedoch mit dem AVBERT erzeugt wurde, trat folgender Fehler auf:

```
OSError: /usr/local/lib/python3.8/dist-packages/torchaudio-
0.10.0+4b64f80-py3.8-linux-
aarch64.egg/torchaudio/lib/libtorchaudio.so: undefined symbol:
_ZNK5torch8autograd4Node4nameB5cxx11Ev
```

Der erste Lösungsansatz bestand darin, die Pakete nicht mit *CUDA*-Unterstützung, sondern in der Version für das Rechnen auf einer CPU zu installieren. Nachdem die Pakete ohne *CUDA* installiert wurden, trat der Fehler jedoch wieder auf. Der zweite Lösungsansatz bestand darin, das Paket torchaudio, das den Fehler verursacht hat, selbst zu kompilieren. Da der Raspberry Pi im Gegensatz zu den zuvor verwendeten Rechnern mit der ARM-Architektur arbeitet, wird der Code durch das Kompilieren auf die verwendete CPU und deren Architektur angepasst.

Im erneuten Test brach das Programm anschließend nicht mehr gleich zu Beginn ab, sondern lief bis zu dem Punkt, an dem die *Spektrogramme* vor der Verarbeitung durch den AVBERT erstellt wurden. An diesem Punkt wurde folgender Fehler ausgegeben:

```
Illegal Instruction (core dumped)
```

Ansatz 2 – PyTorch und torchaudio für ARM selbst bauen

Nachdem der Fehler im Internet recherchiert wurde, konnte festgestellt werden, dass *PyTorch* ebenfalls speziell für die ARM-Architektur gebaut werden musste. Daraufhin wurde versucht, *PyTorch* direkt auf dem Raspberry selbst zu bauen. Dieser Vorgang dauerte über 8 Stunden. In einem anschließenden Test trat derselbe Fehler auf.

Da die Fehlermeldung nicht aussagekräftig war, wurde das Programm mit gdb, dem Debugger von Linux analysiert. Durch das Debugging konnte der Fehler genauer ermittelt werden:

```
Thread 1 "python" received signal SIGILL, Illegal instruction.

0x0000fffff3e40a78 in exec_blas () from /usr/local/lib/python3.8/dist-
packages/torch/lib/libtorch_cpu.so
```

Dieser Fehler wurde in einer Issue (Problem-Meldung) auf der GitHub-Seite von *PyTorch* gefunden. Als mögliche Ursache wurde wieder die ARM-Architektur genannt, mit der Lösung, auch *PyTorch* selbst zu bauen bzw. eine auf dem Raspberry Pi gebaute Version zu verwenden. Daraufhin musste der Versuch aus Mangel an Zeit und Lösungsansätzen, die noch nicht probiert wurden, abgebrochen werden. [62]