Institut für Software Engineering

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

System-Theoretic Process Analysis for AI-Based Systems

Eva Zimmermann

Studiengang: Softwaretechnik

Prüfer/Betreuer: Prof. Dr. Stefan Wagner

Beginn am: 01. Juni 2022

Beendet am: 01. Dezember 2022

Kurzfassung

Durch die fortschreitenden Technologien wird künstliche Intelligenz in immer mehr Bereichen eingesetzt. Dabei rücken auch sicherheitskritische Bereiche immer mehr in den Vordergrund. Um die Sicherheit in diesen Bereichen zu gewährleisten, müssen Sicherheitsanalysen durchgeführt werden. In dieser Masterarbeit wird hierfür STPA (System Theoretic Process Analysis) genutzt. Die Fallstudie, auf die STPA angewendet werden soll, muss sicherheitskritische KI-Teile enthalten und Open-Source zur Verfügung stehen. Dabei fiel die Entscheidung auf das Start-up Comma AI und im Speziellen ihre über Github frei zugängliche Open-Source-Software Openpilot. Nach der Durchführung der STPA-Analyse von Openpilot wurde festgestellt, dass vor allem die Charakteristik des neuronalen Netzes, da es sich hierbei um eine Blackbox handelt, schwierig in Loss Scenarios abbildbar ist. So ist vor allem Verhalten, dass sich über Zeit ändert und nicht direkt zu einem Loss Scenario führt, problematisch. Um diese Problematik anzugehen, wurden erste Lösungsansätze diskutiert. Zunächst wurden dazu die bereits vorhandenen Standards zur funktionalen Sicherheit (ISO 26262, ISO 21448) im Automotive Bereich betrachtet und eine Kombination von ihnen mit STPA, da die Standards ebenfalls noch nicht auf ki-basierte Systeme ausgelegt sind. Anschließend wird vorgestellt, wie aus den STPA Bestandteilen ein Monitoringsystem aufgebaut werden kann, um die Entscheidungen der neuronalen Netze, falls sie unsichere Control-Actions liefern sollten, in sichere zu überführen. Als dritte Möglichkeit wurde ein Blackbox Assessment anhand von Openpilot aufgezeigt. Diese Ansätze können in zukünftigen Arbeiten evaluiert werden, um deren Verwendbarkeit zu zeigen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ETA Event Tree Analysis. 22

FMEA Failure Modes and Effects Analysis. 22

FTA Fault Tree Analysis. 19

GNSS Global Navigation Satellite System. 35

HAZOP Hazard and Operability Analysis. 19

KI Künstliche Intelligenz. 13

ML Maschine Learning. 15

MPC Model Predictive Controller. 41

ROS Robot Operating System. 32

SOTIF Safety of the Intended Functionality. 63

STAMP System Theoretic Accident Model and Process. 15

STPA System-Theoretic Process Analysis. 7

1 Einleitung

Künstliche Intelligenz (KI) hält immer mehr Einzug in unser Leben und in unseren Alltag. Durch den Fortschritt der Technologie wird diese immer häufiger eingesetzt, da ihr Funktionsbereich immer größer wird. So kommt es ebenfalls dazu, dass sie vermehrt in sicherheitskritischen Bereichen verwendet werden. Bei diesen ist es besonders wichtig, die Sicherheit genau zu analysieren und zu betrachten, da sonst unter anderem Menschen zu Schaden kommen können.

Um Sicherheit zu evaluieren, gibt es verschiedene Möglichkeiten. Im Automobilbereich wurde 2020 ein Bericht veröffentlicht [PEG20], der sich mit dem Thema *Projekt zur Etablierung von generell akzeptierten Gütekriterien, Werkzeugen und Methoden sowie Szenarien und Situationen zur Freigabe hochautomatisierter Fahrfunktionen (kurz PEGASUS)* beschäftigt und die Analyse führte zu dem Schluss, dass STPA genutzt werden sollte, da hierdurch, die Probleme die zwischen verschiedenen Komponenten auftreten können, am besten betrachtet werden. Doch ob STPA alle Mittel besitzt, um KI Anwendungen auf Sicherheit zu untersuchen, soll in dieser Arbeit betrachtet werden.

1.1 Aufgabenstellung

Ziel dieser Arbeit ist es, zunächst eine Fallstudie zu identifizieren, die die Kriterien "sicherheitskritische KI-Teile" und "Open-Source" erfüllt. Auf dieser soll anschließend eine STPA-Analyse angewandt und die Limitierungen in Bezug auf STPA aufgezeigt werden. Dabei werden die Herausforderungen, die durch den Einsatz von KI bzlg. Safety auftreten thematisiert und anschließend diskutiert, wie diese gelöst werden können. Dazu werden mögliche Erweiterungen bzw. Ergänzungen für STPA aufgezeigt. Zur Erfüllung der Aufgabenstellung wurde die folgende Gliederung erstellt.

1.2 Gliederung der Arbeit

Kapitel 2: beschreibt die Grundlagen, die für die Masterarbeit notwendig sind.

Kapitel 3: zeigt den Prozess der Wahl der Fallstudie auf und diese wird vorgestellt und inhaltlich thematisiert.

Kapitel 4: stellt die Durchführung der STPA-Analyse dar.

Kapitel 5: bildet die Diskussion und zeigt die Rückschlüsse auf, in welchen Bereichen STPA noch erweitert werden muss. Hierfür werden anschließend drei mögliche Erweiterungen diskutiert. Aufgrund der Diskussion der Nutzung von verwandten Arbeiten, werden diese nicht in einem eigenen Kapitel, sondern in diesem Kapitel diskutiert.

Kapitel 6: fasst zunächst die Arbeit zusammen und zieht ein Fazit und beschreibt welche zukünftigen Arbeiten als nächstes umgesetzt werden müssen.

2 Grundlagen

Zunächst werden die grundlegenden Themen behandelt, die für die Masterarbeit relevant sind. Dabei wird die KI thematisiert und kurz vorgestellt. Anschließend wird das in der Systemtheorie verwendete Sicherheitskonzept System Theoretic Accident Model and Process (STAMP) betrachtet und die darauf aufbauende Systemanalyse STPA. Aufgrund des automobilen Kontextes werden anschließend noch die verschiedenen Level des automatisierten Fahrens betrachtet.

2.1 Grundlagen der künstlichen Intelligenz

Die KI ist ein Teilgebiet der Informatik. Dabei wird die *Automatisierung von intelligentem Verhalten* betrachtet. Eine Definition des Begriffes der Intelligenz ist allerdings nicht möglich, da auch andere Wissenschaften, wie die Biologie oder Neurowissenschaften, noch keine präzisen Definitionen dafür gefunden haben [KS19].

Zunächst wird der zeitliche Verlauf der Entwicklung der KI betrachtet. Begonnen hat die KI mit aussagelogischen Hypothesen in den 1950ern. Hierbei gab es, wie in neuen Themenfeldern üblich, zunächst einige Rückschläge. Ein zeitlicher Ablauf der Entwicklung des Themenfeldes der KI wird in Abbildung 2.1 aufgezeigt [KS19].

Die nächste größere Entwicklung fand in den 1980ern statt, in denen die Basis für die Ansätze des Maschine Learning (ML) gelegt wurden. Dabei ist die grundlegende Idee, dass man ein Programm erstellt, das eine spezifische Aufgabe erhält, die erfüllt werden soll. Auf der Grundlage der gewonnenen Erkenntnisse soll anschließend ein Lernprozess stattfinden. Dieser hilft dann, die Aufgaben zukünftig besser bewältigen zu können. Der Unterschied zu den Programmen, die keine KI beinhalten, wird in Abbildung 2.2 gezeigt. Der Ablauf eines *traditionellen Programms* ist, dass zu Beginn vorhandene Daten in statischen Codes verarbeitet werden und das Ergebnis anschließend ausgegeben wird [KS19].

Bei ML Anwendungen wird auf der bereitgestellten Datenmenge zunächst ein Algorithmus ausgeführt. Aus dessen Ergebnis wird eine Hypothese getroffen, auf die dann das Ergebnis folgt. Auf der Grundlage dieses Ergebnisses wird nun ein Feedback an den Algorithmus gegeben, welches die Rückkopplung für den Lernprozess darstellt [KS19].

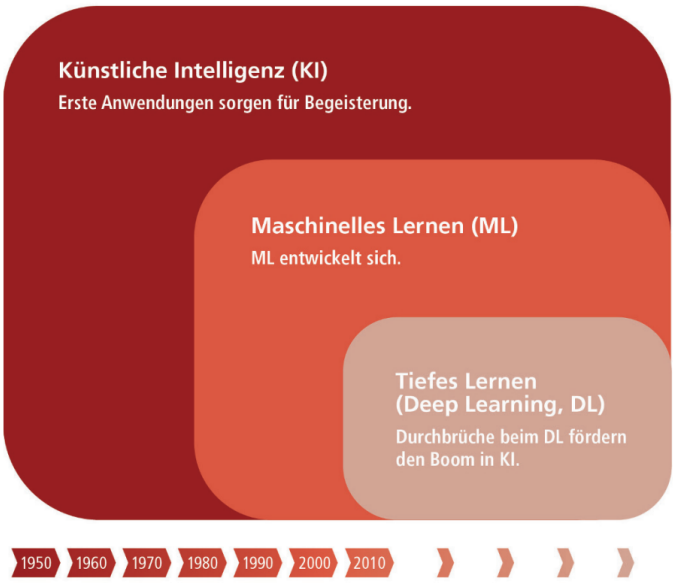

Abbildung 2.1: Übersicht der Entwicklung des Themenfeldes KI (übernommen aus [KS19]).

ML lässt sich in vier Teile unterteilen: das überwachte Lernen (supervised Learning), das unüberwachte Lernen (unsupervised Learning), das semi-überwachte Lernen (semi-supervised Learning) und das verstärkte Lernen (reinforcement Learning). Sie werden im folgenden kurz erläutert:

- *supervised Learning*: Beim supervised Learning handelt es sich um eine Lernmethode, in der dem Algorithmus sowohl Daten, die als Input dienen, als auch Daten, die den korrekten Output zeigen, gegeben werden, damit der Algorithmus einen Vergleich für einen korrekten Output hat. Realisiert wird dies durch Labels, die den Trainingsdaten (Input und Output Paare) zugeordnet werden. Durch diese Möglichkeit kann der Algorithmus das erzielte Ergebnis mit dem mitgelieferten Output vergleichen und das Modell so anpassen, dass die Ergebnisse korrekt werden. Ein Output, der beim supervised Learning entsteht, kann beispielsweise eine Klassifikation sein oder auch eine Regression [Ong17]. Ein gängiges Beispiel wäre hierbei die Klassifikation von Tierbildern [ESB20].
- *unsupervised Learning*: Unsupervised Learning behandelt Daten, in denen keine Output-Daten als Kontrolle zur Verfügung gestellt werden. Anders ausgedrückt handelt es sich hierbei um nicht gelabelte Daten. Diese werden dazu genutzt, um beispielsweise in großen Datenmengen Muster zu finden. Eine der bekanntesten Zuordnungsmöglichkeiten ist k-means. Das Ziel hierbei ist es, eine Art von Struktur oder Muster in den vorliegenden Daten zu finden [Ong17].
- *semi-supervised Learning*: Semi-supservised Learning wird bei denselben Anwendungsfällen verwendet, wie das supervised Learning. Dabei wird aber nur eine geringe Menge an gelabelten Daten (Input mit korrektem Output), sowie eine große

Menge ungelabelte Daten (wie beim unsupervised Learning) verwendet. Dies hat den Vorteil, dass keine großen Datensätze mit gelabelten Daten benötigt werden, sondern nur wenige. Dadurch wird der Aufwand verringert. Eine Verwendung ist hierbei, wie beim supervised Learning, zum Beispiel die Klassifizierung von Daten [Ong17].

- *reinforcement Learning*: Reinforcement Learning wird häufig in der Robotik eingesetzt. Dabei wird das "Trial-and-Error"-Prinzip angewandt. Durch dieses kann herausgefunden werden, welche Entscheidungen die besten Ergebnisse liefern. Es werden hierbei beim Lernen drei Komponenten verwendet: Die Erste ist *der Agent*, der lernen soll. Als Zweites wird *die Umwelt* benötigt, mit der der Agent interagiert. Als Drittes gibt es im Reinforcement Learning noch *die Aktionen*, die alle Aktionen umfasst, die der Agent ausführen kann. Ziel ist es, den maximalen Erfolg aus den ausgeführten Aktionen, in einer bestimmten Zeit, zu erhalten und ermittelt hiermit die besten Ergebnisse [Ong17].

Durch die weiteren Forschungen, die im KI-Themenfeld getätigt wurden, wurde seit dem Jahr 2010 ein weiterer Ansatz immer populärer. Dieser wird als Deep Learning bezeichnet. Dabei geht es vor allem um Deep Learning mit *künstlichen neuronalen Netzen (KNN)*. Die Prozesse, die in einem neuronalen Netz stattfinden, wurden aus der Biologie und den Neurowissenschaften motiviert. Dabei kommt die Bezeichnung *Deep* daher, dass es in einem neuronalen Netz verschieden viele Schichten gibt, die eine Tiefe bieten. Kurzgefasst

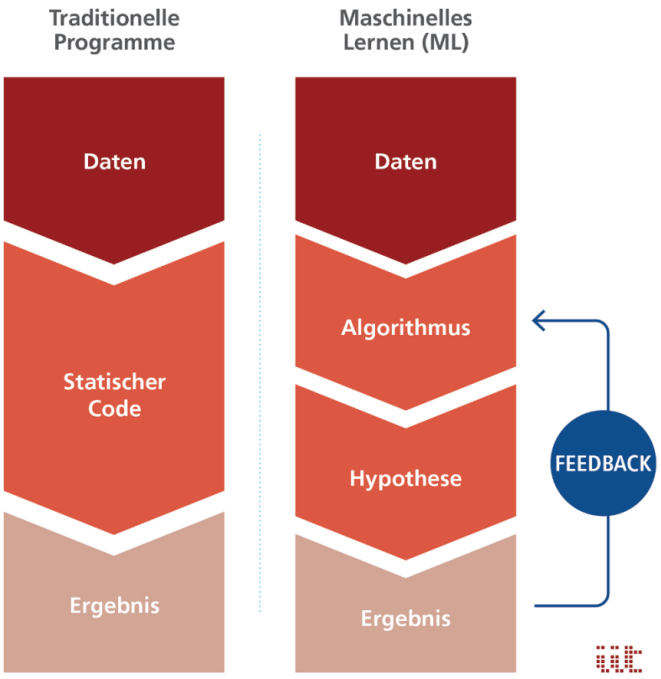

Abbildung 2.2: Übersicht für den Unterschied zwischen traditionellen und ML-Programmen (übernommen aus [KS19]).

nutzt ein neuronales Netzwerk Eingabewerte und führt darauf Berechnungen durch und gibt dann Ausgabewerte aus, wie in Abbildung 2.3 dargestellt ist. Detailliert sieht der Aufbau eines neuronalen Netzwerkes wie folgt aus: Zunächst gibt es eine *Eingabeschicht,* in der Daten eingelesen werden. Ein Beispiel hierfür sind Bilddaten, die eingelesen werden. Das Ergebnis wird in der *Ausgabeschicht* ausgegeben, also ob beispielsweise ein Fahrzeug vor dem Gesteuerten fährt oder nicht. Der Prozess, der zwischen der Eingabe und der Ausgabe stattfindet, ist die Ausführung der sogenannten *versteckten Schichten (hidden layers)*. Dabei können beliebig viele Schichten (1...n), eingefügt werden (Abbildung 2.3). Hierbei können von Schicht zu Schicht 1...n Vorgänger als Input Neuronen für ein Neuron genutzt werden. Damit das neuronale Netz weiß, wie wichtig die ankommenden Neuronen sind, werden diese gewichtet. Eine zusätzliche Möglichkeit ist es, eine Rückkopplung in einem neuronalen Netz zu haben, damit Informationen aus späteren Schichten wieder zurückgegeben werden können (*rekurrente Netze*) [KS19].

Ein neuronales Netz muss, um seine Aufgaben erfüllen zu können, trainiert werden. Das Training ist notwendig, damit Gewichtungen bestimmt und präzisiert werden können. Die neuronalen Netzwerke sind deswegen während des Trainings deutlich rechenintensiver, als während ihrem späteren Einsatz.

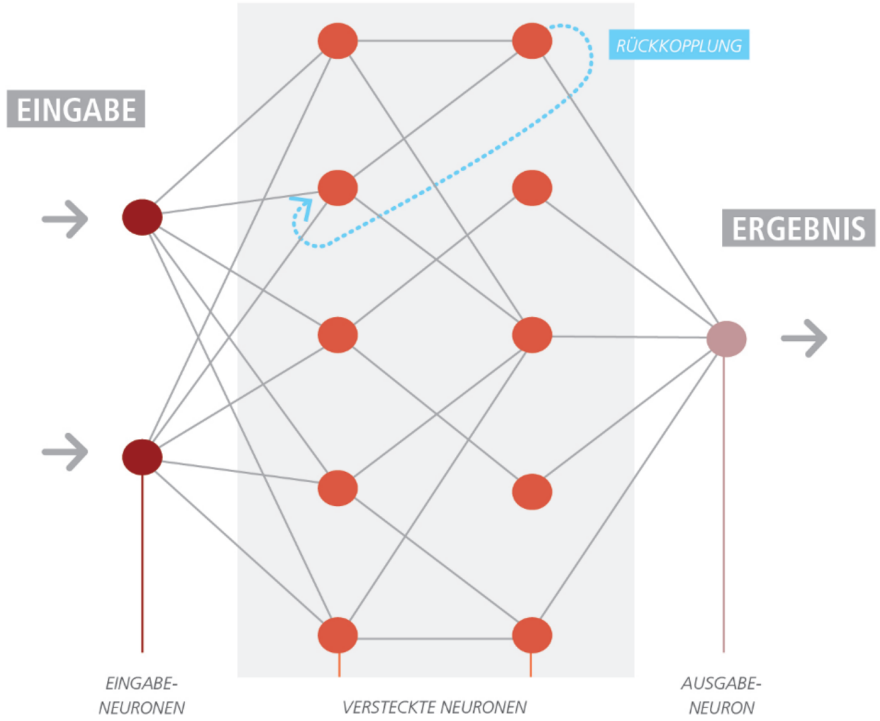

Abbildung 2.3: Aufbau eines neuronalen Netzes mit zwei versteckten Schichten sowie einer Eingabe- und Ausgabeschicht (übernommen aus [KS19]).

2.2 STAMP

STAMP ist ein *Accidentmodell*, das in der Systemtheorie angesiedelt ist. Vorgestellt wurde es von N. Levenson [Lev11] im Buch *Engineering a Safer World Systems Thinking Applied to Safety*, die dieses systemtheoretische Modell entwickelt hat. Dabei sind die essenziellen Aspekte der Systemtheorie, dass das System als Ganzes betrachtet wird und nicht in einzelnen Teilen (*"the whole is more then the sum of its parts"*). Dabei geht es in der Betrachtung darum, emergente Eigenschaften mitzuberücksichtigen, die sich durch das Zusammenspiel der Komponenten ergeben. Es werden also die Interaktionen zwischen den Komponenten betrachtet. Dadurch müssen hier, neben der technischen Umsetzung, auch soziale Faktoren miteinbezogen werden [LT18].

Die Systemtheorie, die STAMP zugrunde liegt, wurde in den 1950ern und 1960ern nach dem Zweiten Weltkrieg entwickelt. Ursache dafür war die steigende Komplexität der Systeme. Erste Anwendungen fand sie schließlich bei Ingenieuren, die Luftabwehrraketen und Frühwarnsysteme entwickelten.

Im Gegensatz zu den traditionellen Sicherheitsanalysetechniken, wie Fault Tree Analysis (FTA) und Hazard and Operability Analysis (HAZOP), bietet STAMP entscheidende Vorteile. Da STAMP für komplexe Systeme geschaffen wurde, ist es hier sehr gut anwendbar. Das System wird hierbei zunächst abstrahiert betrachtet und dann von oben nach unten in die Details durchgearbeitet, nicht wie in den anderen Techniken von unten nach oben. Außerdem wird, zusätzlich zum konkreten System, auf beteiligte Personen Bezug genommen und Sicherheitskulturen, sowie weitere Faktoren betrachtet. Jene weiteren Faktoren müssen somit nicht separat identifiziert und betrachtet werden, falls sie eine Rolle für den Unfall spielen [LT18].

STAMP besteht aus drei Hauptbestandteilen: Sicherheitsconstraints, eine hierarchische Control-Structure, sowie Prozessmodelle. Diese werden im Folgenden beschrieben [Lev11].

2.2.1 Sicherheitsconstraints

Die Grundlage des STAMP Prozesses bilden nicht, wie bei anderen Sicherheitsanalysen Events, sondern Sicherheitsconstraints. Dabei wird zwischen *passiven und aktiven Controls* differenziert [Lev11].

Die *passiven Controls* sind Elemente, die bereits durch ihre Verbauung in das zu analysierende System Sicherheit gewährleisten. Dies bedeutet, dass in einem Fehlerfall das System in einen sicheren Zustand überführt wird. Die Alternative zu dieser Überführung sind sogenannte Sperrvorrichtungen (interlocks). Diese reglementieren die Interaktionen, die zwischen den vorhandenen Komponenten innerhalb eines Systems auftreten, sodass nur sichere Interaktionen durchgeführt werden können [Lev11].

Das zweite Konzept sind *aktive Controls*, die Maßnahmen benötigen, um Sicherheit zu bieten. All die Maßnahmen müssen hierbei vorhanden sein, bevor der Loss auftritt, um ihn korrekt in einen sicheren Zustand zu überführen. Dafür müssen die Gefährdungen (Hazards), die auftreten können, überwacht und identifiziert werden. Dies geschieht, indem verschiedene Variablen bzw. Zustände gemessen werden. Anschließend werden die gemessenen Ergebnisse interpretiert und Rückschlüsse darüber gezogen, wie diese behandelt werden müssen, um den Loss der sonst auftreten würde, zu verhindern [Lev11].

2.2.2 hierarchische Control-Structure

STAMP bildet das betrachtete System als eine hierarchische Struktur ab, wie es in der Systemtheorie vorgesehen ist. Dies bedeutet, dass das System als eine Struktur angesehen wird, die verschiedene Level beinhaltet und es höher liegende (*Controller*) und darunter liegenden Ebenen (*Controlled Process*) gibt. Dabei wird mit der niedrigeren Ebene durch *Control-Actions* kommuniziert, welche wiederum mittels *Feedback* antworten können. Dadurch setzen *Control-Actions* also die vorher diskutierten Sicherheitsconstraints (Abschnitt 2.2.1) durch. Zusammengefasst wird die Kommunikation zwischen den einzelnen Komponenten in Abbildung 2.4.

Ein Unfall, also das Auftreten eines Losses, kann infolgedessen eintreten, wenn die Controlled Processes in ihrem Verhalten die notwendigen Sicherheitsconstraints nicht weitergegeben oder diese auf der darunterliegenden Ebene bspw. nicht korrekt ausgeführt werden. Auch kann die Sicherheit gefährdet werden, wenn ein Sicherheitsconstraint nicht definiert wird und daher fehlt, oder auch wenn Änderungen für die Constraints (z. B. Gesetzesänderungen) mit der Zeit auftreten und so im System nicht beachtet werden [Lev11].

2.2.3 Prozessmodelle

Zusätzlich zu den Sicherheitsconstraints und der hierarchischen Control-Structure werden noch die Prozessmodelle benötigt. Diese werden als Modelle innerhalb eines Controllers genutzt. Ein Controller kann sowohl einen Menschen darstellen, als auch eine Maschine. So bildet das Prozessmodell bei einem Menschen ein mentales Modell ab, während bei einer Maschine eine definierte Logik verwendet wird. Bei beiden handelt es sich um ein Modell, anhand dessen Entscheidungen getroffen werden, welche Control-Action an den Controlled Process geschickt werden soll und wie diese aussieht. Um dies umzusetzen, benötigt ein Prozessmodell Variablen, die die gegebenen Zustände oder Werte abbilden, sowie deren aktuelle Werte oder Zustände und das Wissen darüber, wie diese verändert werden können. Diese können dann durch eine Control-Action, nach bestem Gewissen des Controllers, so abgeändert werden, dass der notwendige Sicherheitsconstraint durchgesetzt werden kann. Anschließend kann der Controlled Process ein Feedback zurücksenden, das bspw. den aktuellen Wert der veränderten Variable enthält [Lev11].

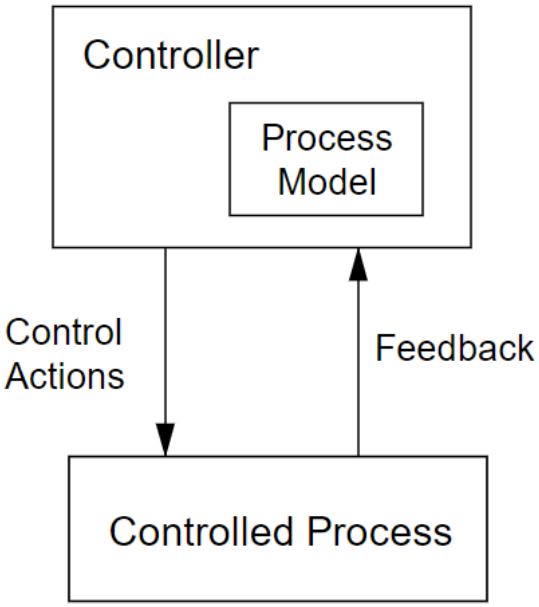

Abbildung 2.4: Überblick über eine Control-Structure und deren Bestandteile (übernommen von N. Levenson [Lev11]).

Die Kommunikation und Interaktion zwischen einem Controller und einem Controlled Process ist von großer Bedeutung, wenn Unfälle auftreten, da hier die Ursachen dafür liegen. Die entscheidende Rolle spielt hierbei das Prozessmodell, denn durch dieses kann auf fehlerhafte Interaktionen geschlossen werden. So können laut N. Levenson [Lev11, Seite 88], die folgenden Probleme ausschlaggebend sein:

1. *"Incorrect or unsafe control commands are given*
2. *Required control actions (for safety) are not provided*
3. *Potentially correct control commands are provided at the wrong time (too early or too late), or*
4. *Control is stopped too soon or applied too long."*

Auf der Grundlage von STAMP wurden mächtige Instrumente für die Durchsetzung und Beachtung von Sicherheit im System (STPA) und für die Analyse von solchen Kommunikationsfehlern, nach dem Auftreten eines Unfalls (CAST) entwickelt.

2.3 Der STPA-Prozess

STPA hat, da es auf STAMP basiert, einen anderen Ansatz für eine Sicherheitsanalyse, wie die traditionellen Analysen (bspw. Failure Modes and Effects Analysis (FMEA) und FTA). Durch die zusätzliche Betrachtung von Interaktionen zwischen verschiedenen Komponenten, ist es für Sicherheitsanalysen, die vor allem komplexere Systeme betrachten, von großer Relevanz. STPA findet, laut N. Leveson [LT18], dann mehr Sicherheitsrisiken, wie FTA, FMECA, HAZOP und Event Tree Analysis (ETA), wenn Software eine Rolle spielt. Somit ist STPA für die Zukunft eine wichtige Analysemethode, die auch in Bezug auf das automatisierte Fahren (welches in dieser Arbeit im Fokus steht) genutzt werden kann, um möglichst viele Sicherheitsrisiken aufzudecken. Daher wird STPA nun ausführlich erklärt [SLZ21].

Grundsätzlich besteht der Prozess aus vier Schritten, die in Abbildung 2.5 zusammengefasst werden.

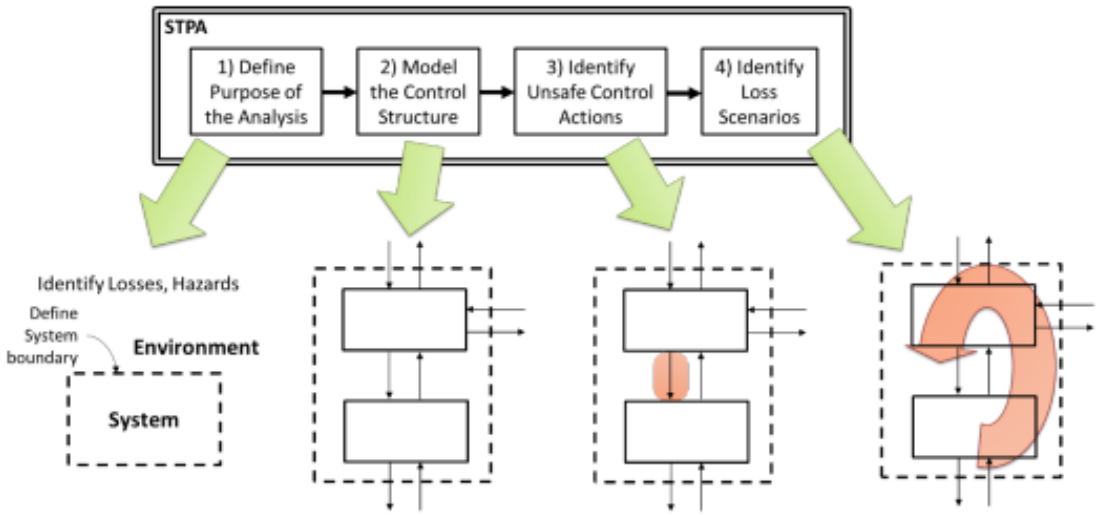

Abbildung 2.5: Übersicht des STPA Prozesses (übernommen von N. Levenson [LT18]).

2.3.1 Schritt 1 - Definition der Ziele der Analyse

Der erste Schritt der Analyse ist es, die Basis für die folgenden Schritte, zwei bis vier, zu erstellen. Dabei werden zunächst die Losses beschrieben, die auftreten können. Danach werden die System-Level Hazards, sowie die korrespondierenden System-Level Constraints dokumentiert.

Loss

Definition - *"A **loss** involves something of value to stakeholders. Losses may include a loss of human life or human injury, property damage, environmental pollution, loss of mission, loss of reputation, loss or leak of sensitive information, or any other loss that is unacceptable to the stakeholders" [LT18].*

Der erste Schritt der Analyse ist es, die Losses zu definieren. Um diese identifizieren zu können, ist es zunächst unabdingbar, die Stakeholder und deren Interessen festzustellen. Durch diese lassen sich im Anschluss die Werte, die jeder einzelne schützen will, ermitteln. Die zu schützenden Werte werden dann in Losses umgewandelt. Nach der Erstellung aller Losses werden diese nach Wichtigkeit priorisiert [LT18].

Ein Beispiel für die zu schützenden Werte ist ein Menschenleben. Ein Autohersteller hat Interesse daran, dass bei einem autonom fahrenden Fahrzeug Menschen bestmöglichst geschützt werden sollen. Daher ist der daraus abgeleitete Loss [LT18]:

L-1: Verlust eines Menschenlebens.

System-Level Hazard

Definition: *" A **system** is a set of components that act together as a whole to achieve some common goal, objective, or end. A system may contain subsystems and may also be part of a larger system." [LT18].*

Definition - *" A **hazard** is a system state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to a loss." [LT18].*

Als Nächstes werden die Hazards definiert, die auftreten können. Dazu muss das System identifiziert werden und gegebenenfalls die darin bestehenden Subsysteme [LT18].

Im Gegensatz zu einem Loss, der sowohl innerhalb eines Systems, als auch in der Umwelt auftreten kann, tritt ein Hazard innerhalb des Systems auf. Auf die Umwelt hat man kaum bzw. keinen Einfluss. Ein Hazard bildet also nur Vorgänge bzw. Zustände ab, die innerhalb des Systems vor sich gehen. Diese werden identifiziert, um sie zu minimieren oder falls möglich zu entfernen [LT18].

Hazards werden als Bedingungen oder Zustände definiert, die dann in Kombination mit " worst-case" Zuständen in der Umwelt zu einem Loss führen können, aber nicht müssen. Ein Hazard wird so beschrieben, dass er exakt den Zustand abbildet, der verhindert werden muss [LT18].

Beispielsweise, wenn ein autonom fahrendes Fahrzeug sich im Straßenverkehr bewegt, kann der Loss wie bereits angegeben **L-1: Verlust eines Menschenlebens** sein. Ein dazugehöriger Hazard wäre [LT18]:

H-1: Das Fahrzeug kollidiert mit einem anderen Fahrzeug [L-1].

System-Level Safety Constraint

Definition - *"A **system-level constraint** specifies system conditions or behaviors that need to be satisfied to prevent hazards (and ultimately prevent losses). " [LT18].*

Die System-Level Constraints sind die Negierung der System-Level Hazards, denn ein Constraint ist die durchzusetzende Bedingung. Die Constraints definieren, wie im System reagiert werden muss, wenn ein Loss auftritt. In weiteren Teilen der Analyse werden Szenarien aufgestellt, um Verletzungen, der in diesem Schritt aufgestellten Constraints, zu identifizieren [LT18].

SC-1: Das Fahrzeug muss einen Mindestabstand zu anderen Objekten einhalten [H-1].

Verfeinerung der System-Level Hazards

Schritt 1 beinhaltet noch einen zusätzlichen Schritt, der optional die Verfeinerung von System-Level Hazards vorsieht. Dieser Schritt ist aber vor allem dann besonders sinnvoll, wenn der Analyseaufwand sehr groß ist und es sich um ein komplexes System handelt. Durch das Verfeinern der System-Level Hazards kann die Erstellung der Control-Structure erleichtert werden [LT18].

Um die Sub-Hazards zu erarbeiten, sollte geklärt werden, welche Bestandteile des Systems man kontrollieren muss, um einen Hazard zu verhindern. Aus diesen lassen sich dann anschließend die Sub-Hazards aufstellen und dokumentieren [LT18].

Ein Beispiel hierfür ist die Verfeinerung des Hazards H-1.

Sub-Level Hazards: H-1: Das Fahrzeug kollidiert mit einem anderen Fahrzeug [L-1].

- **H-1.1: Bei der Bremsung muss der Mindestabstand, zu anderen Objekten, gewährleistet werden.**
- **H-1.2: Bei der Beschleunigung muss der Mindestabstand, zu anderen Objekten, gewährleistet werden.**
- **H-1.3: Bei der Lenkung muss der Mindestabstand, zu anderen Objekten, gewährleistet werden.**

2.3.2 Schritt 2 - Modellierung der Control-Structure

Definition - *"A **hierarchical control structure** is a system model that is composed of feedback control loops. An effective control structure will enforce constraints on the behavior of the overall system. "*[LT18]. In Schritt zwei wird nun eine hierarchische Control-Structure erstellt. Diese ist zusammengesetzt aus einzelnen Komponenten. Dabei gibt es vier Arten von Komponenten [LT18]:

- **Controller**: wird dafür genutzt, um mit Controll-Actions Prozesse zu kontrollieren. Damit werden dann die System-Level Constraints, die in Schritt 1 aufgestellt wurden, durchgesetzt. Dies geschieht, indem das Verhalten des zu kontrollierenden Prozesses mit dem Controller gesteuert wird. Hierfür werden das Process-Modell und der Control-Algorithmus genutzt.
  - **Process-Modell**: Das Processmodell repräsentiert die internen Überzeugungen die ein Controller hat und auf dessen Grundlage Entscheidungen getroffen werden können. Diese können sich ebenfalls im Laufe der Zeit verändern.
  - **Control-Algorithmus**: Auf Grundlage der internen Informationen entscheidet der Control-Algorithmus darüber, welche Aktionen an den zu Controlled Process weitergegeben werden.
- **Controlled Process**: Der Controlled Process empfängt die Control-Actions und führt die Aktionen aus. Anschließend wird, wenn vorhanden, ein Feedback zurückgesendet.

und zwei Arten von Interaktionen zwischen diesen:

- **Feedback**: Ein Feedback ist eine vom Controlled Process gegebene Information an den Controller. Dieses wird als nach oben gerichteter Pfeil Richtung Controller in der Control-Structure dargestellt.
- **Control-Action**: Die Control-Action ist der auszuführende Befehl, den der Controller an den zu Controlled Prozess sendet. In der Control-Structure stellt dies einen nach unten gerichteter Pfeil dar.

Eine Control-Structure ist in einzelne Ebenen gegliedert, in denen die darüberliegende Ebene die Kontrolle über die direkt darunter liegende hat. Wichtig ist, dass die vertikale Achse der Control-Structure genutzt wird. Das bedeutet, dass Abstraktionen genutzt werden sollen, um die Control-Structure erst einfach und rudimentär darzustellen und sie dann im Laufe der Zeit zu verfeinern. Die Control-Structure stellt kein ausführbares Modell dar und Control-Actions dürfen nicht so gesehen werden, dass die Aktion zwingend durchgesetzt werden muss [LT18].

Die Control-Structure kann zunächst sehr abstrakt modelliert werden. Danach kann diese iterativ erweitert und verfeinert werden. Um dies zu erreichen, kann man anhand der aufgestellten System-Level Hazards und System-Level Constraints das System identifizieren [LT18].

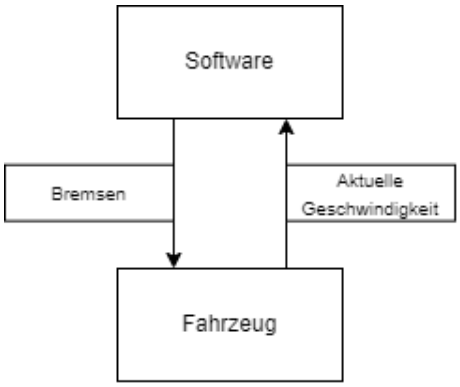

Abbildung 2.6: Beispiel für eine generische Control-Structure einer Bremsung eines Fahrzeugs mittels einer Software.

Responsibilities

Anschließend wird jeder Komponente, der zuvor modellierten Control-Structure, ihre Verantwortung zugeordnet. Diese Zuordnungen werden gemacht, um die Zuständigkeiten zu definieren. Somit sind diese eine Art Verfeinerung der Constraints. Dies bedeutet, dass dadurch analysiert wird, was alle abgebildeten Entitäten tun müssen, um die aufgestellten System-Level Constraints durchzusetzen [LT18].

Beispiel: **R-1: Die Software ist für die Bremsung des Fahrzeugs verantwortlich [SC-1].**

2.3.3 Schritt 3 - Identifizierung der Unsafe Control Actions

Definition: *“ An Unsafe Control Action (UCA) is a control action that, in a particular context and worst-case environment, will lead to a hazard. “ [LT18]*. Es gibt verschiedene Gründe, weshalb die Control-Actions zu einem Hazard führen können, wenn zusätzlich noch sehr schlechte Umwelteinflüsse dazu kommen.

So sind die folgenden vier Gründe maßgeblich daran beteiligt, dass Fehler auftreten [LT18]:

1. Ein Hazard kann auftreten, falls eine Control-Action zu lange andauert oder zu früh gestoppt wird. Dies kann nur bei kontinuierlichen Control-Actions auftreten und nicht bei diskreten.
2. Selbstverständlich kann ein Hazard auch dann auftreten, wenn die Control-Action gar nicht zur Verfügung gestellt wird.

3. Es gibt ebenfalls Control-Actions, die, wenn sie bereitgestellt werden, für die Sicherheit sorgen. Diese können aus folgenden Gründen nicht korrekt eingesetzt werden. Sie können zum einen zu früh sein, zum anderen zu spät. Außerdem können die notwendigen Befehle in der falschen Reihenfolge gegeben werden.
4. Eine Control-Action, die gegeben wird, obwohl sie gerade nicht gegeben werden sollte, kann zu einem Hazard führen.

→ UCA-1: Die Bremsung des Fahrzeugs wird zu spät eingeleitet [H-1]

UCA am Beispiel des autonom fahrendes Fahrzeugs für die jeweiligen Arten der zu Hazard führenden Control-Actions:

Control-Action: Automatische Bremsung des autonomen Fahrzeugs

- *Nicht gegeben:* keine Bremsung erfolgt, wenn eine Bremsung erforderlich ist.
- *Bereitgestellt:* Bremsung wird zur falschen Zeit zur Verfügung gestellt, wenn diese nicht erforderlich ist.
- *Zu früh, zu spät, falsche Reihenfolge:* Bremsung wird zu spät ausgeführt, wenn das Auto eine Kurve durchfährt.
- *Zu früh gestoppt, zu lange gegeben:* Bremsung wird zu früh beim Einfahren in die Kurve gestoppt.

Controller Constraints

Definition - *" A controller constraint specifies the controller behaviors that need to be satisfied to prevent UCAs " [LT18]*. Das bedeutet, dass sobald eine UCA im Schritt 3 identifiziert wurde, können diese in Controller Constraints übersetzt werden. Diese werden dann auf das Verhalten des Controllers übertragen [LT18].

2.3.4 Schritt 4 - Identifizierung der Loss Scenarios

Definition - *" A loss scenario describes the causal factors that can lead to the unsafe control actions and to hazards. " [LT18].*

Bei Erstellung der Loss Scenarios wird mit den UCAs begonnen. Anschließend arbeitet man sich rückwärts die Schritte hinauf, um eine Erklärung zu finden, ob die Control-Actions den Controller unterstützen oder nicht.

Dabei gibt es vier Kategorien, die ein Loss Scenarios darstellen können [LT18]:

1. Es gibt Fehler, die auf einen (physischen) Controller zurückzuführen sind, wie beispielsweise einen Stromausfall.
2. Control-Algorithmen können inadäquat sein
    - der Control-Algorithmus ist fehlerhaft

- die Implementierung ist fehlerhaft
- es können Veränderungen im System auftreten, die zu fehlerhaften Control-Algorithmen führen

3. der Kontrollinput ist unsicher, das bedeutet, dass der vom Controller erhaltene Input eine UCA ist, die deshalb bereits bei diesem betrachtet wird.
4. Process-Modelle können inadäquat sein
   - das an das Process-Modell gesendete Feedback ist inkorrekt
   - die Interpretation des Feedbacks ist inkorrekt oder wird vom Controller ignoriert
   - der Controller benötigt Feedback zu einem bestimmten Punkt, aber dieses bekommt er nur verzögert oder gar nicht
   - das Feedback, das der Controller benötigt, existiert nicht

Loss Scenario L-1: die aktuelle Geschwindigkeit wird in m/s versendet. Diese wird anschließend vom Controller aber in km/h interpretiert. Der Controller interpretiert das Feedback also auf inkorrekte Weise. Dies führt dazu, dass die UCA-1 auftritt.

2.4 Automatisiertes Fahren

Bei der Auswahl der Fallstudie sind, aufgrund der Anforderung: *„sicherheitskritische" Teile, die eine künstliche Intelligenz beinhalten,* meistens autonome beziehungsweise automatisierte Fahrzeuge involviert. Um die Level der Automatisierung zu definieren, sodass alle die gleichen Standards verwenden, wurden eine Taxonomy festgelegt: "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles" [SAE14]. Diese wird seit 2014 als Richtlinie genutzt und verwendet.

Dabei werden insgesamt sechs Level definiert. Zuerst Level 0, welches keine Automatisierung enthält. Level 1 ist ein Fahrassistenzsystem, das den Fahrer unterstützt. Der nächste Schritt ist das partiell automatisierte Fahren (Level 2). Danach wird die partielle Automatisierung durch einen hohen Automatisierungsgrad erweitert, was als Level 3 bezeichnet wird. Um zu Level 4 zu gelangen, ist eine vollständige Automatisierung in einem definierten Rahmen notwendig. Level 5 ist dann schließlich das autonome Fahren, welches das zu erreichende Ziel definiert [SAE14; VDA15]. Im Folgenden werden die einzelnen Level konkreter beschrieben und in Tabelle 2.1 zusammengefasst.

Bei Level 0 steuert die fahrende Person das Fahrzeug ohne Hilfe von Systemen. Das bedeutet, dass der Fahrer die Lenkung des Systems übernimmt, diese wird laterale Bewegung genannt. Außerdem muss ebenfalls die longitudinale Bewegung übernommen werden, die für die Beschleunigung und Bremsung zuständig ist. Zusammengefasst bedeutet das, dass die fahrende Person alle Teile des Fahrzeugs selbst ansteuern muss [SAE14; VDA15].

Das System regelt bei Level 1 entweder die laterale oder longitudinale Bewegung. Die andere Art muss die fahrende Person selbst steuern und im Fehlerfall muss die fahrende Person, beides übernehmen [SAE14; VDA15].

Bei Level 2 wird sowohl die laterale als auch die longitudinale Steuerung des Fahrzeugs vom System übernommen. Die fahrende Person muss dieses jedoch immer noch zu jedem Zeitpunkt überwachen. Im Fehlerfall übernimmt, wie in den vorherigen Stufen, die fahrende Person [SAE14; VDA15].

Bei Level 3 übernimmt das System, wie bereits in Level 2, die Steuerung des Fahrzeugs. Die fahrende Person muss das System nun nicht mehr durchgängig überwachen, aber immer noch in der Lage sein, jederzeit zu übernehmen. Das Fahrzeug ist nun aber in der Lage zu erkennen, wenn die Situation die Möglichkeiten des Systems überschreitet. Wenn dieser Fall eintritt, fordert es daraufhin die fahrende Person, mit genügend Zeit um zu reagieren, zur Übernahme des Fahrzeuges auf. [SAE14; VDA15].

Bei Level 4 wird die Steuerung aller Bewegungen vom System übernommen. Nun ist der Zeitpunkt erreicht, in dem das Fahrzeug innerhalb des definierten Anwendungsfalls alle Aufgaben selbstständig übernehmen kann. Dies inkludiert ebenfalls Fehlerfälle, die auftreten können, selbstständig zu erkennen und zu bewältigen [SAE14; VDA15].

In Level 5 ist das System nicht mehr an spezifische Anwendungsfälle gebunden. Von Beginn der Fahrt bis zum Ziel ist kein Fahrer oder keine Fahrerin mehr notwendig. Dies beinhaltet alle Geschwindigkeiten, Straßenarten und auch jegliche Umweltbedingungen [SAE14; VDA15].

Level	Level 0	Level 1	Level 2	Level 3	Level 4	Level 5
Beschreibung	keine Automation	Fahrassistenzsystem	partiell automatisiertes Fahren	hoch automatisiertes Fahren	voll automatisiertes Fahren	autonomes Fahren
System	-	entweder die Laterale oder Longitudinale Bewegung	lateral und longitudinale Bewegung	lateral und longitudinale Bewegung	lateral und longitudinale Bewegung	lateral und longitudinale Bewegung.
Fahrer	longitudinale und laterale Bewegung	die Bewegung die nicht vom Fahrzeug übernommen wird	-	-	-	-
Überwachung	-	die fahrende Person muss immer aufmerksam sein	die fahrende Person muss immer aufmerksam sein	der Fahrer muss übernehmen können, darf aber unaufmerksam sein	außerhalb der definierten Anwendungsfälle muss die fahrende Person übernehmen können	der Fahrer wird nun **nicht** mehr benötigt
Fehlerfall	-	Übernahme im Fehlerfall	Übernahme im Fehlerfall	Übernahme im Fehlerfall	in einem definierten Anwendungsfall kann System Fehler selbst beheben, sonst Übernahme	keine Übernahme notwendig System händelt Fehlerzustände selbst

Tabelle 2.1: Level des automatisierten bzw. autonomen Fahrens [SAE14; VDA15].

3 Fallstudie

In diesem Kapitel wird die Auswahl der Fallstudie dargestellt und ausgewählte Fallstudie betrachtet sowie die wichtigsten Teile erklärt. Dabei wird zunächst auf die Firma Comma AI im Allgemeinen eingegangen. Anschließend wird dann explizit deren Open-Source-Software Openpilot thematisiert, die für die später folgende STPA-Analyse als Grundlage dient.

3.1 Wahl der Fallstudie

Die Wahl der Fallstudie, die für diese Masterarbeit ausgewählt wurde, muss sicherheitskritische Teile beinhalten, bei denen KI eine Rolle spielt. Außerdem sollte die Software so weit wie möglich Open-Source sein, um eine möglichst umfassende Analyse erstellen zu können.

Infrage kommen hierfür sowohl Automatisierungssoftwares, die neuronale Netze beinhalten, die in der Zukunft autonomes Fahren ermöglichen sollen, als auch der Gesundheitssektor, in dem KI eine zunehmend wichtigere Rolle spielt. Dabei handelt es sich beispielsweise um selbstständige Pflegeroboter oder auch Operationsroboter. Die Recherche in diesem Bereich ergab, dass in diesem Bereich mit KI gearbeitet wird, wie bei Operationsrobotern Vicarious Surgical Robotic System [Vic22] und Theator's Surgical Intelligence Platform [The22]. Das Problem mit Software aus dem Gesundheitssektor, wie bei den beiden Operationsrobotern, ist, dass sie das Kriterium "Open-Source" nicht erfüllen. Deren Software wird nämlich nur in kleinen Teilen oder gar nicht veröffentlicht. Dies hängt auch damit zusammen, dass hier im Gegensatz zum Automobilsektor Daten von Patienten und Patientinnen verwendet werden, die aufgrund von Datenschutzrichtlinien besonders geschützt werden müssen [Ran21]. Dadurch wird die Entwicklung mit Open-Source schwieriger als im Automobilsektor. Die Hardware um Daten für selbstfahrende Fahrzeug zu beschaffen ist für jeden möglich, während sensible Daten nicht für jeden zugänglich sein dürfen.

Im Bereich *autonomes Fahren* gibt es zwei Bereiche, in denen Open-Source-Projekte vorhanden sind. Zum einen gibt es Forschungsprojekte zu diesem Kontext an den Universitäten, die die derzeitigen Möglichkeiten erforschen und testen. Zum anderen gibt es auch die Firma *Comma AI*, die den Ansatz verfolgt, ihre Hardware unter Verschluss zu halten, aber die darauf laufende Software mit einer MIT-Lizenz zur Verfügung zu stellt. Bei der Weiterentwicklung der Software wird die Github Community miteinbezogen [comc]. Das

Ziel des Unternehmens ist es, das Level 5 der Automatisierung (wie bereits in Abschnitt 2.4 beschrieben) zu erreichen. Dies bedeutet, dass sich das Fahrzeug vollständig automatisiert bewegen und jede Situation selbstständig lösen kann.

Die Software wird unter dem Namen Openpilot von Comma AI auf Github [comc] veröffentlicht. In deren Github ist ebenfalls anhand der Commits sichtbar, dass die Software seit 2015 regelmäßig weiterentwickelt wird. Außerdem wird über neue Versionen und Entwicklungen auf deren Blog [Com22] hingewiesen und über die Neuerungen informiert. Die verwendeten Programmiersprachen in Openpilot sind C++, C und Python. Außerdem will Comma AI eine möglichst große Breite an Personen, mit ihrer Software, erreichen und unterstützt deshalb derzeit bereits über 200 Fahrzeugmodelle [comc].

Bei der AutowareFoundation handelt es sich um ein 2015 gestartetes Forschungsprojekt, das seinen Source-Code unter einer Apache 2.0 Lizenz auf Github [Aut22] veröffentlicht. Die Plattform, die bei den recherchierten Forschungsprojekten eingesetzt wird, ist das Robot Operating System (ROS). Autoware hat, laut dem Zeitstrahl auf ihrer Webseite, bereits folgende Feature entwickelt [The21]:

- 2015–2019 wurden die Grundlagen für Autoware auf der ROS Plattform erarbeitet.
- 2020 wurde ein automatisches Parksystem via App Steuerung entwickelt (Autoware.Auto V1.0).
- 2021 wurde die Entwicklung für die Anlieferungen und den Transport von Frachten auf einem definierten Gelände entwickelt (Autoware V2.0). Das zweite entwickelte Feature ist das autonome Fahren in einer Rennstreckenumgebung in Zusammenarbeit mit *FIFENTH*.
- in 2022 soll nun eine universale Architektur vorgestellt werden und Autoware 3.0 entstehen. Dabei soll die Möglichkeit für autonome Busse auf öffentlichen Straßen entstehen, sowie Funktionalitäten für ein *Robo-Taxi* entworfen werden.

Das Ziel von Autoware ist es eine Automatisierung, auf Level 4 oder 5 zu erreichen. Die Motivation für Autoware ist es, die Entwicklung möglichst einfach und kostengünstig zu gestalten. Die genutzten Programmiersprachen sind hierbei C++, Python und für die Oberflächenbindung C#. Für die Ermöglichung einer schnellen Entwicklung wird der Gazebo Simulator verwendet. Außerdem sind laut Autoware Ai [Küt22] über 30 verschiedene Fahrzeuge mit der entwickelten Software kompatibel.

Das Projekt *Aslan* ist ebenfalls ein Projekt im Forschungskontext. Die Teilnehmer und Teilnehmerinnen des Projekts stammen hauptsächlich aus dem UK. Alsan - Autonomy wird wie Autoware in Github [POs] unter einer Apache 2.0 Lizenz veröffentlicht. Der initiale Commit fand am 01.07.2020 statt und der letzte Commit wurde am 24.09.2021 getätigt. Aslan wurde in C++ und Python geschrieben und das Automatisierungslevel, das erreicht werden soll, ist Level 3 bzw. 4. Denn die Funktion, die erreicht werden soll, ist automatisiertes Fahren in einem festgelegten Rahmen (bis zu 15 km/h). Alsan hängt eng mit Autoware AI zusammen und ist deshalb ebenfalls auf dem ROS-Framework aufgebaut [POs]. Der letzte Eintrag auf deren Webseite ist ein Blogeintrag, dass IPG Automotive dem

Projekt *Aslan* beigetreten ist [Pro20]. Es gibt auf der Website [Pro20] einen Plan, was über die nächsten zwei Jahre erreicht beziehungsweise an welchen Themen geforscht werden soll. Allerdings ist nicht vermerkt, wann diese zwei Jahre starten.

Das dritte Forschungsprojekt, das bei Github unter einer GPL-Lizenz veröffentlicht wurde, ist *OpenPodcar* [CWCF22]. Der initiale Commit dieses Projekts fand am 02.10.2018 statt und der letzte am 09.Mai 2022 und verwendet die Programmiersprachen C++, C, C#. Das Ziel der Automatisierung ist hierbei, wie bei *Aslan*, Level 3 beziehungsweise Level 4. Außerdem wird hierbei ebenfalls ROS als Framework verwendet. Ähnlich wie bei Comma AI wird hier auch die Hardware mitentwickelt. Das Ziel ist, eine möglichst kostengünstige Fahrzeugerweiterung zu bauen. Die Hardware wird unter der CERN-OHL-W Lizenz veröffentlicht.

Eine nochmalige Übersicht über alle vier Projekte wird in Tabelle 3.1 dargestellt. Die Entscheidung für die Fallstudie fiel auf *Comma AI*, da diese die meisten Fahrzeuge unterstützt. Obendrein hat Comma AI eine sehr aktive Community, sodass die Entwicklung in Github sehr aktiv ist. Ebenfalls informiert Comma AI sehr gut über ihre Software in ihrem Blog und sie sind transparent über alle Teile ihrer Software, inklusive ihrer neuronalen Netze.

3.2 Comma AI

Comma AI startete 2015 als Start-up [VB15]. Sie stellen sowohl die Software als auch die Hardware für automatisierte Fahrsysteme bereit. Besonders an dieser Firma ist dabei, dass sie ihren gesamten Code auf Open-Source gestellt haben und so eine Community besitzen, die diesen gemeinsam mit der Firma weiterentwickeln kann. Mit dem Titel *“Our Road to Self Driving Victory“* veröffentlichte Comma AI im Juni 2017 seinen groben Plan, wie man an das Ziel gelangt, den besten Autopiloten zu bauen, um die Ziele: *“Win self driving cars“* und *“Make driving chill“* zu erreichen [com17].

Auf der Website von Comma AI wird hierfür die Hardware verkauft, mit der es möglich ist den Autopiloten (derzeitiger Stand der Automatisierung ist Level 2) laufen zu lassen. Für die Verbindung der Hardware mit der Software wird ein Panda genutzt. Comma One hatte am Anfang noch einige Startprobleme, da die US-Sicherheitsbehörde noch nicht vom Sicherheitsmodell überzeugt war [Eth16]. Daraufhin hat Comma AI offiziell verkündet, dass die Veröffentlichung von Comma One auf dem US-Markt hiermit abgesagt wird [com16]. Danach passte Comma AI ihr Verkaufskonzept an. Seit dem wird nur noch die Hardware verkauft und die Software wurde Open-Source gestellt, sodass sie frei verfügbar ist. Dadurch stellt der Comma One laut Comma AI kein Konsumgut mehr dar und fällt deshalb unter weniger strenge Regularien [Big18].

mögliche Fallstudien	OpenPod	Aslan	Autoware	Comma AI
Gründung	Initialer Commit 2.10.2018	Initialer Commit 01.07.2020	2015	2015
Ziel	Forschung	Forschung	Forschung	Kommerziell
Programmiersprachen	C++, C, C#, Matlab	C++, Python	C++, C#, Python	C++, C, Python
unterstützte Fahrzeugmodelle	-	-	30+	200+
Komponenten	Software + Hardware	Software	Software	Software + Hardware
Ziel des Automatisierungsgrades	Level3/4	Level 3/4	Level 4 oder 5	Level 5
Funktionen	Geschwindigkeit (max. 15 km/h) Lenkung Gazebo Simulation Objekterkennung Tracking	Geschwindigkeit (max. 15 km/h) Objekterkennung Gazebo Simulation Pathplanning mit dem A-Planner Routenplanung mit Wegpunkten	Geschwindigkeit Objekterkennung Lokalisierung Gazebo Simulation	Geschwindigkeit Lenkung Objekterkennung automatische Fahrspurerkennung
Plattform	ROS, geringe Kosten	ROS, Einfachheit	ROS	Eigene Plattform
Lizenz	Software: GPL Lizenz Hardware: CERN-OHL-W Lizenz	Software: Apache 2.0 Lizenz	Software: Apache 2.0 Lizenz	Software: MIT-Lizenz Hardware: nicht Open-Source

Tabelle 3.1: Mögliche Fallstudien, die verwendet werden können, sowie deren Inhalte ([Aut22; comc; CWCF22; POs; Pro20; The21]).

Danach wurden der Comma Two und der Comma Three entwickelt. Diese beiden stellen die zwei Geräte dar, die derzeit hauptsächlich verwendet werden und mit denen Openpilot genutzt wird. In diesen ist der benötigte Panda direkt verbaut. Comma Three löste ab dem 31. Juli 2021 den Comma Two ab. Seitdem wird über Comma AI nur noch der Comma Three verkauft [Com21; com21a]. Comma Two wurde bis zum Release 0.8.13.1 [Clo22](27.04.2022) noch weiter unterstützt [Ade22]. Um die Veränderungen von Comma Two zu Comma Three möglichst transparent aufzuzeigen, werden sie in Tabelle 3.2 zusammengefasst. Die größten Veränderungen sind dabei, dass das Kamerasystem um eine weitere Außenkamera erweitert wurde. Zudem wurde die Fähigkeit für Nachtfahrten und die Wahrnehmung auf weite Distanzen verbessert. Außerdem wurde, damit größere Mengen an Daten gespeichert werden können, eine externe SSD eingebaut [com21a; TEG22]. Ein weiteres ganz neues Feature ist die Navigation, die beim Comma Two noch nicht möglich war [Com21]. Es wurden außerdem um ein OBD-C Port und eine USB 3.1 Schnittstelle ergänzt [com21a; TEG22].

(a) Vorderansicht eines Comma Threes [com21a].

(b) Rückansicht eines Comma Threes [com21a].

Abbildung 3.1: Ein Comma Three Gerät [com21a].

Comma AI besitzt ein Git-Repository, das aus 102 Sub-Repositorys besteht. Dabei ist das wichtigste Repository *openpilot*, es enthält die für Openpilot relevanten und notwendigen Funktionen. Diese werden ausführlich in Abschnitt 3.3 beschrieben. Es gibt jedoch vier weitere Repositorys, die erwähnenswert sind und deshalb hier ebenfalls erläutert werden:

comma2k19: Hierbei handelt es sich ein großes Datenset. Dieses besteht aus gesammelten Daten von einem Highway Abschnitt, der sich von California's San José bis nach San Francisco erstreckt. Der insgesamt 33-stündige Datensatz enthält dabei 2019 einminütige Fahrtabschnitte, die sich jeweils auf einen 20 km langen Abschnitt des genannten Highways beziehen [SSHB18].

Mithilfe dieses Datensets wurde Laika entwickelt, bei der es sich um eine Global Navigation Satellite System (GNSS) Verarbeitungsbibliothek für Python handelt [SSHB18]. Dabei werden comma2k19 und Laika in wissenschaftlichen Arbeiten verarbeitet, wie beispielsweise in der Arbeit von Rahman et al. [RICK22].

Comma Two	Comma Three
zwei Kameras (eine Außen-, eine Innenansicht)	drei 360° Kameras (zwei Außen-, eine Innenansicht)
Qualcomm Snapdragon 821	Qualcomm Snapdragon 845
interner Speicher	Samsung 980 NVMe SSD (250GB oder 1TB)
1920 × 1080 LCD display	OLED display 2160x1080
Wi-Fi, LTE	Wi-Fi, LTE
sehr präzises GPS	sehr präzises GPS
	IR LEDs um Nachtsicht im Innenraum zu gewährleisten
	''Narrow cam" um weit entfernte Objekte zu sehen
	OBD-C port (USB-C und CAN)
	USB 3.1 Gen 2

Tabelle 3.2: Features von Comma Two [TEG22] und Comma Three [Com21; com21a] im Vergleich.

comma10k: Diese Bibliothek ist darauf ausgelegt, das neuronale Netzwerk von Openpilot, mithilfe von weiteren Trainingsbildern, noch weiter zu verbessern. Dafür werden die bereitgestellte Bilder mittels Farben so annotiert, dass zwischen Straßen, den Fahrbahnlinien, sich bewegenden Objekten, dem eigenen Auto und der Umgebung unterschieden werden kann. Diese Farben werden von den Personen eingezeichnet, die sich beteiligen möchten, um zum Datenset für das Training des neuronalen Netzes von Openpilot beizutragen. Selbstverständlich werden die Ergebnisse kritisch begutachtet, bevor diese genutzt werden. Durch die Arbeit von vielen Freiwilligen wurde das erste Set mit 10.000 Bildern bereits vollständig editiert und ein zweites hinzugefügt. Außerdem wurde bereits begonnen, Bilder für den Innenraum zu labeln. Diese Trainingsdaten werden dann genutzt, um das neuronale Netz weiter zu trainieren [coma].

calib_challenge: Die Kalibrierungs-Challenge ist ein Feature, dass die Idee hinter dem Open-Source-Projekt verstärkt. Denn durch die Kalibrierungs-Challenge können Personen herausgefiltert werden, die als zukünftige Entwickler und Entwicklerinnen bei *Comma AI* infrage kommen. Denn sie haben sich, wenn sie die Challenge bestehen, intensiv mit dem Openpilot-Code auseinandergesetzt und sehr gute Ergebnisse erzielt. In dem Projekt liegen zehn Videos, die je 1min lang sind, vor. Fünf davon wurden bereits mit einem 2D Array gelabelt, der die Richtung des Fahrweges für jede Sekunde ausgibt und einige weitere Zusatzinformationen enthält. Die Aufgabe ist es nun, die anderen fünf, nicht gelabelten

Videos, ebenfalls mit solchen Arrays zu versehen. Außerdem ist ein Script gegeben, dass die Fehlerrate bestimmt, um der Person, die eine Lösung programmiert hat, Aufschluss über das erzieltes Ergebnis zu geben [comb].

Überblick Aufbau

Der Aufbau der Kommunikation wird in Abbildung 3.2 gezeigt. Dabei wird zunächst der Comma Two/Three mit dem Fahrzeug verbunden. Diese beiden kommunizieren über CAN-Messages. Danach kommuniziert die Hardware (Comma Two/Three) mit der Software (Openpilot). Hierbei wird vom Comma der CarState, der spezifische Informationen des Fahrzeugs beinhaltet, an Openpilot übergeben. Außerdem werden Sensordaten übergeben. Diese werden dann von Openpilot interpretiert und verwendet. Anschließend werden die verarbeiteten Sensordaten an die Modelle weitergegeben und genutzt. Außerdem werden die Daten, die geloggt werden, an eine weitere Instanz geschickt, wo sie verwendet werden, um daraus Trainingsdaten zu generieren und auch Logs zu analysieren, sowie für die Infrastruktur. Aus den hier gewonnenen Erkenntnissen können die von Openpilot verwendeten Modelle verbessert werden. Aus den genutzten Modellen wird der Fahrweg abgeleitet und auch weitere (Meta) Vorhersagen können vorgeschlagen werden. Openpilot gibt dann die Signale, die durch die Verarbeitung entstanden sind, an die Hardware weiter. Diese sind, dass das Fahrzeug bremst, beschleunigt oder lenkt. Dabei werden die Kommunikation des Fahrzeugs, Comma Three/Two und Openpilot sowie die Modelle in Echtzeit verarbeitet, während das Training, die Infrastruktur und die Analyse offline funktionieren [com21b].

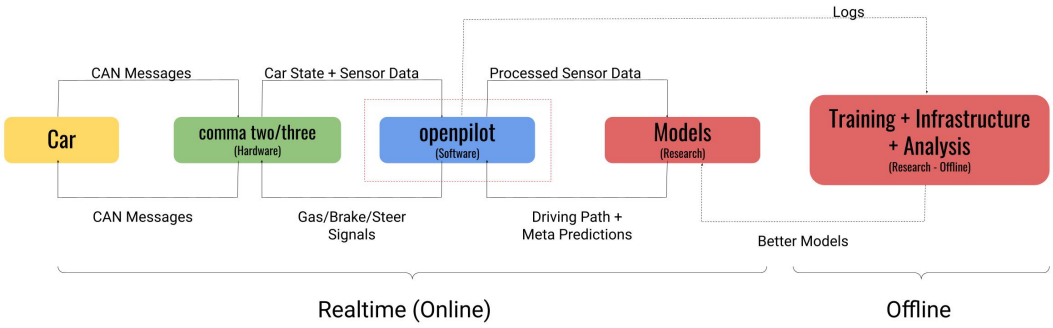

Abbildung 3.2: Übersicht über die Kommunikation des Comma AI Geräts und die Verarbeitungsteile [com21b].

3.3 Openpilot

Openpilot wurde als initialer Release am 30.11.2016 in Github veröffentlicht [comc]. Das Projekt wurde hauptsächlich in den Programmiersprachen Python, C++ und C geschrieben. Die Verteilung der Programmiersprachen wird in Abbildung 3.3 abgebildet.

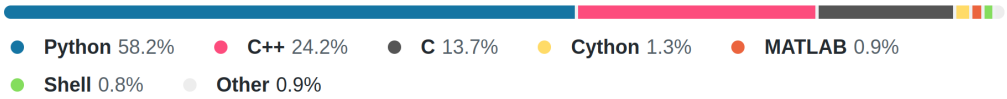

Abbildung 3.3: Übersicht der verwendeten Programmiersprachen in Openpilot [comc].

Die essenziellen Features von Openpilot sind [comc; com21a]:

- Automatische Fahrlinienzentrierung (automatic line centering)
- Adaptive Geschwindigkeitsregelung (adaptive cruise control)
- Überwachung des Fahrers oder der Fahrerin (driver monitoring)
- Spurwechselassistent (assisted lane change)
- zusätzliche Softwareupdates

Im folgenden Abschnitt werden die, für die STPA-Analyse wichtigen Packages und Funktionen erklärt, damit diese in Kapitel 4 bekannt sind.

3.3.1 Übersicht über die einzelnen Packages und ihre Funktionen

Openpilot verwendet zum Erreichen der Funktionalität verschiedene Submodule. Diese werden genutzt, um Funktionalitäten der Openpilotsoftware zu ergänzen und zu ermöglichen. Hierbei handelt sich um die Module: *Cereal, Laika, OpendBC, Panda und Rednose*. Kurz zusammengefasst wird deren Nutzen für Openpilot in Abbildung 3.4

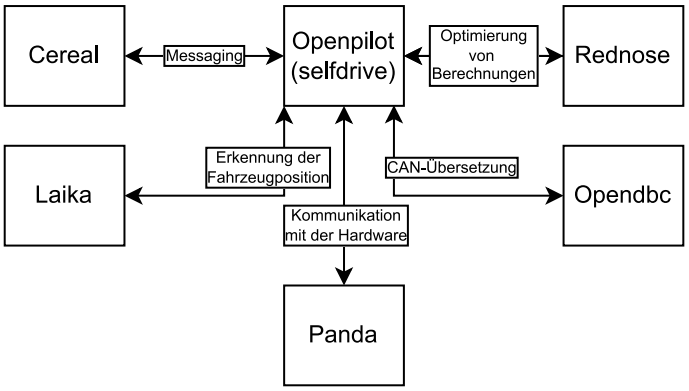

Abbildung 3.4: Übersicht über die Funktionen, die die Module *Cereal, Laika, OpendBC, Panda, Rednose* zu Openpilot beitragen (zusammengestellt aus den Informationen von [com21b; Fon21].

- **Cereal** - wird dafür genutzt, dass Daten zwischen dem Fahrzeug und der Software ausgetauscht werden können. Cereal wird beim Messaging im Robotik-Bereich genutzt. Das genutzte Protokoll IPC (inter-process communication protocol) hat einen Publisher, dem beliebig viele Subscriber zuhören können. Geschrieben ist Cereal in C++, sowie in Cython, um das Submodule ebenfalls mit Python verwenden zu können. Der Hauptteil des Submodules ist in Capt'n Proto geschrieben, ein Datenformat, das zum schnellen Austausch bei Server Client Programmen verwendet wird [com21b; Fon21].
- **Laika** - ist eine in Python geschriebene Bibliothek, die in Openpilot als Submodul dient und für die GNSS Verarbeitung verwendet wird. Laika konzentriert sich darauf, gut lesbar sein. Außerdem liegt viel Wert auf der Nutzbarkeit und es muss möglichst leicht sein, andere Optimizer zu integrieren, wenn sie benötigt werden [comc]. GNSS ist ein essenzieller Teil von Openpilot, da er eine wichtige Schnittstelle zur Erkennung der Position des Fahrzeugs bildet. Dabei können verschiedene Quellen genutzt werden, um auch die Geschwindigkeit abzuschätzen. Da hierbei Satelliten zum Einsatz kommen, sollte jederzeit eine Position gegeben sein [Fon21].
- **Opendbc** - ist ein Submodul, das zur CAN-Übersetzung genutzt wird. Bei CAN handelt es sich um ein einfaches Protokoll, das als einheitliche Schnittstelle verwendet wird, um mit Fahrzeugen zu kommunizieren. Hiermit kann also eine Nachricht an die Steuerungseinheit des Fahrzeugs gesendet werden. Genutzt werden hierzu im Regelfall entweder Panda oder NEO Interfaces. Die vom Fahrzeug vorhandenen Informationen werden durch den DBC-File Standard in ein für Menschen lesbares Format transformiert. Bei den einzelnen Fahrzeugen werden hierbei .dbc Files erzeugt, in denen die Informationen abgelegt werden. Dadurch ist es möglich, alle relevanten Informationen zu extrahieren und dem Fahrzeug mitzugeben. Fahrzeuge können ebenfalls mehr als einen CAN-Bus besitzen, dann werden hier mehrere DBC-Files erzeugt [comc; com21b; Fon21].

- **Panda** Das Submodul *Panda* besteht aus mehreren Teilen: Dem *board* auf welchem der Code läuft, der auf dem STM32 ausgeführt wird. Der Ordner *drivers* ist nur notwendig, wenn mit C++ entwickelt wird, da dieser in Python nicht benötigt wird. In *Python* befinden sich die Userspace Bibliotheken, die beim Interfacing benötigt werden, sowie *Tests* und weitere Hilfsprogramme [comc; com21b].
- **Rednose** - ist das letzte Submodul. Es ist ein mächtiges Tool, das zu Optimierungszwecken entwickelt wurde. Genutzt wird dieses für "visual odometry, sensor fusion localization, SLAM" [comc]. Wichtig ist es, dass die Optimierung sowohl online als auch offline funktionsfähig ist. Außerdem sollte das Optimierungsmodul selbstverständlich möglichst akkurate Ergebnisse liefern und soweit es im Bereich des möglichen ist, nicht zu rechenintensiv sein. Es können hiermit außerdem vergleichsweise einfach Pythonfilter designt werden. Genutzt werden in diesem Submodul ebenfalls *Kaiman Filter* [comc].

Die größten und wichtigsten Funktionen für das partiell automatisierte Fahren werden in dem *selfdrive* Modul in Github gebündelt. Deren Zusammenhänge in der STPA-Analyse behandelt werden.

athenad: Athenad wird genutzt um eine Verbindung, mittels eines Websockets, zu `https://connect.comma.ai/` herzustellen [com21b]. Von dort aus ist es möglich, auf die Daten des eigenen Commas zuzugreifen und beispielsweise aufgezeichnete Fahrten anzusehen. Dabei ist sowohl eine Karte der zurückgelegten Strecke, sowie ein Video der Fahrt verfügbar. Des Weiteren kann von dort auf Comma Prime Features zugegriffen werden. Bei Comma Prime handelt es sich um einen zusätzlich zu bezahlenden Service [Com]. Athenad bearbeitet hierfür alle Anfragen zum Upload von Daten, die per REST an die `https://athena.comma.ai` Schnittstelle gesendet werden. Die Identifikation der Geräte erfolgt dabei über die Dongle-ID des Comma Geräts. Zusätzlich zum Hochladen von Daten kann durch die API aber auch beispielsweise die Batteriespannung abgerufen werden [com21b].

boardd: Boardd ist für die Kommunikation zwischen dem Fahrzeug und dem gekauften Comma Two/Three verantwortlich. Dabei wird Panda als Übertragungsmedium genutzt. Dieser erhält seine Informationen mittels des Busses vom Fahrzeug und kommuniziert mit dem Comma Gerät via einer USB-Schnittstelle. Dabei werden hier die Daten entsprechend geparst, damit die jeweilige Seite, mit den erhaltenen Informationen eine Aktion durchführen kann. Für die Kommunikation mit dem Comma wird hierbei ein Socket *sendcan* genutzt, von dem sich der Comma die neusten Informationen abruft. Hierfür wird das bereit beschriebene Cereal Paket als Messaging System genutzt [Fon21]. Openpilot kann hierbei sowohl mit dem in den Comma Two/Three Geräten festverbauten Pandas sprechen [Kik21], als auch mit einem manuell verbundenen Panda Black oder Red. Ältere Panda Versionen, wie der Panda Grey, werden nicht mehr unterstützt [Fon21].

cars: Cars beinhaltet ein generisches Interface, das genutzt wird, um Nachrichten des CAN Busses zu empfangen und auch an diesen zu senden. Dabei sind für jede Fahrzeugmarke, sowie das Modell, eine eigene DBC-Datei hinterlegt. Relevant sind hierbei vor allem

der *carcontroller* und der *carstate*, die unter anderem von *controlsd* genutzt werden, um CAN Nachrichten, die versendet werden sollen, korrekt zu bauen (CarController.py) und Nachrichten zu parsen, um die Daten in Openpilot zu nutzen (Carstate.py) [comb].

calibrationd: ist ein notwendiger Service, um die eingehenden Bilder so zu konfigurieren, damit sie anschließend vom neuronalen Netz, das diese als Input verwendet, genutzt werden können. Um dies zu erreichen, wird ein "calibrated frame" angewandt. Der Grund für den Einsatz ist hierbei vor allem, dass verschiedene Fahrzeugmodelle angeboten werden, auf denen Openpilot genutzt werden kann. Aufgenommene Bilder unterscheiden sich somit, je nach Fahrzeugmodell. Da das neuronale Netz bei ähnlichen Inputs aber bessere Ergebnisse liefert, werden diese vorher in einem "calibrated frame" ähnlicher gemacht [com21b; Fon21].

controlsd: Controlsd ist die zentrale Komponente, die die Steuerung über das Fahrzeug besitzt. So werden die spezifischen Fahrzeuginformationen über Controlsd an die Komponenten verteilt, die diese benötigen. Nachdem diese ihre jeweiligen Verarbeitungsschritte vorgenommen haben, werden alle Informationen wieder in Controlsd gebündelt und dann in Richtung Fahrzeug zurückgeschickt [com21b].

radard: Bei Radard werden zunächst aus dem *Car*-Ordner, dynamisch zur Laufzeit, das passende Radar-Interface eingelesen, da die Fahrzeuge, je nach Marke und Modell, andere Radare besitzen [Fon21]. Diese werden dann zu einem kanonischen Format konvertiert. Danach wird der *radarstate*, beispielsweise im Plannerd weiterverwendet [com21b].

plannerd: der Planning Teil ist in zwei Teile aufgeteilt. Dabei handelt es sich zum einen um das *laterale Planning*, das die Lenkung steuert und auf der anderen Seite um das *longitudinale Planning*, welches das Bremsen sowie das Gas geben steuert. Für die Umsetzung wird hierzu ein Model Predictive Controller (MPC) Regler verwendet, der für die Optimierung sowie die Planung genutzt wird [com21b]. Durch diesen MPC Regler wird es aufgrund der verfügbaren Rechenleistung möglich, Berechnungen durchzuführen, die eine Steuerung ermöglichen, die im besten Fall optimal ist. Diese Berechnung geschieht in Echtzeit und ist für Systeme geeignet, die sich dynamisch entwickeln und wird genutzt, da es sich hierbei um eine der derzeit erfolgreichsten Methoden zur Kontrolle von Reglern handelt [Ins].

Bei plannerd werden zwei neuronale Netze verwendet, das *driving neuronal network* und die *neuronal network path prediction*. Das *driving neuronal network* sagt dabei voraus, wo das Fahrzeug sich befinden soll, während der lateral Planner dafür eingesetzt wird herauszufinden, wie das Fahrzeug dorthin gelangen kann. Mithilfe der *neuronal network path prediction* und dem bereits erklärten MPC Reglers kann nun der lateral Planner die Kurven abschätzen und weiß, wie stark er in den nächsten Sekunden die Lenkung verändern muss [com21b]. Für das Gas geben und Bremsen muss ebenfalls der longitudinal Planner mit Informationen ausgestattet werden. Dies geschieht über eine Kombination aus den Daten vom neuronalen Netz, sowie dem verbauten Radar. Dabei wird das vorausfahrende Fahrzeug als Richtwert betrachtet. Denn auf dieser Grundlage werden die Daten fusioniert und eine

Schätzung erstellt. Diese wird vom MPC Regler interpretiert und für die Berechnung eines Profils für die optimale Beschleunigung genutzt, die in den darauffolgenden Sekunden verwendet wird [com21b].

locationd: Wo sich das Fahrzeug auf der Welt befindet, wird durch Locationd bestimmt. Dabei werden diverse Quellen genutzt, um dies möglichst präzise zu bestimmen. Die Sensordaten werden hierbei durch einen Kalmanfilter "gesäubert", um möglichst genaue Ergebnisse zu erhalten. Diese Sensordaten umfassen: Kameras, GPS Daten und Inertialsensoren [Fon21]. Dabei werden sowohl die Geschwindigkeit des Fahrzeugs, als auch die Geschwindigkeitswinkel, Orientierung, Position und die Beschleunigung des Fahrzeugs angegeben. So weiß Openpilot genau, mit welcher Geschwindigkeit das Auto fährt und ob es sich dabei auf einer geraden Strecke oder an einem Berg befindet. Des Weiteren erkennt Openpilot so, ob das Fahrzeug gerade aus oder derzeit eine Kurve fährt [com21b].

loggerd: In loggerd werden alle Nachrichten geloggt. Diese werden dazu genutzt, mögliche Fehler zu entdecken und so den korrekten Ablauf sicherzustellen. Dazu gehören neben loggerd auch *locat* und *proclogd*. Dabei werden drei Arten von Daten geloggt. Als Erstes werden alle Nachrichten geloggt, die zwischen den Openpilot Prozessen ausgetauscht werden. Des Weiteren werden die Kameradaten geloggt. Diese werden encodiert und dann in Dateien gespeichert. Außerdem werden Teile der Daten in geringeren Qualitäten gespeichert, um diese beispielsweise zum Debuggen zu verwenden [comc; com21b].

manager: Der Manager ist dafür zuständig, alle Prozesse zu starten und zu stoppen. Mit *allen Prozessen* sind alle Prozesse gemeint, die zu den Funktionalitäten im selfdrive Module beitragen [Fon21].

modeld: Modeld verarbeitet den von VisionIPC bereit gestellten ImageStream der als Input für das "main driving neuronal network" dient. Dann werden die Inputdaten, sowie weitere Informationen, wie z. B. ein Fahrspurwechsel genutzt und das neuronale Netzwerk sagt auf dieser Grundlage voraus, wo das Fahrzeug fahren soll und ebenfalls wie das Fahrzeug sich bewegen muss. Das Model des neuronalen Netzwerks, das hierbei läuft, hat den Namen *Supercombo* und ist im .onxx Format im Github Repository von Openpilot frei verfügbar. Als Output gibt es zusätzlich zum Fahrweg des Fahrzeugs auch die Fahrspurlinien, sowie die Kanten der Fahrbahn und auch die vorausfahrenden Fahrzeuge aus. Diese Metadaten werden dann nicht nur weiterverarbeitet, sondern auch in der Benutzeroberfläche des Comma Two/Three ausgegeben, sodass die fahrende Person, die erkannten Objekte ebenfalls sehen kann [com21b].

dmonitoringmodeld: Bei diesem Feature handelt es sich um eine Funktion, welche essenziell für die Sicherheit von Openpilot ist. Denn hier wird überprüft, ob der Fahrer oder die Fahrerin, wenn nötig, jederzeit in der Lage ist, die Fahrfunktionen des Fahrzeugs zu übernehmen, wenn er/sie dazu aufgefordert wird, oder eine Gefahrensituation eintritt. Die Entscheidung basiert hierbei auf dem dmonitoring_model.onxx. Dieses Netz erhält als Input einen Imagestream der von einer in den Fahrerraum gerichteten Kamera stammt. Auf

Grundlage dessen wird dann durch das neuronale Netz vorhergesagt, wie die Kopfdrehung gerade ist und ob die fahrende Person gerade wach ist oder die Augen geschlossen hat. Dafür wird ebenfalls beachtet, ob die Person eine Sonnenbrille trägt [com21b].

dmonitoringd: Dmonitoringd enthält die Logikentscheidung, ob die fahrende Person in der Lage ist zu übernehmen. Sollte die Entscheidung sein, dass der Fahrer oder die Fahrerin dazu nicht in der Lage ist, wird er/sie alarmiert. Die Entscheidung wird hauptsächlich basierend auf dem Output des dmonitoring_models getroffen. Jedoch werden zusätzliche Informationen, die das Fahrmodell über die Szene besitzt, miteinbezogen. Wenn die fahrende Person über einen längeren Zeitraum abgelenkt war, wird sie anschließend zunächst davon blockiert, Openpilot zu verwenden [comc; com21b].

camerad: kümmert sich um, die im Comma Three verbauten, Kameras. Dabei wird hier vor allem auch die Belichtung angepasst [com21b].

sensord: Liest die vorhandenen Sensoren ein, die noch nicht direkt von andern Services eingelesen wurden. Diese werden von *sensord* ebenfalls konfiguriert. Beispiele für diese Art von Sensoren sind: Lichtsensoren und Gyroskope [com21b].

thermald: Der Service *thermald*, ist für die Überwachung des Zustandes des Comma Two/Three zuständig, auf dem Openpilot ausgeführt wird. So wird hier die Speichernutzung des Comma Geräts überwacht. Des Weiteren werden hier beispielsweise auch der Status der Stromversorgung und die Temperatur des Systems überwacht [com21b].

ui: Der Service *ui* behandelt alles, was für den Nutzer sichtbar ist. Dabei gibt es eine Ansicht, wenn das Fahrzeug ausgeschaltet ist, mit Anleitungen, die neuen Benutzern den Einstieg erleichtern sollen. Außerdem wird darauf der aktuelle Status des Systems, sowie Einstellungen angezeigt, die offline getroffen werden können. Wenn das Fahrzeug angeschaltet ist, wird über den Comma Two/Three Bildschirm die Ansicht der aktuellen Frontkamera gezeigt. Über diese wird eine Ansicht gelegt, welche die Straßenlinien, den aktuellen Fahrweg als Linie, sowie die vorausfahrenden Fahrzeuge bzw. die Fahrzeuge, an denen sich Openpilot orientiert, anzeigt [com21b].

Sicherheitsmodell von Openpilot

Comma AI befindet sich aktuell in Level 2 des autonomen Fahrens. Dieses beinhaltet wie in Tabelle 2.1 zu sehen ist, dass der Fahrer noch eingreifen kann und auch muss, falls nötig. Dabei werden von Openpilot zwei wichtige Sicherheitsvorkehrungen hervorgehoben, die sie aus FMEA und Hazard Risk Analysis abgeleitet haben. Diese wurden hierbei aber auf einem sehr hohen Level angesetzt und beinhalten [comc]:

1. Die fahrende Person muss zu jederzeit in der Lage sein, die Kontrolle über das Fahrzeug übernehmen zu können. Dies muss sofort zum geforderten Zeitpunkt geschehen und darf nicht erst nach einer Verzögerung erfolgen. Dabei ist mit der Kontrolle vorrangig gemeint, dass der Fahrer die Bremse betätigen oder den Stoppknopf drücken kann (je nach Modell).

2. Auch beim Lenken muss die Sicherheit gewährleistet sein. So muss die Kurvendurchführung in so einer Geschwindigkeit und Schnelligkeit erfolgen, dass der Fahrer oder die Fahrerin die Möglichkeit hat, rechtzeitig einzugreifen. Das bedeutet, dass die verwendeten Aktuatoren nur in gesetzten, sinnvollen Bereichen arbeiten, um dies gewährleisten zu können.

Des Weiteren wird auf das Panda Safety Modell verwiesen. Dort werden Sicherheitsimplementierungen für spezifische Fahrzeughersteller und -typen verwaltet [comc].

4 STPA-Analyse

Nun wird STPA auf das System angewandt. Das System wird hierbei so definiert, dass es die Openpilot-Software umfasst, sowie die Kommunikation über einen Panda (diese befinden sich beide im Comma Two/Three Gerät) und die Übertragung an das Fahrzeug. Die verwendeten Informationen dieses Kapitels beziehen sich auf die Angaben, die Comma AI in ihrem Blog [com21b] und in ihrem Openpilot Repository [comc] machen, sowie der Arbeit von F. Fontana [Fon21].

4.1 Schritt 1 - Grundlagen der Analyse

Zunächst werden die Grundlagen für die Analyse festgelegt. Diese umfassen die Losses, System-Level Hazards, sowie deren dazugehörige System-Level Safety Constraints.

4.1.1 Losses

Die Losses sind in dieser STPA-Analyse die Werte, die Comma AI schützen möchte und die im Umgang mit Openpilot von Bedeutung sind.

L-1: Tod oder Verletzungen von Menschen, die sich im Fahrzeug befinden.

L-2: Totalschaden oder Beschädigungen am Fahrzeug.

L-3: Sachschäden von Objekten in der Umgebung.

L-4: Tod oder Verletzung von anderen Verkehrsteilnehmern (bspw. Fußgänger, Motorrad-, Radfahrer).

4.1.2 System-Level Hazard

Hier werden die System-Level Hazards beschrieben, die bei gewissen Zuständen von Openpilot in einem worst-case Szenario zu einem Loss führen können.

H-1: Der Verlust der Fahrzeugkontrolle [**L-1**, **L-2**, **L-3**, **L-4**].

H-2: Das Fahrzeug kommt zu nah an andere Objekte [**L-1**, **L-2**, **L-3**, **L-4**].

H-3: Die fahrende Person ist abgelenkt/schläft und kann nicht eingreifen, wenn notwendig [**L-1**, **L-2**, **L-3**, **L-4**].

H-4: Das Fahrzeug bzw. Openpilot hat keine Verbindung zum GPS Signal und kennt deshalb seine aktuelle Position und somit die Straßenführung nicht [**L-1**, **L-2**, **L-3**, **L-4**].

H-5: Ein ungeplanter Spurwechsel beziehungsweise das Verlassen der Spur [**L-1**, **L-2**, **L-3**, **L-4**].

H-6: Das ungeplante Nehmen einer Kurve [**L-1**, **L-2**, **L-3**, **L-4**].

H-7: Das ungeplante Beschleunigen [**L-1**, **L-2**, **L-3**, **L-4**].

H-8: Das ungeplante Bremsen [**L-1**, **L-2**, **L-3**, **L-4**].

4.1.3 Safety-Constraints

Nun werden die System-Level Safety Constraints aufgezeigt, die das Systemverhalten von Openpilot definieren sollen, damit die System-Level Hazards verhindert werden können.

SC-1: Das Fahrzeug muss zu jedem Zeitpunkt kontrollierbar sein [**H1**].

SC-2: Das Fahrzeug muss einen Mindestabstand zu anderen Objekten gewährleisten [**H2**].

SC-3: Der Fahrer oder die Fahrerin muss zu jederzeit in der Lage sein, die Kontrolle über das Fahrzeug zu übernehmen [**H3**].

SC-4: Die Straßenführung muss zu jeder Zeit bekannt sein, wenn nicht, sollte der unsichere Zustand gemeldet werden [**H4**].

SC-5: Spurwechsel darf nicht auftreten, wenn kein sicherer Spurwechsel gewährleistet ist [**H5**].

SC-6: Kurvenfahren darf nicht auftreten, wenn keine Kurve vorhanden ist [**H6**].

SC-7: Das Fahrzeug darf nicht beschleunigen, wenn das Fahrzeug geparkt ist [**H7**].

SC-8: Das Fahrzeug darf nicht bremsen bzw. zum Stillstand kommen, wenn das Fahrzeug dadurch in einen unsicheren Zustand übergeht (z.B. Stillstand auf der Autobahn oder starkes Abbremsen, wenn sich ein Fahrzeug direkt dahinter befindet, . . .) [**H8**].

4.2 Schritt 2 - Control-Structure

In Schritt 2 werden die wichtigen Komponenten in einer Control-Structure zusammengefasst. Des Weiteren werden die Responsibilities der jeweiligen Controller benannt und die Safety Constraints verlinkt, die durch die Responsibilities sichergestellt werden sollen.

4.2.1 Control-Structure

Die High-Level Control-Structure Abbildung 4.1 bildet eine abstrahierte Form der Control-Structure. Aus dieser wird anschließend eine verfeinerte Control-Structure (Abbildung 4.2) erstellt, die dann zur Analyse in Schritt 3 und 4 genutzt wird. Der Fokus liegt hierbei auf der Openpilot Software.

Die fahrende Person kann das Fahrzeug ganz normal steuern. Das Fahrzeug ist aber ebenfalls via eines CAN-Busses mit einem im Comma Two/Three verbauten Panda verbunden. Darüber werden dann die Nachrichten ausgetauscht. Anschließend werden die Nachrichten im Panda übersetzt und dann via des USB-Kabels an das Mainboard übertragen. Auf diesem läuft dann die Openpilot Software.

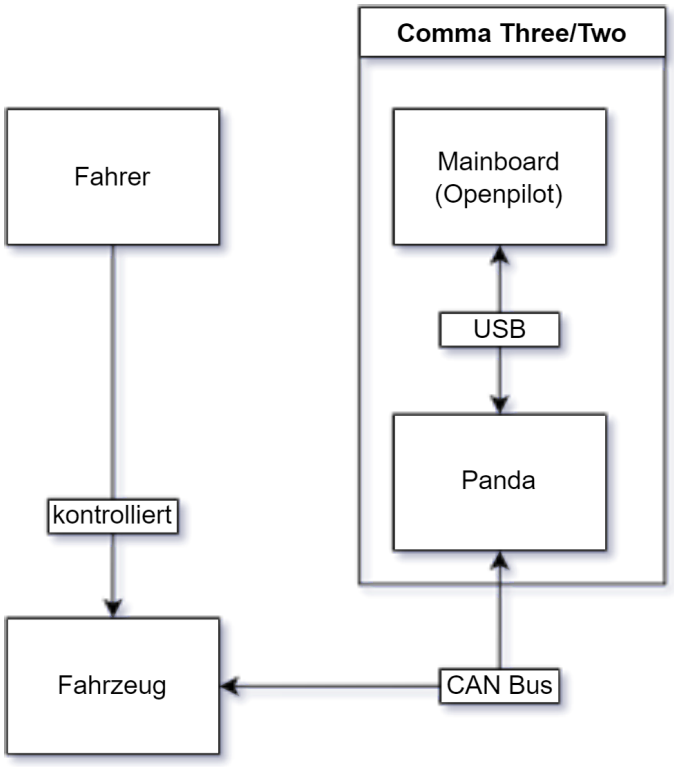

Abbildung 4.1: High-Level Control-Structure des Systems.

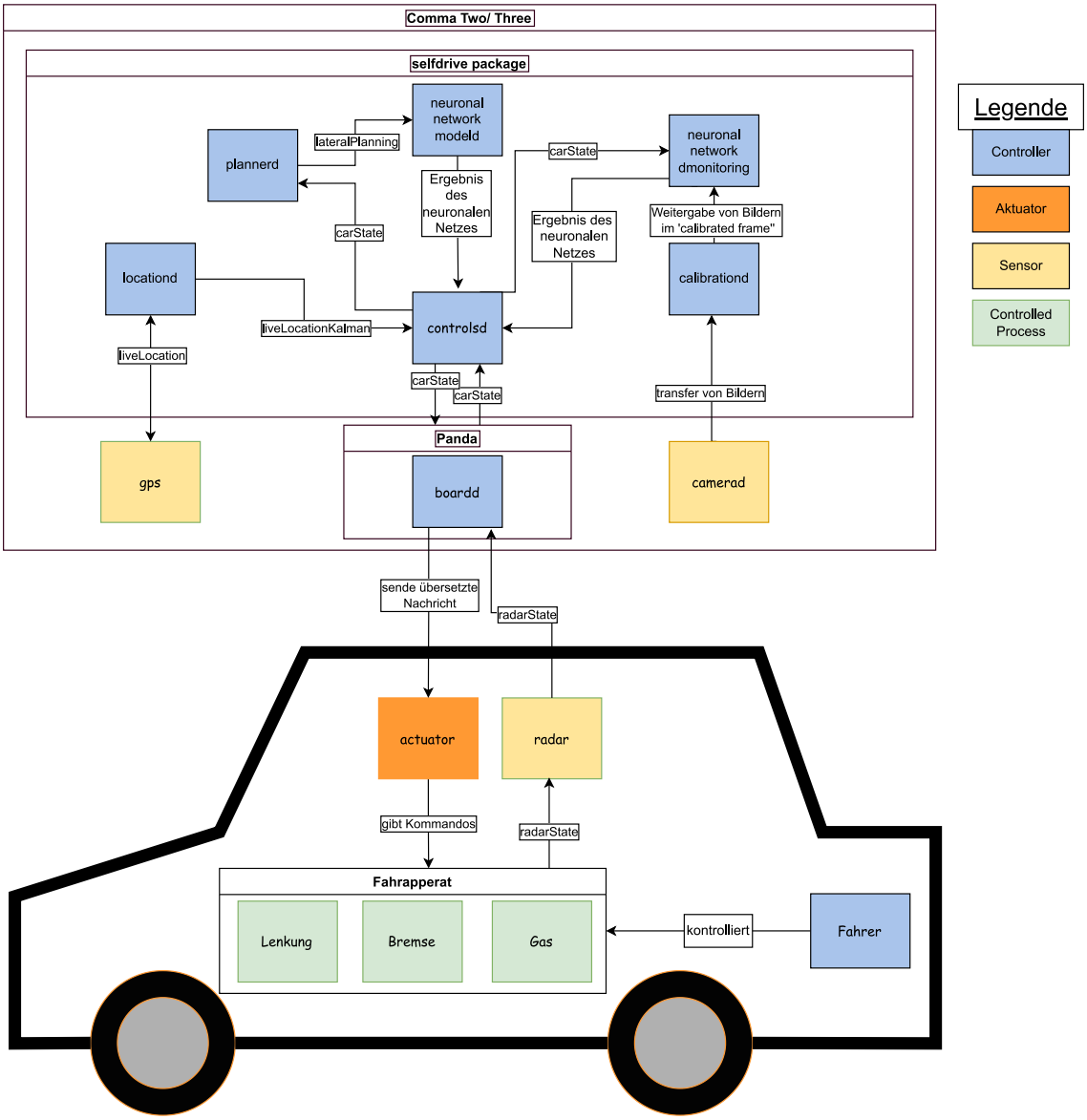

Abbildung 4.2: Midlevel Control-Structure des Selfdrive Pakets (modelliert aus Informationen von [comc; com21b]).

Controller

Die Controller nehmen eine wichtige Rolle in der Control-Structure ein. Sie kontrollieren andere Prozesse und erhalten von diesem Feedback über die ausgeführten Aktionen.

Die Controller und deren Aufgaben werden, im folgenden, aufgelistet:

C-1: *Fahrende Person*: Die fahrende Person kann die manuelle Kontrolle über die Lenkung, sowie die Beschleunigung und Bremsung des Fahrzeugs übernehmen.

C-2: *controlsd* ist die zentrale Kontrollkomponente. Zuerst werden alle Informationen, die vom Fahrzeug an die Software übertragen werden, mittels eines *carStates* gebündelt und an alle Komponenten weitergegeben, welche die Informationen benötigen. Komponenten, die Informationen verarbeiten, geben ihre Erkenntnisse und Ergebnisse an controlsd zurück. Die daraufhin gesammelten Ergebnisse werden dann wieder mittels des Pandas übersetzt und an das Fahrzeug übermittelt.

C-3: *plannerd*: Plannerd ist für die laterale Planung verantwortlich. Das bedeutet, dass die Lenkung durch Plannerd geplant wird.

C-4: *locationd*: Beinhaltet und verarbeitet die aktuelle Position des Fahrzeugs.

C-5: *dmonitoring*: Für die Eingabe in das neuronale Netz wird das von der Kamera aufgenommene Bild, das den Innenraum des Fahrzeugs filmt, in einen kalibrierten Rahmen gelegt. Dadurch wird sichergestellt, dass die Bilder so ähnlich wie möglich sind und dass die verschiedenen Fahrzeugmarken und -modelle das Ergebnis so wenig wie möglich beeinflussen.

Der Nutzen von Dmonitoring besteht darin, dass es erkennt, ob der Fahrer oder die Fahrerin aufmerksam ist oder ob er/sie zum Beispiel schläft. Wenn das nicht der Fall ist, wird die fahrende Person gewarnt, dass er oder sie jederzeit konzentriert sein muss.

C-6: *Calibrationd*: Calibrationd wird genutzt, um die Bilder in einen “calibrated Frame“ zu setzen und gibt sein Ergebnis dann an *dmonitoring* weiter.

C-7: *modeld*: Modeld gibt die ImageStreams verarbeitet weiter. Diese werden anschließend an verschiedene Komponenten (unter anderem dmonitoring) weitergegeben.

Sensoren und Actuators

- *board*: Der Board Daemon ist Teil der Übersetzung von den CAN Nachrichten, die zwischen dem Fahrzeug [**CP1**] und Openpilot [**C2**] ausgetauscht werden.
- *camerad*: Der Camera Daemon sendet die von den Kameras aufgenommenen Bilder an calibrationd [**C5**].

- *sensord*: Sensord bündelt die vorhandenen Sensorinformationen und sendet diese weiter, damit diese von controlsd [**C2**] verteilt werden können.

Controlled Processes

CP-1: *Fahrzeug*: Das Fahrzeug ist im abstrahierten Sinne der Controlled Process, der im Detail aus drei Komponenten besteht. Dabei gibt es das Gaspedal, die Lenkung und das Bremspedal.

CP-2: *Gaspedal*: Das Gaspedal wird vom Fahrer betätigt, oder die Beschleunigung wird durch die Software veranlasst.

CP-3: *Bremse*: Das Bremspedal wird vom Fahrer betätigt, oder die Bremsung wird durch die Software veranlasst.

CP-4: *Lenkung*: Die Lenkung wird vom Fahrer ausgeführt, oder die Lenkbewegung wird durch die Software veranlasst.

4.2.2 Responsibilities

Als Nächstes werden die Responsibilities der jeweiligen Controller aufgelistet und Verknüpfungen zu den jeweiligen Safety Constraints aufgezeigt.

C1 - fahrende Person

R-1: Die fahrende Person überwacht das System und übernimmt, wenn notwendig, die Kontrolle [**SC3**].

R-2: Die fahrende Person überwacht das System bei einem Spurwechsel und übernimmt, wenn notwendig, die Kontrolle [**SC5**].

R-3: Die fahrende Person überwacht das System bei dem Fahren einer Kurve und übernimmt, wenn notwendig, die Kontrolle [**SC6**].

R-4: Die fahrende Person überwacht das System bei der Beschleunigung und übernimmt, wenn notwendig, die Kontrolle [**SC7**].

R-5: Die fahrende Person überwacht das System bei der Bremsung und übernimmt, wenn notwendig, die Kontrolle [**SC8**].

C2 - Controlsd

R-6: Sammelt und verteilt alle Informationen, die zwischen der Software und dem Fahrzeug ausgetauscht werden [**SC1**].

C3 lateralPlanning

R-7: Plannerd verarbeitet die gegebenen Informationen und berechnet den Weg für einen Spurwechsel [**SC5**].

R-8: Plannerd verarbeitet die gegebenen Informationen und berechnet den Weg für die nächsten Sekunden, die eine Kurve beinhalten können [**SC6**].

R-9: Plannerd verarbeitet die gegebenen Informationen und berechnet den Weg, sodass der Mindestabstand eingehalten wird [**SC2**].

C4 locationd

R-10: Lokalisierung und Meldung der Position des Fahrzeugs [**SC4**].

C5 dmonitoring

R-11: Überprüfung, dass die fahrende Person in der Lage ist, die Kontrolle über das Fahrzeug zu übernehmen [**SC3**].

C6 calibrationd

R-12: Kalibrierung des Imagestreams, um die Bilder möglichst ähnlich zu gestalten [**SC1**].

C7 modeld

R-13: Modeld berechnet die Beschleunigung des Fahrzeugs [**SC7**].

R-14: Modeld berechnet die Bremsung des Fahrzeugs [**SC8**].

R-15: Modeld berechnet die Geschwindigkeit, um den definierten Mindestabstand einzuhalten [**SC2**].

4.3 Schritt 3 - Unsafe Control Actions

In Schritt 3 werden die Unsafe Control Actions identifiziert, die aus der in Schritt 2 modellierten Control-Structure abgeleitet wurden. Tabelle 4.1 betrachtet, die Control-Action *erteile den Bremsbefehl* und die möglichen unsicheren Control-Actions. Dabei gibt es die Möglichkeiten, dass Control-Actions, wenn sie bereitgestellt (*provided*), nicht bereitgestellt (*not provided*), zu früh oder zu spät oder außer der Reihe zur Verfügung gestellt werden (*Too soon, too late, out of order*) oder zu früh gestoppt oder zu lange ausgeführt werden *(stopped too soon, applied too long)* zu einem Hazard, in diesem Fall zu **H2** (*Das Fahrzeug kommt zu nah an andere Objekte*), führen. Dasselbe bildet Tabelle 4.2 für die Control-Action *erteile den Beschleunigungsbefehl*, sowie Tabelle 4.3 mit der Control-Action *erteile den Lenkungsbefehl*, ab.

Da der Fokus in dieser Masterarbeit auf dem neuronalen Netz liegt, werden nicht alle Control-Actions betrachtet, sondern nur die Control-Actions, die vom neuronalen Netz bis zum Fahrzeug relevant sind. Dies beinhaltet die Control-Action *Alarmierung der fahrenden Person*, das die Entscheidung des neuronalen Netzes in der Dmonitoring Komponente widerspiegelt. Abgebildet wird dies in Tabelle 4.4. Des Weiteren werden die Control-Actions *stelle das errechnete Beschleunigungsprofil bereit* und *stelle die errechnete Lenkung bereit* betrachtet, wie in Tabelle 4.5 zu sehen ist.

Control Action	Providing causes hazard	Not providing causes hazard	Too early, too late, out of order	Stopped too soon, applied too long
Erteile Bremsbefehl	**UCA-1:** Die Controlsdkomponete erteilt den Befehl zu Bremsen, wenn die Bremsung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**].	**UCA-2:** Die Controlsdkomponete erteilt den Befehl nicht zu Bremsen, obwohl die Bremsung eine Verletzung des Mindestabstands zu einem anderen Objekt verhindert würde [**H2**].	**UCA-3:** Die Controlsdkomponete erteilt den Befehl zu Bremsen zu früh, wenn die Bremsung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**]. **UCA-4:** Die Controlsdkomponete erteilt den Befehl zu Bremsen zu spät, wenn die Bremsung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**].	**UCA-5:** Die Controlsdkomponete erteilt den Befehl zum Bremsen. Dieser wird zu früh beendet, sodass die Bremsung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**]. **UCA-6:** Die Controlsdkomponete erteilt den Befehl zum Bremsen zu lange, sodass die Bremsung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**].

Tabelle 4.1: Übersicht der Unsafe Control-Actions, die von der Controlsdkomponente mit dem Bremsbefehl ausgelöst werden können, die sich auf Hazard **H2** beziehen.

Control Action	Providing causes hazard	Not providing causes hazard	Too early, too late, out of order	Stopped too soon, applied too long
Erteile Beschleunigungsbefehl	**UCA-7:** Die Controlsdkomponete erteilt den Befehl zum Beschleunigen, wenn die Beschleunigung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**].	**UCA-8:** Die Controlsdkomponete erteilt den Befehl nicht zu beschleunigen, obwohl die Beschleunigung eine Verletzung des Mindestabstands zu einem anderen Objekt verhindern würde [**H2**].	**UCA-9:** Die Controlsdkomponete erteilt den Befehl zur Beschleunigung zu früh, wenn die Beschleunigung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**]. **UCA-10:** Die Controlsdkomponete erteilt den Befehl zur Beschleunigung zu spät, wenn die Beschleunigung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**].	**UCA-11:** Die Controlsdkomponete erteilt den Befehl zum Beschleunigen, dieser wird zu früh beendet, sodass die Beschleunigung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**]. **UCA-12:** Die Controlsdkomponete erteilt den Befehl zur Beschleunigung zu lange, sodass die Beschleunigung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**].

Tabelle 4.2: Übersicht der Unsafe Control-Actions, die von der Controlsdkomponente mit dem Beschleunigungsbefehl ausgelöst werden können, die sich auf Hazard **H2** beziehen.

Control Action	Providing causes hazard	Not providing causes hazard	Too early, too late, out of order	Stopped too soon, applied too long
Erteile Lenkbefehl	**UCA-13:** Die Controlsdkomponete erteilt den Befehl zur Lenkung, wenn die Lenkung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**].	**UCA-14:** Die Controlsdkomponete erteilt den Befehl zur Lenkung nicht, obwohl die Lenkung eine Verletzung des Mindestabstands zu einem anderen Objekt verhinderen würde [**H2**].	**UCA-15:** Die Controlsdkomponete erteilt den Befehl zur Lenkung zu früh, wenn die Lenkung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**]. **UCA-16:** Die Controlsdkomponete erteilt den Befehl zur Lenkung zu spät, wenn die Lenkung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**].	**UCA-17:** Die Controlsdkomponete erteilt den Befehl zur Lenkung, dieser wird zu früh beendet, sodass die Lenkung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**]. **UCA-18:** Die Controlsdkomponete erteilt den Befehl zur Lenkung zu lange, sodass die Lenkung den Mindestabstand zu einem anderen Objekt verletzen würde [**H2**].

Tabelle 4.3: Übersicht der Unsafe Control-Actions, die von der Controlsdkomponente mit dem Lenkungssbefehl ausgelöst werden können, die sich auf Hazard **H2** beziehen.

Control Action	Providing causes hazard	Not providing causes hazard	Too early, too late, out of order	Stopped too soon, applied too long
Alarmierung der fahrenden Person	**UCA-17:** Die Komponente Dmonitoring kommt zu der Entscheidung, dass die Control-Action *alarmiere die fahrende Person,* ausgelöst wird, obwohl die Person aufmerksam ist [**H3**].	**UCA-18:** Die Komponente Dmonitoring kommt zu der Entscheidung, dass die Control-Action *alarmiere die fahrende Person,* nicht ausgelöst wird, während die Person unaufmerksam ist [**H3**].	**UCA-19:** Die Komponente Dmonitoring kommt zu der Entscheidung, dass die Control-Action *alarmiere die fahrende Person,* zu früh ausgelöst wird, obwohl die Person noch aufmerksam ist [**H3**]. **UCA-20:** Die Komponente Dmonitoring kommt zu spät zu der Entscheidung, dass die Control-Action *alarmiere die fahrende Person,* nicht ausgelöst wird, während die Person unaufmerksam ist [**H3**].	**UCA-21:** Die Komponente Dmonitoring kommt zu der Entscheidung, dass die Control-Action *alarmiere die fahrende Person,* ausgelöst wird und stoppt die Alarmierung zu früh wieder, während die Person unaufmerksam ist [**H2**]. **UCA-22:** Die Komponente Dmonitoring kommt zu der Entscheidung, dass die Control-Action *alarmiere die fahrende Person,* ausgelöst wird und führt die Alarmierung zu lange fort, obwohl die Person wieder aufmerksam ist [**H3**].

Tabelle 4.4: Übersicht der Unsafe Control-Actions, die von der Dmonitonigkomponente mit dem Befehl alarmiere die fahrende Person ausgelöst werden können.

Control Action	Providing causes hazard	Not providing causes hazard	Too early, too late, out of order	Stopped too soon, applied too long
Stelle das errechnete Beschleunigungsprofil bereit	**UCA-23:** Plannerd stellt die Control-Action *aktuelles Beschleunigungsprofil* bereit, obwohl die fahrende Person die manuelle Kontrolle über das Fahrzeug hat [**H1**].	**UCA-24:** Plannerd stellt die Control-Action *aktuelles Beschleunigungsprofil* nicht bereit, während sich das Fahrzeug in Bewegung befindet [**H7**].	**UCA-25:** Plannerd stellt die Control-Action *aktuelle Beschleunigungsprofil* zu spät bereit, während sich das Fahrzeug in Bewegung befindet [**H7**].	**UCA-26:** Plannerd stellt die Control-Action *aktuelle Beschleunigungsprofil* bereit, aber stoppt diese zu früh, während sich das Fahrzeug in Bewegung befindet [**H7**].
Stelle die berechnete Lenkung bereit	**UCA-27:** Plannerd stellt die *berechnete Lenkung* für die nächsten Sekunden bereit, obwohl die fahrende Person die manuelle Kontrolle über das Fahrzeug hat [**H1**].	**UCA-28:** Plannerd stellt die *berechnete Lenkung* für die nächsten Sekunden nicht bereit, während das Fahrzeug sich im normalen Fahrbetrieb befindet [**H6**].	**UCA-29:** Plannerd stellt die *berechnete Lenkung* für die nächsten Sekunden zu spät bereit, während das Fahrzeug sich im normalen Fahrbetrieb befindet [**H6**].	**UCA-30:** Plannerd stellt die *berechnete Lenkung* für die nächsten Sekunden bereit, stoppt diese aber zu früh, während das Fahrzeug sich im normalen Fahrbetrieb befindet [**H6**].

Tabelle 4.5: Übersicht der Unsafe Control-Actions, die von der Plannerd Komponente mit dem Befehl *Stelle das errechnete Beschleunigungsprofil bereit*, sowie mit dem Befehl Stelle errechnete Lenkung bereit, ausgelöst werden können.

4.4 Schritt 4 - Loss Scenarios

Zunächst gibt es die **physikalischen Failures**, die auftreten können. Dabei gibt es folgende Fälle:

L-1: das Mainboard, auf dem die Openpilot Software läuft, fällt aus.

L-2: das USB-Kabel, das den Panda mit dem Mainboard verbindet, ist defekt.

L-3: das Panda-Gerät, das als universelle CAN Schnittstelle genutzt wird, ist defekt.

L-4: die CAN-Kabel, die die physischen Elemente des Fahrzeuges mit dem CAN-Interface verbinden, sind/ ist defekt.

L-5: die physische Einheit, die für das Bremsen notwendig ist, ist defekt.

L-6: die physische Einheit, die für das Beschleunigen notwendig ist, ist defekt.

L-7: die physische Einheit, die für die Lenkung notwendig ist, ist defekt.

Die Komponete Dmonitoring kommt zu der Entscheidung, dass die Control-Action, *alarmiere die fahrende Person* ausgelöst wird. Die dazugehörigen UCAs werden in Tabelle 4.4 aufgezeigt und die infrage kommenden Loss Scenarios werden im Folgenden abgebildet:

L-8: Die Control-Action *alarmiere die fahrende Person* wird ausführt, obwohl die fahrende Person aufmerksam ist. Dadurch kann sie verwirrt und abgelenkt werden und kann deshalb, wenn nun eine Gefahrensituation auftritt, wie dass man einem anderen Objekt zu nahe kommt (**H2**), nicht eingreifen.

L-9: Obwohl die Person nicht aufmerksam ist, wird sie nicht alarmiert, sodass die Aufmerksamkeit der fahrenden Person nicht zurück zur Straße geholt wird. Dadurch kann das Fahrzeug zu nah an andere Objekte geraten (**H2**), wenn die Person im Fehlerfall nicht übernehmen kann.

L-10: Trotz dass die fahrende Person aufmerksam ist, wird sie alarmiert, dass sie aufmerksam sein muss. Allerdings hat die fahrende Person zu dem Zeitpunkt der Alarmierung, die Kontrolle über das Fahrzeug. Durch den Alarm wird die Person nun abgelenkt und kann zu nah an andere Objekte geraten (**H2**).

L-11: Die fahrende Person ist unaufmerksam und wird zu spät alarmiert, dass sie aufmerksam sein muss, dadurch kann das Fahrzeug zu nah an andere Objekte geraten (**H2**), weil die Person durch die Unaufmerksamkeit nicht schnell genug eingreifen kann.

L-12: Die Person wird alarmiert, dass sie ihr Verhalten verändern muss, um in der Lage zu sein, die Kontrolle zu übernehmen, bevor das Fahrzeug bspw. einem anderen Objekt zu nahe kommt (**H2**). Der Alarm stoppt, obwohl die Person ihr Verhalten nicht ändert.

L-13: Die fahrende Person wird alarmiert, dass sie ihr Verhalten verändern muss und in der Lage sein muss, die Kontrolle zu übernehmen, bevor das Fahrzeug bspw. einem anderen Objekt zu nahe kommt (**H2**). Der Alarm stoppt aber nicht, obwohl die Person aufmerksam ist, dadurch kann die Person verwirrt werden.

Die Komponente Plannerd stellt die Control-Action, *gebe errechnetes Beschleunigungsprofil weiter*, dar, die dazugehörigen UCAs werden in Tabelle 4.5 aufgezeigt und die entsprechenden Loss Scenarios werden im Folgenden abgebildet

L-14: Das Fahrzeug bewegt sich im normalen Verkehr, erhält aber für den nächsten Zeitraum kein Beschleunigungsprofil, sodass die Beschleunigungen und Abbremsungen auf Grundlage von Berechnungen nicht vorliegen. Dadurch kann das Fahrzeug für die fahrende Person und andere Verkehrsteilnehmer unvorhersehbar bremsen (**H8**) oder beschleunigen (**H7**). Dies kann dann nur verhindert werden, wenn die am Steuer sitzende Person eingreift.

L-15: Das aktuelle Bewegungsprofil wird zu spät bereitgestellt, da eine Verzögerung bei der Berechnung aufgetreten ist. Dadurch stehen keine Daten zur Bremsung und Beschleunigung für den folgenden Zeitraum bereit. Das bedeutet, dass die Gefahr von ungeplanten Beschleunigungen (**H7**) oder Bremsungen (**H8**) besteht, die nur verhindert werden können, wenn die am Steuer sitzende Person die Gefahr erkennt und eingreift oder auf den vorliegenden Fehler vom System hingewiesen wird.

L-16: Das aktuelle Bewegungsprofil wird zwar bereitgestellt, aber wird zu früh gestoppt. Hierdurch entsteht eine Lücke, was zur Folge hat, dass zu dieser Zeit keine Daten zur Beschleunigung bzw. Bremsung vorliegen ((**H7**, **H8**). So muss auch hier die fahrende Person eingreifen, um daraus resultierende Gefahren abzuwenden. Entstehen kann dieser Fehler aber nicht nur durch Verzögerungen, die im Prozess auftreten, sondern auch durch physische Fehler, wie defekte CAN-Kabel (**L4**).

Die Komponente Plannerd stellt die Control-Action, *gebe errechnete Lenkbewegung weiter*, dar. Die dazugehörigen UCAs werden in Tabelle 4.5 aufgezeigt und die entsprechenden Loss Scenarios werden im Folgenden abgebildet

L-17: Das Fahrzeug erhält, während es sich im Fahrbetrieb befindet, keine Informationen darüber, wie die errechneten Lenkwinkel im nächsten Zeitraum aussehen. Dadurch kann es zu ungeplanten Bewegungen des Fahrzeugs kommen, die eine Gefährdung (**H6**) darstellen und dann nur durch die aufmerksame Person am Steuer verhindert werden können.

L-18: Das Fahrzeug erhält, während es sich im Fahrbetrieb befindet, erst verzögert, also zu spät, Informationen über die errechneten Lenkbewegungen. Dadurch ist für einen gewissen Zeitraum die Bewegung nicht definiert, wobei durch die ungeplante Lenkbewegung (**H6**) eine Gefahr entsteht. Diese muss von der fahrenden Person, mittels einer manuellen Fahrzeugsteuerung abgefangen werden.

L-19: Das Fahrzeug erhält, während es sich im Fahrbetrieb befindet, Informationen, aber beendet diese Übertragung zu früh. Dies bedeutet, dass für den Zeitraum nicht berechnete Lenkbewegungen vorhanden sind. Dieser unvollständige Zeitraum stellt damit die gleiche Gefährdung (**H6**) dar, wie die beiden vorherigen Loss Scenarios.

5 Diskussion

STPA erkennt offensichtliche generische Schwachstellen in neuronalen Netzen.

Aber um tatsächliche Daten abzugleichen und zu prüfen, ob Fehler innerhalb des neuronalen Netzwerkes auftreten, ist STPA ungeeignet. Es erreicht keine ausreichende Abdeckung an Fällen, die innerhalb eines neuronalen Netzes auftreten können. Doch diese stellen Unsicherheiten dar. Deshalb müssen sie, im Zweifel, durch eine Erweiterung von STPA abgedeckt werden. Herausforderungen, die bei diesen Erweiterungen adressiert werden müssen, sind:

1. Die Veränderungen, die über die Zeit auftreten können, müssen separat betrachtet werden.
2. Die Risikoabschätzungen der Loss Scenarios, wie häufig so ein Scenario auftreten kann, sind schwierig zu ermitteln, ohne konkrete Daten über bspw. die Genauigkeiten des neuronalen Netzwerks zu haben.
3. Um neuronale Netze im Detail zu betrachten, muss die STPA-Analyse in Teilen bis auf Codeebene heruntergebrochen werden, da einzelne Variablenwerte oder Datenströme hierbei entscheidend sein können. Dies ist beispielsweise dann der Fall, wenn das neuronale Netz von mehreren Inputwerten abhängig ist.
4. Ob und wie die neuronalen Netzwerke konkret trainiert worden sind, ist entscheidend dafür, welche Fehler im Gebrauch des neuronalen Netzwerkes auftreten können.
5. Die Anzahl der Loss Scenarios, die aufgrund eines neuronalen Netzwerkes auftreten können, sind zahlreich. Dadurch ist die Wahrscheinlichkeit. dass eine Analyse unvollständig ist, wahrscheinlich.
6. Aus STPA lassen sich nur schwierig, für einzelne Szenarien, Constraints ableiten, die alle Anforderung des neuronalen Netzes umsetzen können.

In der klassischen Safety Definitionen waren KI und deren Algorithmen noch nicht vorhergesehenen. Daher ist es bei allen gängigen Definitionen, die derzeit existieren, wichtig, sich an die neuen Gegebenheiten anzupassen. Dafür wird bereits versucht, die ISO 26262 Norm anzupassen bzw. zu erweitern [ISO21].

Dadurch wird deutlich, dass dieses Feld derzeit ein Themenfeld darstellt, in dem viele neue Richtlinien aufgestellt werden und der Fokus auf Safety in ML Anwendungen wächst, da die Anwendungen immer mehr in sicherheitskritische Anwendungsbereiche vorstoßen.

Es gibt mehrere Möglichkeiten STPA zu erweitern, um sicherheitskritische KI-Bereiche besser zu überwachen und die Sicherheit zu gewährleisten. Da die verschiedenen Erweiterungsmöglichkeiten auf unterschiedlichen Ebenen ansetzen, werden diese im Folgenden vorgestellt und diskutiert.

5.1 Erweiterung um SOTIF

Bei Safety of the Intended Functionality (SOTIF) handelt es sich um den ISO Standard 21448, der fehlerhaftes Verhalten bei beabsichtigten Funktionen überdenkt. Dieser ist eine Ergänzung zu ISO26262, der unbeabsichtigtes fehlerhaftes Verhalten betrachtet. Denn das System kann immer noch fehlerhaft sein, auch wenn es nach ISO26262 sicher ist. Innerhalb des zulässigen Betriebsbereichs können Gefährdungen durch Wahrnehmungen in der Umwelt oder auch durch nicht ausreichende Robustheit auftreten.

SOTIF stellt dafür die Risiken fest, die aufgrund von Funktionsmängeln auftreten können und bietet dabei einen Prozess, der durch seine Systematik Risiken aufdeckt, die unangemessen hohe Problematiken mit sich bringen. Die Zielsetzung von SOTIF ist es, die identifizierten Risiken anhand von Akzeptanzkriterien zu bewerten und dann die Wahrscheinlichkeit für die Szenarien, die bekannt oder auch unbekannt sein können und ein gefährliches Verhalten abbilden, zu verringern [ACA+22; BHS+22].

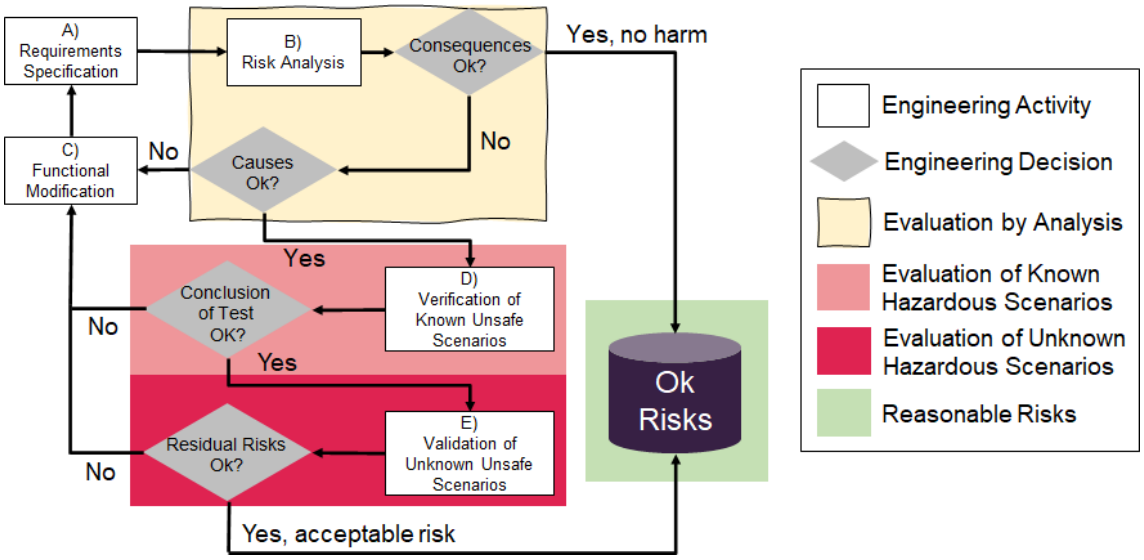

Abbildung 5.1: Aufbau des SOTIF Prozesses [BHS+22].

Der Ablauf des Prozesses ist vereinfacht in Abbildung 5.1 dargestellt. Zunächst werden in A) die Anforderungen spezifiziert, danach wird in B) die Risikoanalyse durchgeführt. Anschließend werden die Folgen identifiziert, die potenziell durch die in B) erkannten Risiken hervorgerufen werden können. Wenn das Risiko keinen bzw. einen vertretbaren Schaden verursacht, wird das Risiko in Grün gekennzeichnet, also als akzeptables Risiko gewertet. Sollte das Risiko aber einen Schaden verursachen, der nicht vertretbar ist, wird anschließend die Ursache des Szenarios analysiert. Dabei liegt der Fokus auf der Identifizierung der Bedingung, die ausgelöst wird und deren Bewertung. Wenn diese zu dem Schluss kommt, dass die Systemantwort auf Auslösung dieser Bedingung nicht akzeptabel ist, muss C) ausgeführt werden. Dies bedeutet, dass die Funktion passend zu

den gegebenen Anforderungen modifiziert werden muss. Ist die Systemantwort akzeptabel, wird als nächstes D) ausgeführt. Hier wird verifiziert, dass das System weiß, wie es mit den Anforderungen der bekannten unsicheren Szenarien umgehen kann. Falls dies nicht der Fall ist, wird wieder C) ausgeführt. Anderenfalls wird E) ausgeführt, wobei noch zusätzlich validiert wird, ob es noch weitere Szenarien gibt, durch die Gefährdungen entstehen können. Wenn hierbei noch weitere Szenarien entdeckt werden, werden diese zu bekannten Gefährdungen. Als Letztes wird nun die Wahrscheinlichkeit von unbekannten Szenarien bestimmt, durch deren Eintritt eine Gefährdung besteht. Wenn die abgeschätzte Wahrscheinlichkeit klein genug ist, wird das Risiko als Grün, also akzeptable eingestuft. Falls das nicht der Fall ist, wird wieder nach C) zurückgegangen [BHS+22].

Warum muss nun STPA zusätzlich verwendet werden? Die Zusammenfügung von ISO 26262 und ISO/FDIS von der Celik et al. [ACA+22] führt auf Arbeiten von Becker et al. [BBY20] und Kirovskii et al. [KG19] zurück. STPA wird nun zusätzlich in diese Kombination integriert, um die Lücken, die durch die Verwendung von ISO 26262 und ISO/FDIS 21448 noch bestehen, zu schließen. So wird in der ISO 26262 keine Richtlinie festgelegt, wie der Umgang mit Deep Learning Algorithmen, in Bezug auf Safety, aussehen muss. ISO/FDIS 24811 bezieht sich auf die Messeinheiten, die für Safety verwendet werden sollen. Doch auch hier sind diese nicht für Maschine Learning Algorithmen ausgelegt.

Zunächst wird STPA in den ISO 26262 Zyklus, sowie in ISO/FDIS 21448 eingeordnet. Dazu wurde von Celik et al. [ACA+22] eine Übersichtsgrafik (Abbildung 5.2) erstellt, die folgendes darstellt: Die Item-Definition, die Hazard Analyse, die Risikoanalyse sowie die Festlegung der Safety-Ziele des ISO 26262 Standards werden mit dem Schritt 1 und 2 (in Abbildung 5.2 als Schritt 0) in STPA als überlappend angesehen. In ISO/FDIS 21448 wird in diesem Schritt die Spezifikation und das Design festgelegt, sowie die Hazards definiert und anschließend evaluiert [ACA+22].

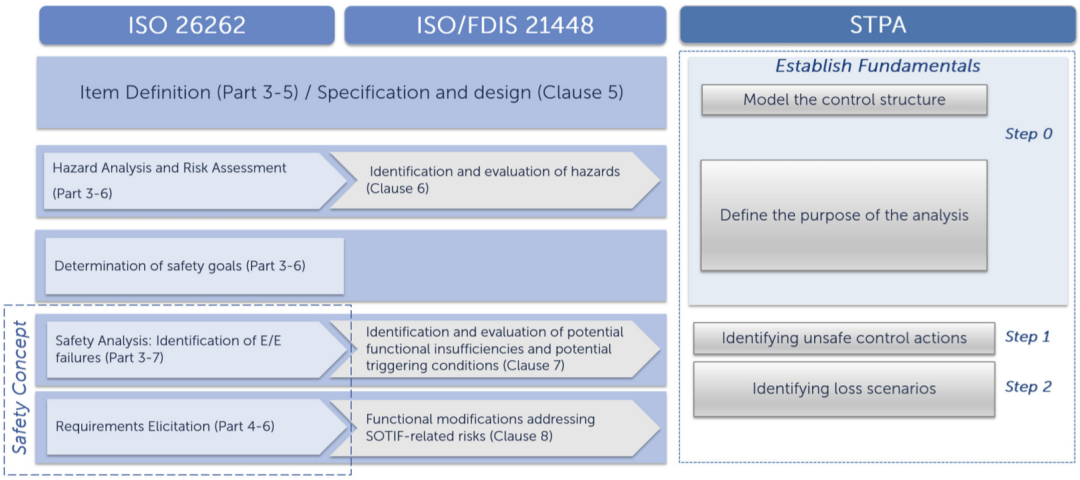

Abbildung 5.2: Integration des STPA Prozesses in ISO 26262 und ISO/FDIS 21448 [ACA+22].

Das vorgeschlagene Safety Konzept bezieht sich nun auf die folgenden beide Schritte der ISO 26262. Zunächst werden in der Sicherheitsanalyse die E/E Failures identifiziert und darauf aufbauend die Anforderungen erhoben, die erfüllt werden müssen. ISO/FDIS 21448 behandelt zunächst die Identifizierung von potenziell mangelhaften Funktionen. Dazu wird evaluiert, welche Schwachstellen im System vorliegen und durch welche Bedingungen diese getriggert werden können. Anschließend werden auf dieser Grundlage Modifikationen auf funktionaler Ebene durchgeführt, die auf SOTIF-bedingte Risiken abzielen. In STPA sind hier der Schritt 3, Identifizierung der UCA's (in Abbildung 5.2 Schritt 1) und Schritt 4, die Identifizierung der Loss Scenarios (in Abbildung 5.2 Schritt 2), angesiedelt [ACA+22].

Durch die Identifizierung der Loss Scenarios soll nun die Erkennung von unsicheren Szenarien unterstützt werden, die in SOTIF verwendet werden. Durch die Identifizierung der Szenarien in STPA soll die Anzahl an unbekannten Szenarien reduziert werden. Denn nur wenn die Szenarien bekannt sind, ist es möglich diese zu eliminieren. Dies geschieht dann über ein Sicherheitskonzept, dessen Rahmen in der ISO 26262 definiert wird. Alternativ können im Rahmen der ISO/FDIS 21448 funktionale Modifikationen getroffen werden [ACA+22].

Zunächst werden dann, wie bei STPA üblich, die Schritte 1-3 durchgeführt. Anschließend werden in Schritt 4 die Loss Scenarios, die eine Komponente mit einem neuronalen Netz beinhaltet, wie in Abbildung 5.4 zu sehen ist, definiert. Dabei gibt es *Triggering Conditions*, diese sorgen im Kontext einer bestimmten *Operational Situation*, durch die *Functional Insuffency* dafür, dass eine fehlerhafte neuronale Netzwerk-Ausgabe entsteht. Aus dieser fehlerhaften Ausgabe entstehen Loss Scenarios. Der Ansatz ist deshalb wichtig, da hierbei die *Functional Insuffiencies* der ISO 21448 in den Vordergrund treten [ACA+22].

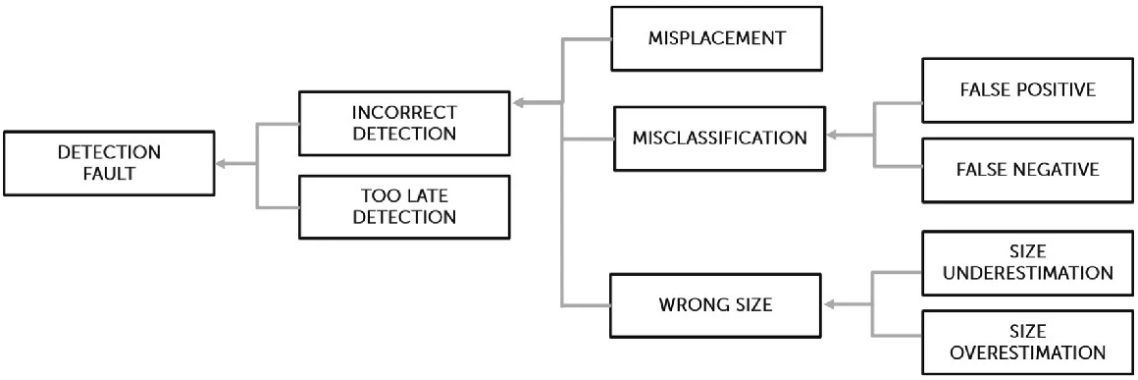

Abbildung 5.3: Taxonomie über die Gründe, die zu fehlerhaften Ausgaben eines neuronalen Netzwerkes führen können [ACA+22].

Dabei gibt es verschiedene Gründe, weshalb bei den neuronalen Netzwerken fehlerhafte Ausgaben auftreten. Diese werden von Celik et al. [ACA+22] in einer Taxonomie (Abbildung 5.3) zusammengefasst.

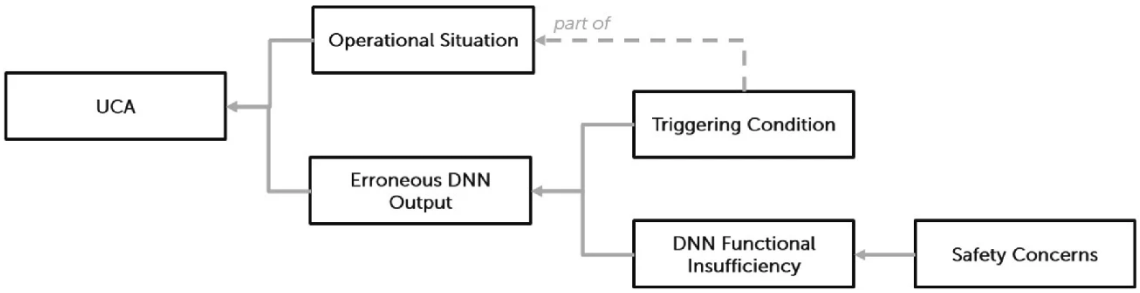

Abbildung 5.4: Beziehungen zwischen den einzelnen Faktoren, die in letzter Instanz zu einem Loss Scenario führen [ACA+22].

Die Szenarien, die von den Autoren Celik et al. [ACA+22] dabei als besonders relevant eingeschätzt wurden, sind zum einen, dass der Controller ein inkorrektes Feedback von Komponenten erhält, die das neuronale Netzwerk beinhaltet und zum anderen, dass der Controller kein Feedback zum benötigten Zeitpunkt bekommt. Dies kann zum Beispiel passieren, wenn das Feedback zu spät kommt.

Für das Auftreten eines inkorrekten Feedbacks kann es verschiedene Auslöser geben. So kann laut den Faktoren von SOTIF sein, dass das neuronale Netzwerk aufgrund von Sicherheitsanliegen Funktionsmängel aufweist. Laut ISO 26262 ist es auch möglich, dass Fehler in der Implementierung des neuronalen Netzes vorliegen. Dann werden diese UCAs als Verletzung der funktionalen Sicherheit angesehen [ACA+22].

Zieht man nun ISO/FDIS 21448 in Betracht, so muss das Ergebnis ein Akzeptanzkriterium sein. Dies sollte für jede abgeleitete Sicherheitsanforderung gelten und kann beispielsweise die *"minimum lenght of the required endurance run combined with a maximum number of observed failures for each type"* [ACA+22, Seite 328] sein. Dabei sind die relevanten Metriken, um Safety einzuhalten, im Regelfall Performance-Metriken.

Für die Identifizierung der Metriken, die verwendet werden sollen, werden zunächst die Sicherheitsbedingung aufgestellt, die zu einem Loss Scenario gehören. Anschließend werden basierend auf dieser Bedingungen die Sicherheitsbedenken formuliert und daraufhin die sicherheitsrelevanten Metriken abgeleitet, die die Sicherheit im besten Fall gewährleisten sollen [ACA+22].

Ein Beispiel hierzu, wird anhand von Openpilot aufgezeigt: *Das neuronale Netz der Komponente plannerd, erkennt das vorausfahrende Fahrzeug, auch wenn die Wetterbedingungen schlecht sind, zuverlässig und gibt ein Bewegungsprofil heraus. Die Sicherheitsbedenken sind dabei die Brittleness des neuronalen Netzwerkes, das mit einer Robustness-Metrik bewertet werden kann.*

Der Ansatz bietet die Möglichkeit, sehr nah an den ISO-Standards zu bleiben. Denn diese werden in der Praxis genutzt und sind erprobte Mittel, um Sicherheit zu gewährleisten. Allerdings ist dieser Ansatz sehr theoretisch und die Erweiterung beschäftigt sich mit Möglichkeiten, wie Sicherheit für dieses Szenario überprüft werden kann. Die aufgestellten

Metriken, die die Sicherheit gewährleisten sollen, müssen allerdings in einem separaten Schritt dann umgesetzt werden. Die anderen beiden Ansätze, die in dieser Masterarbeit diskutiert werden, sind darauf bedacht, die Sicherheit während der Ausführung der Software umzusetzen und basieren auf den Loss Scenarios.

5.2 Ergänzung von STPA um ein Monitoringsystem

STPA kann auch als Möglichkeit genutzt werden, um auf dessen Grundlage ein Monitoringsystem für das KI-basierte System aufzubauen. Dies spricht die Herausforderungen an, dass eine künstliche Intelligenz Entscheidungen trifft und es sich hierbei um eine Blackbox handelt. Ergebnisse sind hierbei zwar erwartbar, aber nicht vorhersehbar. Loss Scenarios sind somit teilweise nur Mutmaßungen darüber, welche gefährlichen Situationen auftreten könnten. Um diese Schwachstelle anzugehen, kann ein Monitoring System aufgebaut werden, aus dem später vielleicht auch Informationen darüber gezogen werden können, an welcher Stelle noch Probleme vorhanden sind. Da aber die Probleme, wie bereits in den Herausforderungen (Kapitel 5) adressiert, sehr vielseitig sein können, kann die Lösung ebenfalls vielfältig sein. Eine mögliche Lösung wäre es, die gefährlichen Situationen in sichere Zustände zu überführen.

Dazu kann auf der Grundlage einer ausführlichen STPA-Analyse ein Monitoringsystem aufgebaut werden. So soll eine Verbindung zwischen den theoretischen Hazards und den identifizierten Loss Scenarios zu einem Monitoringsystem geschaffen werden. Damit werden die Erkenntnisse der STPA-Analyse in die Architektur des Monitoringsystems übertragen. Ziel der Monitoringkomponente ist es, Hazards zur Laufzeit erkennen zu können, die durch die Verletzungen von Sicherheitsconstraints oder fehlerhaften Anforderungen entstehen können. Das Paper *"Loss Scenarios and Causal Factors as Design Guides for Multilevel Runtime Monitoring"* von Gautham et al. [GBW+22] behandelt diesen Ansatz und stellt den Prozess für den Aufbau eines solchen Monitors vor. Ein Überblick dieses Ansatzes ist in Abbildung 5.5 abgebildet.

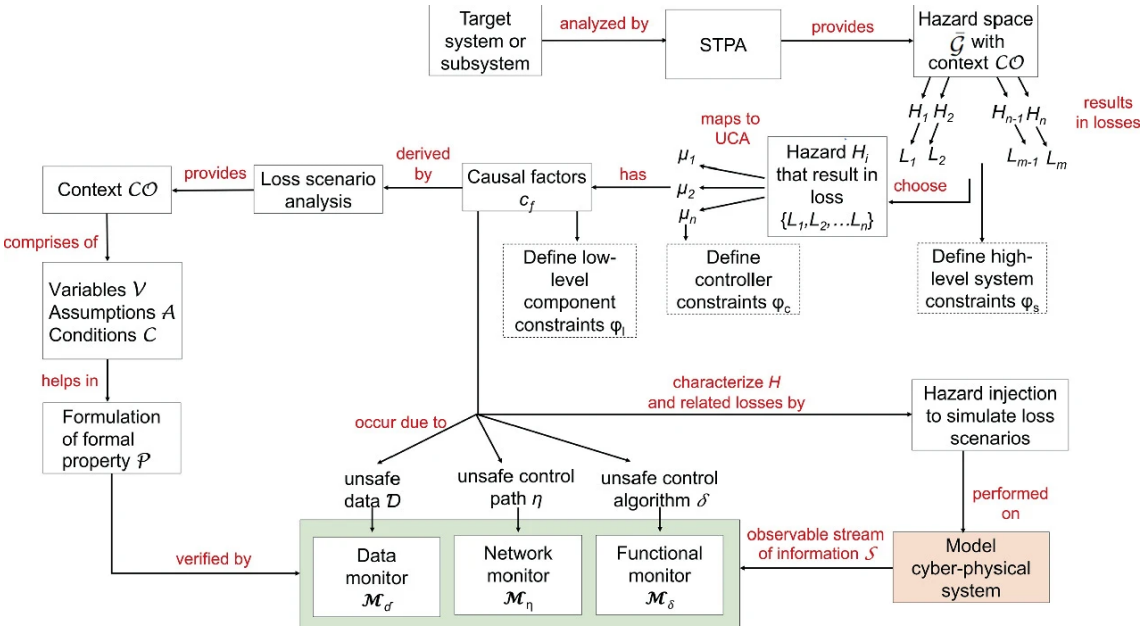

Abbildung 5.5: Abbildung des Prozesses, wie aus der STPA-Analyse ein Monitoring System abgeleitet werden kann [GBW+22].

Zunächst wird, wie bei der STPA-Analyse üblich, das zu analysierende System identifiziert. Anschließend wird die Analyse selbst auf diesem durchgeführt. Die daraus entstandenen Erkenntnisse werden nun weiterverarbeitet. Aus den identifizierten Hazards $H = H_1, H_2, ..., H_m$ und den Losses $L = L_1, L_2, ..., L_n$ werden die High-Level Constraints φ_s abgeleitet. Danach werden die Hazards auf die dazugehörigen UCAs $\mu = \mu_1, \mu_2, ..., \mu_o$ aufgelistet, aus denen sich anschließend die Controller Constraints φ_c ableiten lassen [GBW+22].

Nun muss der Kontext CO identifiziert werden, in dem die UCA auftritt, denn dieser entscheidet, ob daraus eine unsichere oder sichere Control-Action entsteht. Diese werden dann in unterschiedlichen Mengen notiert. So werden in der Menge α alle sicheren und in μ alle unsichere Control-Actions gesammelt. Ein Hazard H_i kann aber mehrere unsichere Control-Actions besitzen, sodass dies eine Menge u_k ist, die die zum Hazard gehörenden UCAs beschreibt. Durch die Definition dieser Mengen soll gewährleistet werden, dass, wenn eine identifizierte unsichere Control-Action auftritt, diese in eine sichere Control-Action überführt werden kann, um so zu vermeiden, dass ein gefährlicher Systemzustand entsteht [GBW+22].

Dazu sind Causal Factors c_f notwendig, denn diese führen dazu, dass in einem gewissen Kontext CO Control-Actions sicher oder unsicher sind. Um diese zu erkennen, werden die Loss Scenarios von STPA herangezogen. Aus diesen werden für die Kontexte drei relevante Informationen herausgefiltert [GBW+22]:

- V kann je nach Systemzustand, Umwelt und in diesem Fall Fahrzeugbedingungen verschiedene Werte haben
- A hat je nach getroffenen Annahmen unterschiedliche Variablenwerte
- C basierend auf den Variablen V und den Annahmen A werden Systembedingungen definiert

Diese drei stellen die Grundlage für die Causal Factors dar, die auftreten können. Denn wenn die Variable V von der Norm abweicht und somit die Annahmen A verletzt und durch die Bedingungen C auftritt, bilden sie die Grundlage für einen Causal Factor. und infolgedessen somit für eine unsichere Control-Action. Die daraus resultierenden Hazards müssen nun im besten Fall verhindert werden.

Wenn die Causal Factors identifiziert werden, müssen davon die low-level component constraints φ_i abgeleitet werden. Nun sind alle von STPA benötigten Faktoren so formalisiert worden, dass diese zur Generierung der einzelnen Monitorteile genutzt werden können. Dabei werden sie von Gautham et al. [GBW+22] in drei Teilen unterschieden, wie auch in Abbildung 5.5 zu sehen ist.

- Der DataMonitor M_{cf} überprüft die unsicheren Daten. Im Detail sind das die Daten, die von Sensoren und Aktoren im Austausch mit den physikalischen Stellen des Systems stattfinden. Hiermit können auch Teile, die mit der KI in Relevanz stehen, auf ihre Funktionalitäten überprüft und beobachtet werden. *So kann hier beispielsweise*

der Carstate der von der physischen Komponente in die Software übergeben wird, auf Unregelmäßigkeiten überprüft werden. Ein Beispiel für einen unsicheren Zustand wäre, wenn das Fahrzeug sich bewegt und als aktuelle Geschwindigkeit 0 km/h angegeben wird.

- Der Netzwerkmonitor M_η überwacht die Datenströme, die übertragen werden. *Ein Beispiel hierfür ist das camerad Streaming, das als Input für das neuronale Netzwerk genutzt wird und daher gut überwacht werden sollte. Dies ist deshalb relevant, da Bildinputs maßgeblich für die Entscheidungen des neuronalen Netzwerkes sind. Ein Beispiel für einen problematischen Input wäre beispielsweise ein schwarzes Bild, auf dessen Grundlage keine Entscheidung getroffen werden kann, ob die fahrende Person aufmerksam ist.*
- Der funktionale Monitor M_δ identifiziert die unsicheren Controllalgorithmen. M_δ wird mittels *"functional correctness properties"* den Relationen zwischen den Eingaben und den Ausgaben überprüft. Dies bedeutet, dass M_δ die kausalen Faktoren, die auftreten können, überwacht. Dies beinhaltet unter anderem z. B. unsichere Control-Algorithmen, aber auch bspw. interne Variablen. *Ein Beispiel für einen unsicheren Control-Algorithmus ist, wenn Controlsd von Plannerd eine Fahranweisung erhält, die zu einem gefährlichen Zustand führen würde.*

Das System wird somit überwacht, was dafür sorgt, dass Veränderungen, die über die Zeit innerhalb der neuronalen Netze entstehen, entdeckt werden können, wenn die Informationen des Monitoringsystems beobachtet werden.

Um das Monitoringsystem aufstellen zu können, muss zunächst die Control-Structure modelliert worden sein. Hierzu wurden während der Masterarbeit einige Erkenntnisse gewonnen, die nun vorgestellt werden. Es wurde festgestellt, dass der Detaillierungsgrad der Control-Structure ein maßgeblicher Schritt ist. Um STPA durchzuführen wurde die Codeebene zunächst angedacht, dann aber festgestellt, dass das *Mid-Level (Abbildung 4.2)* besser geeignet war, da hier die Informationen, die zwischen den Komponenten ausgetauscht wurden, gebündelt dargestellt werden. Die Detaillierung auf Codeebene zwischen den einzelnen Variablen hätte in Bezug auf die Masterarbeit keinen Vorteil ergeben, da der Informationsfluss der Komponenten, die ein neuronales Netz beinhalten, identifiziert werden muss. Die Erweiterung um ein Monitoringsystem zeigt nun aber, dass es notwendig ist, die Control-Structure, bei dieser Erweiterung, bis auf Code bzw. Variablenebene herunterzubrechen. Im Falle von Openpilot sehen die essenziellen Teile des selfdrive Packages in einer hierarchischen Control-Structure aus, wie in Abbildung 5.6 dargestellt. Allerdings wird hierdurch auch schnell ersichtlich, welche Nachteile ein solcher Detaillierungsgrad hat, vor allem wenn fachfremde Personen diesen erstellen sollen. Da hierbei der Code bis auf Codeebene verstanden werden muss und die Funktionalitäten, die zwischen den Komponenten stattfinden nicht ausreichen. Außerdem muss eine Abwägung getroffen werden, an welchen Stellen so ein Detaillierungsgrad die Analyse bereichert und an welchen diese Detaillierung die Analyse unnötig erschwert. Daher wäre es möglich, dies bspw. nur für Komponenten durchzuführen, die von sicherheitskritischer Bedeutung sind.

Durch die genaue Beschäftigung mit der STPA-Analyse werden nicht nur fehlerhafte Szenarien erkannt, sondern auch sichere Zustände definiert, die in Zukunft auch in Bezug auf die nächsten Automatisierungslevel des Fahrens eine Grundlage bieten.

Dabei gibt es zwei Möglichkeiten. Man kann die Szenarien bereits im Entwurf festlegen oder die Szenarien, sowie unsichere und sichere Control-Actions mit dem Wachsen des Systems erweitern, umso einen vollständigen Überblick zugewinnen. Dies kann ebenfalls mit viel Aufwand bis ins Detail erarbeitet und darauf angewendet werden.

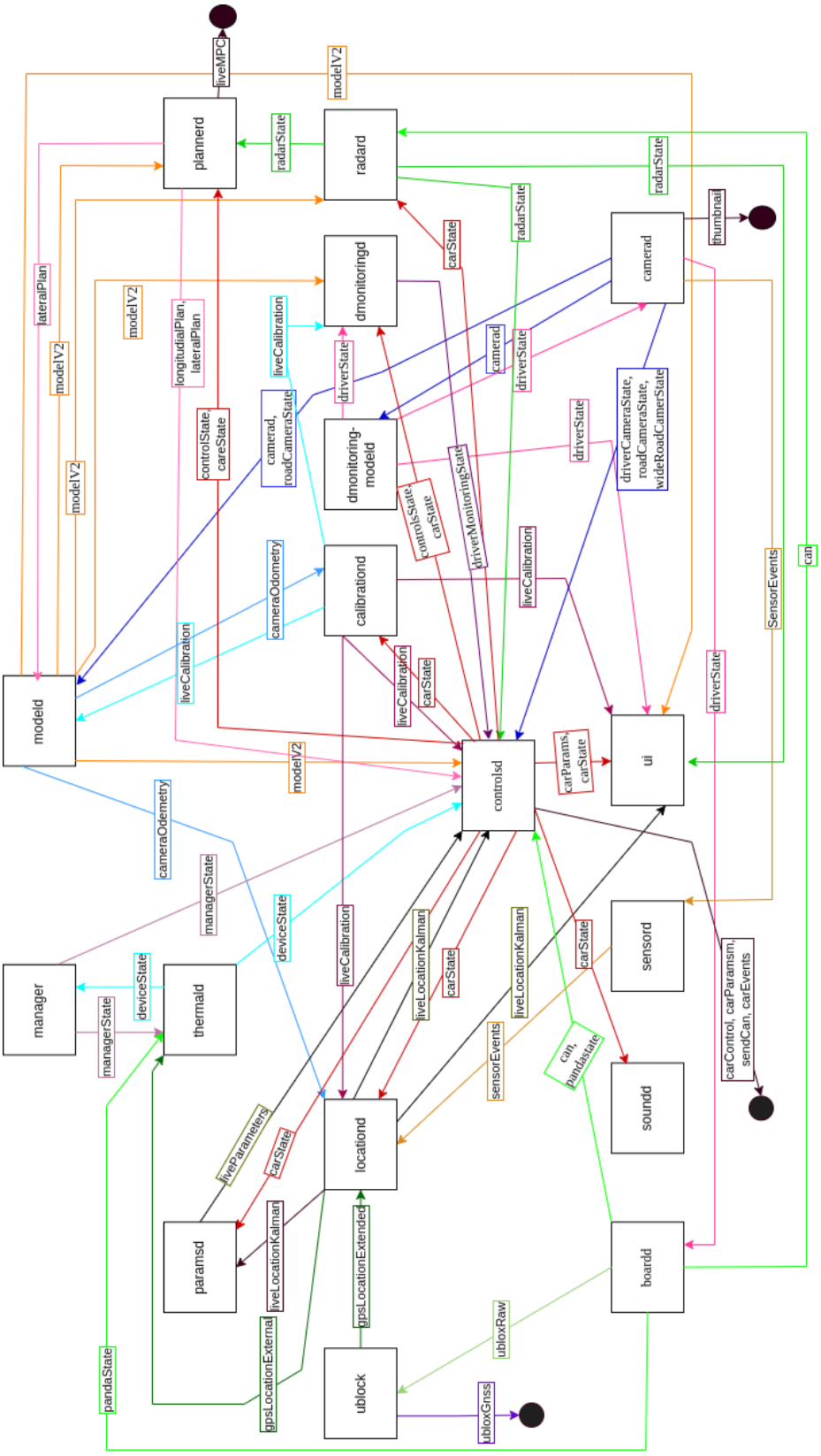

Abbildung 5.6: Detaillierte Control-Structure des Selfdrive Pakets (modelliert aus Informationen von [comc; com21b; Fon21]).

5.3 Blackbox Assessment der Openpilot Software

Eine weitere Möglichkeit zur Evaluierung der Szenarien ist es, die Wahrscheinlichkeiten des Eintretens von Unfällen zu bestimmen, wie es in *"Efficient Black-box Assessment of Autonomous Vehicle Safety"* von Norden et al. [NOS20] der Fall ist. Zunächst wurde hierbei diskutiert, ob eine normale Monte Carlo Wahrscheinlichkeit dazu genutzt werden kann, um die gewünschte Wahrscheinlichkeit zu bestimmen. Dies wurde aber als nicht sinnvoll erachtet, da es sich hier um Ereignisse handelt, die im besten Fall nur ganz selten auftreten, da es hierbei um Unfälle geht. Daher wurde im Paper eine alternative Wahrscheinlichkeit aufgezeigt, die einem *Nonparametric importance-sampling approach* folgt.

Openpilot wurde für die Anwendung des Papers ein wenig abgeändert und daraufhin *TESTPILOT* genannt. Dabei wurde die Hardware auf x86 angepasst, um die Anwendung deterministisch auf dieser ausführen zu können. Durch die x86 Hardware Unterstützung wird die Ausführung in einer verteilten Container-Umgebung möglich. Außerdem werden dadurch Features für die Synchronisierung der Simulationszeit und optional die GPU Beschleunigung verfügbar [NOS20].

TESTPILOT wird dann im CARLA Simulator ausgeführt. Der Fokus wird hierbei auf die Regler gelegt, die die Lenkung und auch die Mechanismen zur Drosselung steuern, sowie deren Zustände. Diese werden dann verfolgt. In *TESTPILOT* wird die dmonitoring Komponente, also die Beobachtung der fahrenden Person nicht betrachtet, da dies im Simulator keine sinnvollen Daten mit sich bringt und auch keine Radardaten mit einbezogen, da diese spezifische Fahrzeugdaten senden. Allerdings dürfen diese Komponenten bei einer Analyse des Gesamtsystems wie bei STPA nicht ausgeschlossen werden, da diese ebenfalls Unfälle begünstigen oder verhindern können [NOS20].

Zunächst wird von Norden et al.[NOS20] nun aufgezeigt, dass der alternative Ansatz besser geeignet ist als der Monte Carlo Ansatz. Genauer wird dies aber hier nicht behandelt, da die optimale Berechnung von Wahrscheinlichkeiten der Loss Scenarios und der Rahmenbedingung dieser Berechnungen hier nicht thematisiert werden kann.

Die Analyse der Fehlerwahrscheinlichkeit und die Failuremodes wurden dann anhand der folgenden Daten evaluiert. Zunächst wurde im CARLA Simulator die Karte *Town_04* verwendet und die Geschwindigkeit, mit der jedes Fahrzeug startet, sowie deren Grundkonfiguration festgelegt. Die Länge der Simulation betrug dabei im Regelfall 10s und wurde nur dann früher beendet, wenn bereits vor dem Ablauf der Zeit eine Kollision aufgetreten ist [NOS20].

Die Fehlerwahrscheinlichkeit hat hierbei die folgenden Resultate ergeben. Dazu muss aber noch definiert werden, dass nicht nur Unfälle, sondern auch gefährliche Situationen als nicht erwünschte Szenarien gewertet werden. Die Ergebnisse der Analyse sind, dass die Failure Rate ergibt, dass in einer von 10^4 Situationen eine solche Situation auftritt. Bei einer Durchschnittsgeschwindigkeit von ca. 72 km/h und der bereits genannten Simulationslänge von 10s bedeutet dies, dass eine gefährliche Situation auf 1250 Meilen (ca. 2.012 km) Fahrstrecke auftritt. Es gibt ebenfalls eine Aussage von Comma AI zu diesem Thema,

die 2018 angegeben haben, dass pro Fahrstunde die fahrende Person einmal eingreifen bzw. das Fahrzeug übernehmen musste [NOS20]. Die offiziellen Angaben von Comma AI lassen sich nicht mit den Zahlen von anderen Herstellern vergleichen, aber mit den Zahlen, die von Norden et al. [NOS20] errechnet wurden. Als Vergleich kann beispielsweise die Aussage von Waymo aus dem Jahr 2018 genommen werden. Bei Waymo handelt es sich um eine Tochterfirma von Alphabet Inc. [Way22], die das *Google self-driving car project* fortführt. Diese gaben an, dass in 1 von 11017 Meilen (ca. 17.730 km) eine Person das Steuer übernehmen muss [Cro18]. Im Vergleich hierzu lässt sich sagen, dass bei Comma AI noch deutlich häufiger eingegriffen werden muss. So muss bei Comma AI zwischen 9- und 10-mal häufiger eingegriffen werden, wie bei Waymo.

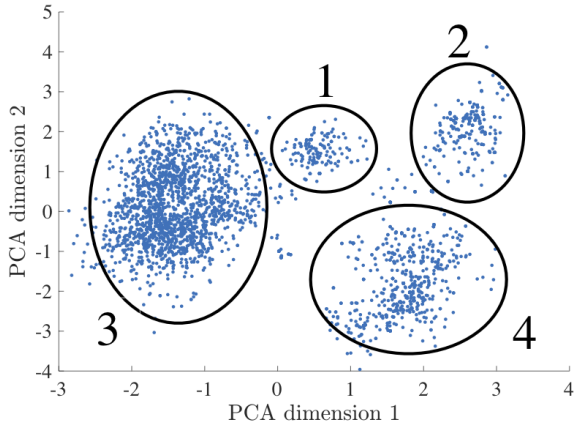

Abbildung 5.7: Erkannte Cluster im Zusammenhand zwischen *PCA dimension 1* (den Wetterdaten – sonnig oder regnerisch) und *PCA dimension 2* (der relativen Geschwindigkeit – langsam oder schnell) [TRU19].

Außerdem werden neben der Fehlerwahrscheinlichkeit auch die Fehlermodi betrachtet. Dieser Teil der Analyse ist nur möglich, da Openpilot Open-Source ist, und es dadurch möglich ist sich auf den Code zu beziehen. In der Grafik (5.7) wurden vier relevante Cluster abgebildet, die häufig für Probleme sorgen und als Fehlerquellen angesehen werden. Dabei werden hierfür zwei Dimensionen festgelegt. Zum einen handelt es sich hierbei, wie in Abbildung 5.7 zu sehen ist, um Dimension 1, die die Wetterdaten und zum anderen um Dimension 2, die die relative Geschwindigkeit abbildet. Dabei wurden vier Cluster identifiziert, die in Abbildung 5.7 dargestellt wurden. Um welche es sich hierbei handelt, wird in Tabelle 5.1 zusammengefasst. Cluster 1 beschreibt somit, dass hier Sonnenschein und eine relativ geringe Geschwindigkeit vorhanden sind, während in Cluster 2 die Geschwindigkeit dieselbe ist, aber hier Regen herrscht. Analog dazu wurden Cluster 3 und 4 mit hoher Geschwindigkeit dargestellt [NOS20].

Die auffallenden Faktoren anhand der Cluster sind, dass große Unsicherheiten bzw. falsche Ergebnisse vor allem dann auftreten, wenn stark von der Sonne geblendet wird oder starker Regen herrscht. Hierdurch können die vorausfahrenden Fahrzeuge oder auch die Spuren falsch identifiziert werden. Es wird dabei ebenfalls angenommen, dass bei geringen

Cluster Nummer	Dimension 1	Dimension 2
1	Sonnenschein	Geringe relative Geschwindigkeit
2	Regen	Geringe relative Geschwindigkeit
3	Sonnenschein	Schnelle relative Geschwindigkeit
4	Regen	Schnelle relative Geschwindigkeit

Tabelle 5.1: Übersicht der Dimensionen der vier Cluster aus Abbildung 5.7 (Die Informationen wurden aus [NOS20] übernommen.).

Geschwindigkeiten keine Kollisionen, sondern nur Streifungen von andern Objekten auftreten und es nur bei hohen Geschwindigkeiten zu Kollisionen kommen wird. Eine weitere Erkenntnis aus den vorliegenden Daten ist, dass der Median aussagt, dass das Auftreten von Failures doppelt so hoch bei Niederschlag ist, wie bei Sonnenschein. Daraus ergeben sich die Möglichkeiten zur Verbesserung. Zum einen sollte an Verbesserungen bezüglich der Erkennung der Wettersituationen gearbeitet werden, zum anderen sollte der Feedbackloop zwischen der Kamera, die die Straße filmt, also der Wahrnehmung der Straße und der Plannerd Komponente bearbeitet werden. Denn diese Schnittstelle ist laut den Autoren Norden et al. [NOS20] anfällig für Fehler.

Die Anwendung von STPA wäre hier möglich, da konkrete Situationen getestet werden können. Dabei sollten Testungen durchgeführt werden, welche Wertekombinationen zu Problemen führen, sowie Tests mit konkreten Daten. Ein Beispiel hierfür sind Tests mit spezifischen Wetterereignissen. Außerdem können diese verbunden werden, um so zu überprüfen, ob das Zusammenspiel von verschiedenen Komponenten zu Problemen führen kann. So wird bei dem Schluss der Autoren [NOS20] deutlich, dass in ihrem Beispiel wieder Plannerd und die Wahrnehmung des vorausfahrenden Fahrzeugs und der Strecke das Problem darstellte. Denn als das Fahrzeug von der Sonne geblendet wurde, wurde die Position des vorausfahrenden Fahrzeugs verschoben und so die Möglichkeit geschaffen, dass Plannerd weiter beschleunigt, da dieser auf den gegebenen Informationen seine Berechnung ausführt. Infolgedessen wurde zu nah auf das vorausfahrende Fahrzeug aufgefahren oder es kam zur Kollision [NOS20].

Umgekehrt ist die Integration in STPA aber nicht nur sinnvoll, um Wahrscheinlichkeiten von konkreten Szenarien zu berechnen und daraus Entscheidungen ableiten zu können, wie mit diesen umgegangen werden muss, sondern auch, um durch Simulationen weitere Loss Scenarios ergänzen zu können. Dabei kommt der Blackbox Ansatz als Testverfahren den KI-Teilen des Systems sehr entgegen, da diese hiermit ebenfalls getestet werden können. Allerdings muss, um ein möglichst realistisches Bild zu erhalten, sichergestellt sein, dass alle relevanten Daten in die Simulation eingefügt werden können. Der größte Vorteil durch die Simulation ist es, dass bereits viele Loss Scenarios getestet und definiert werden können, bevor physische Fahrzeugtests durchgeführt werden. Die Fahrzeugtests gewinnen hierdurch im Optimalfall ebenfalls ein höheres Maß an Sicherheit. Dies muss höchste Priorität haben, da sonst Menschen in Gefahr geraten können.

6 Zusammenfassung und Ausblick

Die ausgewählte Fallstudie Comma AI ist eine Automationssoftware für über 200 Fahrzeugmodelle, die derzeit einen Automatisierungsgrad Level 2 besitzt, auf deren Software Openpilot die Kriterien Open-Source und sicherheitskritische KI-Teile zutreffen. Anschließend wurde eine STPA-Analyse auf dem System ausgeführt. Dabei werden vor allem die Komponenten, die neuronalen Netze beinhalten, betrachtet, sowie die Datenströme, die zu und von den neuronalen Netzen ausgehen.

Danach wurde hiervon abgeleitet, welche Herausforderungen durch die KI-Teile entstanden sind, die STPA nicht umsetzen kann. So können durch Modifizierungen, die durch die Veränderungen des Datensatzes während des Trainings geschehen, neue Gefahren entstehen. Des Weiteren ist die Risikoabschätzung der Loss Scenarios, durch den Blackbox Charakter des neuronalen Netzes schwierig. Außerdem müssten für eine vollständige Analyse der neuronalen Netzkomponenten, alle eingehenden Neuronen betrachtet und somit STPA auf Codeebene heruntergebrochen werden. Dies setzt selbstverständlich ein sehr großes Detaillevel voraus und auch ein großes Level an Komplexität. Die Vollständigkeit der möglichen Loss Scenarios wird durch dieselben Gründe sehr schwer zu überblicken. Die Constraints, die aus den Loss Scenarios abzuleiten sind, sind im iterativen Prozess sehr schwer zu identifizieren, da ein neuronales Netz unterschiedliche Ergebnisse ausgeben kann.

Anschließend wurden verschiedene Ansätze aufgezeigt, wie STPA erweitert werden kann. Dabei wurde zunächst die Möglichkeit aufgezeigt, STPA mit der ISO 26262 und der ISO 21448 zusammenzufügen. Als Nächstes wurde ein Monitoringsystem vorgestellt, dass aus den Loss Scenarios abgeleitet wird. Als Letztes wurde ein Blackbox-System aufgezeigt, mit dem Openpilot getestet wurde und mit deren Hilfe Wahrscheinlichkeiten für gefährliche Situationen und Unfällen bestimmt werden sollen.

Ausblick

Auf diesen Möglichkeiten kann nun in zukünftigen Arbeiten aufgebaut und evaluiert werden, inwieweit die Möglichkeiten zur Safety beigetragen haben und deren Mehrwert dann aufzeigen. Zunächst wurde beim Blackbox Assessment von Norden et al. [NOS20] eine bestimmte Wahrscheinlichkeit eingesetzt, um die selten eintretenden Ereignisse zu bestimmen. Hier könnten verschiedene Ansätze verglichen werden, welche die optimale Wahrscheinlichkeitsberechnung darstellt und ob es hierbei verschiedene Wahlmöglichkeiten

abhängig von gewissen Grundvoraussetzungen gibt. Es gibt eine weitere Möglichkeit, wie das Ziel erreicht werden kann, das nicht direkt mit STPA in Verbindung steht. So muss ebenfalls die Erweiterung der ISO 26262 in Betracht gezogen werden. Doch hier wird derzeit diskutiert, ob es möglich ist einen Standard zu haben, der sowohl die traditionellen Systeme für die funktionale Sicherheit gewährleistet, als auch für die KI-Systeme oder ob hierfür ein eigener Standard benötigt wird. Behandelt wird dieses Thema derzeit unter anderem in den Arbeiten von Salay und Czarnecki [SC18], Henriksson et al. [HBE18] und dem ISO und IEC [ISO21]. Hierbei ist erkennbar, dass in diesem Themenfeld derzeit intensiv geforscht wird, da die Paper sehr aktuell, also von 2018 und später, sind. Derzeit sind noch keine fertigen Ergebnisse vorhanden, wie die Lösung für Safety in Kombination mit KI-Systemen aussehen kann und wie der optimale Ansatz aussehen könnte.

Literaturverzeichnis

[ACA+22] E. Acar Celik, C. Cârlan, A. Abdulkhaleq, F. Bauer, M. Schels, H. J. Putzer. „Application of STPA for the Elicitation of Safety Requirements for a Machine Learning-Based Perception Component in Automotive“. In: *Computer Safety, Reliability, and Security*. Hrsg. von M. Trapp, F. Saglietti, M. Spisländer, F. Bitsch. Cham: Springer International Publishing, 2022, S. 319–332. ISBN: 978-3-031-14835-4 (zitiert auf S. 63–66).

[Ade22] S. Adeeb. *comma_two_master - RELEASES.md*. Apr. 2022. URL: https://github.com/commaai/openpilot/blob/commatwo_master/RELEASES.md (zitiert auf S. 35).

[Aut22] Autoware, AI. *Autoware Github Repository*. 2022. URL: https://github.com/autowarefoundation/autoware (zitiert auf S. 32, 34).

[BBY20] C. Becker, J. C. Brewer, L. Yount. *Safety of the intended functionality of lane-centering and lane-changing maneuvers of a generic level 3 highway chauffeur system*. Techn. Ber. United States. National Highway Traffic Safety Administration. Electronic System Safety Research Division, 2020 (zitiert auf S. 64).

[BHS+22] M. Borg, J. Henriksson, K. Socha, O. Lennartsson, E. S. Lönegren, T. Bui, P. Tomaszewski, S. R. Sathyamoorthy, S. Brink, M. H. Moghadam. „Ergo, SMIRK is Safe: A Safety Case for a Machine Learning Component in a Pedestrian Automatic Emergency Brake System“. In: *arXiv preprint arXiv:2204.07874* (2022) (zitiert auf S. 63, 64).

[Big18] P. Bigelow. *After a Pause, Comma.ai Delivers a Driver-Assist System That Rivals Super Cruise and Autopilot*. Juli 2018. URL: https://www.caranddriver.com/news/a22480267/after-a-pause-commaai-delivers-a-driver-assist-system-that-rivals-super-cruise-and-autopilot/ (zitiert auf S. 33).

[Clo22] C. Clough (last editor). *comma two*. Sep. 2022. URL: https://github.com/commaai/openpilot/wiki/comma-two (zitiert auf S. 35).

[Com] Comma AI. *Comma Connect - Demo*. URL: https://connect.comma.ai/ (zitiert auf S. 40).

[coma] comma.ai. *Github Comma.Ai - comma10k*. URL: https://github.com/commaai/comma10k (zitiert auf S. 36).

[comb] comma.ai. *Github CommaAi*. URL: https://github.com/commaai (zitiert auf S. 37, 41).

[comc] comma.ai. *Github Openpilot*. URL: https://github.com/commaai/openpilot (zitiert auf S. 31, 32, 34, 38–40, 42–45, 48, 72).

[com16] @comma_ai. *Absage von Comma One*. Okt. 2016. URL: https://twitter.com/comma_ai/status/791958413345382400 (zitiert auf S. 33).

[com17] comma.ai. *Our Road to Self Driving Victory*. Juni 2017. URL: https://blog.comma.ai/our-road-to-self-driving-victory/ (zitiert auf S. 33).

[Com21] Comma AI. *comma three Press Release*. Juli 2021. URL: https://blog.comma.ai/comma-three-press-release/ (zitiert auf S. 35, 36).

[com21a] comma.ai. *Comma Ai Shop*. 2021. URL: https://comma.ai/shop/products/three (zitiert auf S. 35, 36, 38).

[com21b] comma.ai. *How openpilot works in 2021*. Okt. 2021. URL: https://blog.comma.ai/openpilot-in-2021/ (zitiert auf S. 37, 39–43, 45, 48, 72).

[Com22] Comma AI. *comma.ai blog*. 2022. URL: https://blog.comma.ai/ (zitiert auf S. 32).

[Cro18] S. Crowe. *Waymo autonomous vehicles leave apple in the dust*. Feb. 2018. URL: https://www.therobotreport.com/waymo-autonomous-vehicles-apple/ (zitiert auf S. 74).

[CWCF22] F. Camara, C. Waltham, G. Churchill, C. Fox. *OpenPodcar: an Open Source Vehicle for Self-Driving Car Research*. 2022. URL: https://github.com/OpenPodcar/OpenPodcar (zitiert auf S. 33, 34).

[ESB20] H. Ernst, J. Schmidt, G. Beneken. *Grundkurs Informatik: Grundlagen und Konzepte für die erfolgreiche IT-Praxis – Eine umfassende, praxisorientierte Einführung*. Jan. 2020. ISBN: 978-3-658-30330-3. DOI: 10.1007/978-3-658-30331-0 (zitiert auf S. 16).

[Eth16] D. Etherington. *Comma.ai cancels the Comma One following NHTSA letter*. Okt. 2016. URL: https://techcrunch.com/2016/10/28/comma-ai-cancels-the-comma-one-following-nhtsa-letter/ (zitiert auf S. 33).

[Fon21] F. Fontana. *Thesis - Self-driving cars and Openpilot : a complete overview of the framework*. Dez. 2021. URL: https://www.politesi.polimi.it/handle/10589/181889 (zitiert auf S. 39–42, 45, 72).

[GBW+22] S. Gautham, G. Bakirtzis, A. Will, A. V. Jayakumar, C. R. Elks. „STPA-Driven Multilevel Runtime Monitoring for In-Time Hazard Detection“. In: *Computer Safety, Reliability, and Security*. Hrsg. von M. Trapp, F. Saglietti, M. Spisländer, F. Bitsch. Cham: Springer International Publishing, 2022, S. 158–172. ISBN: 978-3-031-14835-4 (zitiert auf S. 68, 69).

[HBE18] J. Henriksson, M. Borg, C. Englund. „Automotive Safety and Machine Learning: Initial Results from a Study on How to Adapt the ISO 26262 Safety Standard“. In: *2018 IEEE/ACM 1st International Workshop on Software Engineering for AI in Autonomous Systems (SEFAIAS)*. 2018, S. 47–49 (zitiert auf S. 78).

[Ins] Institute for Systems Theory and Automatic Control - Group of Frank Allgöwer. *Model Predictive Control*. URL: https://www.ist.uni-stuttgart.de/research/group-of-frank-allgoewer/model-predictive-control/ (zitiert auf S. 41).

[ISO21] ISO/IEC. *ISO/IEC TR 24029-1:2021(en) Artificial Intelligence (AI) — Assessment of the robustness of neural networks*. 2021. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:tr:24029:-1:ed-1:v1:en (zitiert auf S. 61, 78).

[KG19] O. Kirovskii, V. Gorelov. „Driver assistance systems: analysis, tests and the safety case. ISO 26262 and ISO PAS 21448“. In: *IOP Conference Series: Materials Science and Engineering*. Bd. 534. 1. IOP Publishing. 2019, S. 012019 (zitiert auf S. 64).

[Kik21] G. Kikelj. *Comma3*. Aug. 2021. URL: https://github.com/commaai/openpilot/wiki/comma-three (zitiert auf S. 40).

[KS19] M. Kirste, M. Schürholz. „Einleitung: Entwicklungswege zur KI“. In: *Künstliche Intelligenz: Technologie | Anwendung | Gesellschaft*. Hrsg. von V. Wittpahl. Springer Berlin Heidelberg, 2019, S. 21–35. ISBN: 978-3-662-58042-4. DOI: 10.1007/978-3-662-58042-4_1 (zitiert auf S. 15–18).

[Küt22] S. Kütük. *Past, Present and the Future of Autoware*. Sep. 2022. URL: https://www.autoware.org/post/past-present-and-the-future-of-autoware (zitiert auf S. 32).

[Lev11] N. G. Leveson. *Engineering a Safer World: Systems thinking applied to safety.* MIT Press, Cambridge, 2011 (zitiert auf S. 19–21).

[LT18] N. G. Leveson, J. P. Thomas. *STPA Handbook*. März 2018. URL: https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf (zitiert auf S. 19, 22–27).

[NOS20] J. Norden, M. O'Kelly, A. Sinha. *Efficient Black-box Assessment of Autonomous Vehicle Safety*. Juni 2020. DOI: 10.48550/ARXIV.1912.03618. URL: https://arxiv.org/abs/1912.03618 (zitiert auf S. 73–75, 77).

[Ong17] P. Ongsulee. „Artificial intelligence, machine learning and deep learning“. In: *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)*. 2017, S. 1–6. DOI: 10.1109/ICTKE.2017.8259629 (zitiert auf S. 16, 17).

[PEG20] PEGASUS Projektpartner. *Schlussbericht für das Gesamtprojekt - PEGASUS (Projekt zur Etablierung von generell akzeptierten Gütekriterien, Werkzeugen und Methoden sowie Szenarien und Situationen zur Freigabe hochautomatisierter Fahrfunktionen) - gefördert durch das Bundesministeriums für Wirtschaft und Energie*. Feb. 2020. URL: https://www.pegasusprojekt.de/files/tmpl/pdf/PEGASUS_Abschlussbericht_Gesamtprojekt.PDF (zitiert auf S. 13).

[POs] E. Panagiotaki, F. OSullivan, sgvandijk. *GitHub Aslan*. URL: https://github.com/project-aslan/Aslan (zitiert auf S. 32, 34).

[Pro20] Project Aslan. *Aslan Autonomy*. 2020. URL: www.project-aslan.org (zitiert auf S. 33, 34).

[Ran21] M. Rangaiah. *Artificial Intelligence in Healthcare: Applications and Threats*. Juni 2021. URL: https://www.analyticssteps.com/blogs/artificial-intelligence-healthcare-applications-and-threats (zitiert auf S. 31).

[RICK22] M. Rahman, M. R. Islam, M. Chowdhury, T. Khan. „A Physics-Based Longitudinal Driver Model for Automated Vehicles“. In: *IEEE Access* 10 (2022), S. 80883–80899. DOI: 10.1109/ACCESS.2022.3193997 (zitiert auf S. 35).

[SAE14] SAE On-Road Automated Vehicle Standards Committee. *Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems*. 2014. URL: https://www.sae.org/standards/content/j3016_202104/ (zitiert auf S. 28–30).

[SC18] R. Salay, K. Czarnecki. *Using Machine Learning Safely in Automotive Software: An Assessment and Adaption of Software Process Requirements in ISO 26262*. Aug. 2018. DOI: https://doi.org/10.48550/arXiv.1808.01614 (zitiert auf S. 78).

[SLZ21] L. Sun, Y.-F. Li, E. Zio. „Comparison of the HAZOP, FMEA, FRAM, and STPA Methods for the Hazard Analysis of Automatic Emergency Brake Systems“. In: *ASCE-ASME J Risk and Uncert in Engrg Sys Part B Mech Engrg* 8.3 (Okt. 2021). 031104. DOI: 10.1115/1.4051940 (zitiert auf S. 22).

[SSHB18] H. Schafer, E. Santana, A. Haden, R. Biasini. *A Commute in Data: The comma2k19 Dataset*. 2018. DOI: https://doi.org/10.48550/arXiv.1812.05752 (zitiert auf S. 35).

[TEG22] TEGARA Co. Ltd. *[Release information] New lineup of driving support system device comma "comma three dev kit"*. Apr. 2022. URL: https://www.tegakari.net/en/ (zitiert auf S. 35, 36).

[The21] The Autoware Foundation. *Projekt Autoware - Autoware Roadmap*. 2021. URL: https://www.autoware.org/autoware (zitiert auf S. 32, 34).

[The22] Theator Inc. *Theator's Surgical Intelligence Platform*. 2022. URL: https://theator.io/surgical-intelligence-platform/ (zitiert auf S. 31).

[TRU19] TRUSTWORTHY AI TEAM. *Risk, simulation, and the road to trustworthy autonomous vehicles*. Dez. 2019. URL: https://trustworthy.ai/2019/12/13/risk-simulation-and-the-road.html (zitiert auf S. 74).

[VB15] A. Vance, B. Bloomberg. *The First Person to Hack the iPhone Built a Self-Driving Car. In His Garage - George Hotz is taking on Google and Tesla by himself*. Dez. 2015. URL: https://www.bloomberg.com/features/2015-george-hotz-self-driving-car/ (zitiert auf S. 33).

[VDA15] VDA - Verband der Automobilindustrie e.V. *Automatisierung - Von Fahrerassistenzsystemen zum automatisierten Fahren*. Sep. 2015. URL: http://docplayer.org/8522457-Automatisierung-von-fahrerassistenzsystemen-zum-automatisierten-fahren.html (zitiert auf S. 28–30).

[Vic22] Vicarious Surgical. *Vicarious Surgical Robotic System*. 2022. URL: https://www.vicarioussurgical.com/ (zitiert auf S. 31).

[Way22] Waymo-LLC. *Waymo Story*. 2022. URL: https://waymo.com/company/#story (zitiert auf S. 74).

Alle URLs wurden zuletzt am 26. 11. 2022 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift