Technische Hochschule Wildau
Technical University of Applied Sciences

Masterarbeit

zur Erlangung des akademischen Grades Master

Technische Hochschule Wildau

Fachbereich Ingenieur- und Naturwissenschaften

Studiengang Telematik (M.Eng.)

Thema (deutsch): Entwicklung einer Anwendersoftware, die Fahrdaten und Videomaterial der Fahrsimulation BeamNG.tech verbindet und wiederholbare Szenarien anbietet

Thema (englisch): Development of a user-oriented software that fuses driving and video data supplied by the driving simulation BeamNG.tech, while providing repeatable scenarios

Autor/in:	Maximilian Hartmann
Seminargruppe:	TM/21
Betreuer/in:	Prof. Dr. rer. nat. Alexander Kleinsorge
Zweitgutachter/in:	Prof. Dr. Alexander Stolpmann
Spätestmögliche 6Ab-gabe:	22.02.2025

Masterarbeit

Antrag vom: 16.09.2024

Name:	Maximilian Hartmann		
Studiengang:	Telematik	Matrikel-Nr.:	50054798
		Seminargruppe:	TM/21
Betreuende/r Hochschuldozent/in:	Prof. Dr. rer. nat. Alexander Kleinsorge	Beginn der Arbeit:	22.09.2024
Zweitgutachter/in:	Prof. Dr. Alexander Stolpmann	Abgabetermin:	22.02.2025
Themensteller (z.B. Betrieb, Institution, Behörde):	Technische Hochschule Wildau	Fachliche Betreuungsperson des Themenstellers:	Alexander Kleinsorge
		Straße:	Hochschulring 1
		PLZ, Ort:	15745 Wildau

Kurzthema

Entwicklung einer Anwendersoftware, die Fahrdaten und Videomaterial der Fahrsimulation BeamNG.tech verbindet und wiederholbare Szenarien anbietet

Kurzthema in Englisch

Development of a user-oriented software that fuses driving and video data supplied by the driving simulation BeamNG.tech, while providing repeatable scenarios

Zielstellung

Ziel der Arbeit ist der Entwurf und die Entwicklung einer Schnittstellen Software, die die Daten der Fahrsimulation BeamNG.tech für Studenten zugänglich machen soll, damit diese ihre selbst entwickelten Fahrassistenz Programme gefahrenlos testen können

Inhaltliche Anforderungen / Teilaufgaben

- Entwicklung der Schnittstellensoftwarelogik
- Entwicklung einer grafischen und textuellen Oberfläche für diese Schnittstellen Software
- Vergleichen von Videokompressionsverfahren zur effizienten Verarbeitung von Videodaten
- Entwicklung eines aktiven Spurhallteassistenten, welcher die entwickelte Schnittstellensoftware nutzt, um dessen Funktionalität zu verifizieren

Sprache der Arbeit

Deutsch

Konsultationen erfolgen nach Vereinbarung mit dem betreuenden Hochschullehrer.

Prof. Dr. rer. nat. Alexander (Hochschuldozent/in)	Prof. Dr. Alexander Stolpmann (Zweitgutachter/in)	Maximilian Hartmann (Student/in)	*genehmigt* Kleinsorge (Prüfungsausschuss)

Sperrvermerk ☐ Ja ☒ Nein

Bibliographische Beschreibung

Autor: Maximilian Hartmann

Titel: Entwicklung einer Anwendersoftware, die Fahrdaten und Videomaterial der Fahrsimulation BeamNG.tech verbindet und wiederholbare Szenarien anbietet

Metadaten: Masterarbeit 2024/2025, Technische Hochschule Wildau, 79 Seiten, 28 Abbildungen, 7 Tabellen; Anhang: 3 Tabellen

Ziele:

Ziel dieser Arbeit ist es, eine Schnittstelle zu einer Fahrsimulationssoftware als unterstützendes Werkzeug zur Schulung von Studierenden zu entwickeln. Diese soll es den Anwendenden ermöglichen, ihre Bildverarbeitungsalgorithmen in einem realistischen Umfeld risikofrei und kostengünstig zu testen. Hierfür wird auf die Simulationssoftware „BeamNG.tech" der BeamNG GmbH aus Hamburg zurückgegriffen. Zusätzlich wird ein Spurhalte Assistent als Funktionsbeweis entwickelt, der die implementierte Schnittstellen Software nutzt.

Inhalt:

Eingangs wird eine kurze Einführung in die Funktionsweise der verwendeten Fahrsimulationssoftware gegeben. Daraufhin erfolgt eine Darstellung der zuvor benutzten Daten und die Vorstellung zweier zuvor erstellten Prototypen.
Anschließend wird eine Schnittstellensoftware konzipiert und umgesetzt, die die Fahrsimulation BeamNG.tech zur Erzeugung und Weitergabe von Fahrdaten und Bildern nutzt. Dazu wird ein Spurhalteassistent entworfen und entwickelt, der die Funktionalität der zuvor umgesetzten Schnittstellensoftware in einer konkreten Anwendung validiert.

Bibliographic Description

Author: Maximilian Hartmann

Title: Development of user-oriented software that fuses driving and video data supplied by the driving simulation

BeamNG.tech, while providing repeatable scenarios

Metadata: Master's Thesis 2024/2025, Technische Hochschule Wildau, 79 Pages, 28 Figures, 7 Tables; Appendix: 3 Tables

Goal:

The aim of this thesis is to develop an interface for driving simulation software as a supportive tool for training students. This interface will enable users to test their image processing algorithms in a realistic environment without risk and at a low cost. For this purpose, the simulation software "BeamNG.tech" by BeamNG GmbH from Hamburg will be used. Additionally, a lane-keeping assistant will be developed as proof of concept, utilizing the implemented interface software.

Abstract:

Initially, a brief introduction to the functionality of the driving simulation software used is provided. Following this, there is a presentation of the previously used testing data and the introduction of two previously created prototypes. Subsequently, an interface software is designed and implemented, which uses the driving simulation BeamNG.tech to generate and transmit driving data and images. For validating the functionality of the previously implemented interface software, a lane keep assist is designed and developed.

Hinweise zum Lesen der Arbeit

In dieser Arbeit wurden Abkürzungen verwendet. Diese sind mit ihren Bedeutungen in einer Tabelle auf der Seite „IX“ zu finden und im Fließtext durch eine *kursive Formatierung* markiert. Außerdem wurden in der Arbeit Fachbegriffe und Abkürzungen verwendet, die in einem Glossar auf der Seite „XI“ erklärt wurden. Im Fließtext sind diese Fachbegriffe durch eine anhängende und kleingestellte Zahl in eckigen Klammern gekennzeichnet (Beispiel Open Source[1]). Quellen sind immer am Ende eines Paragrafen angegeben und können in größerem Detail auf der Seite 64 gefunden werden.

Danksagung

Ich möchte mich besonders bei Herrn Professor Dr. rer. nat. Alexander Kleinsorge für sowohl die Betreuung und hilfreichen Rat während der Masterarbeit als auch die Möglichkeit, dieses interessante Thema bearbeiten zu können bedanken.

Mein Dank gilt zudem der BeamNG GmbH, welche dieses Projekt mit dem Zugang zur Simulationssoftware BeamNG.tech ermöglicht hat und mir bei Herausforderungen stets zeitnah weitergeholfen hat.

Ein besonders großes Dankeschön geht an meine Eltern, Iris und Marcus Hartmann, für ihre stets bedingungslose Unterstützung und Motivation während meines Studiums.

Auch Julia Reinke und Frieder Keilholz möchte ich für ihre Geduld und Ratschläge während dieser Zeit danken.

Viele Dank!

Maximilian Hartmann

Berlin, den 22. Februar 2025

Inhaltsverzeichnis

Abkürzungsverzeichnis

Kürzel	*Bedeutung*
ABS	Antiblockiersystem – Verhindert bei einem starken Bremsvorgang das Blockieren der Räder und erhält dadurch die Lenkfähigkeit eines Fahrzeuges. (Stefan & Andrea, 2015)
ASR	Antischlupfregelung – Verhindert das Durchdrehen der Antriebsräder durch Verringerung der Motorkraft oder abbremsen der betroffenen Räder. (Stefan & Andrea, 2015)
ESP	Elektronisches Stabilitätsprogramm – Im Englischen auch ESC/Electronic Stability Control genannt. Verhindert Schleuderbewegungen des Fahrzeuges durch den Einsatz verschiedener Assistenzsysteme. (Stefan & Andrea, 2015)
VESA	Video Electronics Standards Association – Eine Organisation, welches ich mit der Erstellung von Standards für Computer- Video und Grafik beschäftigt.
LDA	Lane Departure Assist – Englisch für Spurhalteassistent. Warnt den Fahrer eines Fahrzeuges davor, dass er die aktuelle Spur ungewollt verlässt. Einige aktive Spurhalteassistenten lenken das Fahrzeug selbstständig zurück in die Fahrspur.
CSV	Comma Separated Values – ist ein Dateiformat zur Speicherung von Tabellendaten. Die Daten werden dabei durch Trennzeichen, wie Kommas oder Semikolons, voneinander getrennt.
JSON	JavaScript Object Notation – ist ein textbasiertes Datenformat, das zum Austausch zwischen Anwendungen dient. JSON ist von Programmiersprachen unabhängig.
CV2	Open-Source Computer Vision Library (2. Version) – CV2 ist eine leistungsfähige Bibliothek für die Arbeit mit Bildern in Python.
.jpg/.jpeg	Joint Photographic Experts Group - ist ein Dateiformat für digitale Bilder. .jpg-Dateien werden bei der Speicherung automatisch komprimiert, um die Dateigröße zu reduzieren.
XML	XML (Extensible Markup Language) - ist eine universelle Auszeichnungssprache, die verwendet wird, um strukturierte Daten darzustellen und zu speichern. Es handelt sich um eine textbasierte Sprache, die es ermöglicht, Informationen in einer hierarchischen Form zu organisieren

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

ID	Bezeichnung	Erklärung
[1]	Open Source	Open Source, englisch für „Offene Quelle“, beschreibt Programme, dessen Quellcode öffentlich sichtbar gemacht wurde und die, unter bestimmten Bedingungen, auch Modifikationen an ihrem Quellcode zulassen.
[2]	Canny-Algorithmus	Der Canny Algorithmus ist ein Algorithmus zur Detektion von Kantenbildern. Er besteht aus einer Bildglättung, einer Kantendetektion mit Berechnung von Betrag und Richtung der Kanten, einer Kantenverdünnung durch Nicht-Maximum-Unterdrückung sowie einer Schwellwertoperation die die Unwichtigen Kanten herausfiltert.
[3]	Hough-Linien-Transformation	Die Hough-Linien-Transformation ist ein Algorithmus, der versucht in einem Binär-Bild (schwarz-weiß ohne Graustufen) Linien zu erkennen und deren Steigung und Achsenabschnitt zurückzugeben.
[4]	Python Streaming Socket	Ein Socket bezeichnet eine Schnittstelle, die der Kommunikation zwischen verschiedenen Prozessen dient. Ein Python Streaming Socket stellt dabei einen spezifischen Typ eines Netzwerk-Sockets dar, dessen Funktionalität in der durchgängigen, bidirektionalen Datenübertragung zwischen Prozessen (Anwendungen) besteht. Im Vergleich zu herkömmlichen Sockets zeichnet sich dieser durch die konstante Aufrechterhaltung der Verbindung aus. (Jennings, 2024)
[5]	Alpha	Der Begriff Alpha, im Kontext von Software, beschreibt ein Programm im führen Entwicklungsstadium.
[6]	Gierwinkel	Der Gierwinkel ist ein Lagewinkel und beschreibt die Rotation eines Objektes auf der Z-Achse.

1 Einleitung

1.1 Idee

Die Idee eine Fahrsimulationssoftware als unterstützendes Werkzeug zur Schulung von Studierenden zu nutzen, entstand erstmalig während des Kurses „Bildverarbeitung im Automobil“ im Sommersemester 2022 des Studiengangs „Telematik Master“ an der Technischen Hochschule Wildau. Dort wurde den Studierenden unter anderem gelehrt, wie sie Spuren, Fluchtpunkte und Verkehrsschilder mit Hilfe eigens geschriebener Software erkennen können. Diese neu erworbenen Kenntnisse sollten in einem Semesterabschlussprojekt genutzt werden, um eine Bibliothek an Fahrassistenzsystemen zu erstellen.

Da zu dieser Zeit nur mit voraufgezeichneten Bildern und Videos getestet werden konnte fiel auf, dass nur passive Assistenzsysteme, wie eine Verkehrsschilderkennung, programmiert werden konnten. Ebenfalls konnten Qualitätsmerkmale, wie Latenz und Konsistenz der Ergebnisse nur äußerst eingeschränkt nachvollzogen werden.

Mit einer Simulation als Werkzeug können viele verschiedene Situationen kostengünstig und wiederholbar dargestellt werden. Zusätzlich können Daten zu diesen Situationen in Echtzeit produziert werden, sodass sie auch nur auf diese Weise ausgewertet werden müssen. Ebenfalls verleihen Fahrdaten wie Geschwindigkeit und Lenkwinkel den Bilddaten wichtigen Kontext, welcher bei voraufgenommenen Bildern oder Videos meist fehlt.

Um diese Vorteile nutzen zu können, wurde das Semesterabschlussprojekt um eine Software erweitert, die zunächst Bilder und Fahrdaten aus einer Simulation aufzeichnen konnte. Aufgrund der limitierten Zeitressourcen, die durch die Struktur eines Semesters vorgegeben sind, gelang zwar die Entwicklung eines funktionsfähigen Prototyps, jedoch führte die umständliche Handhabung zu keiner weiteren Nutzung. Die Konzeption dieser Arbeit zielt darauf ab, eine Software mit erweiterten Funktionalitäten zu kreieren, die eine leichte Handhabung ermöglicht, ohne dass hierfür spezifisches Fachwissen erforderlich ist.

1.2 Ziel

Der Fokus dieser Arbeit liegt auf der Entwicklung einer softwarebasierten Plattform, die es Studierenden ermöglicht, ihre Bildverarbeitungsalgorithmen in einem realistischen Umfeld risikofrei und kostengünstig zu testen. Dabei soll es den Anwendenden der Software erspart werden, komplexe Dokumentationen zu studieren, da der Lernfokus auf der Bildverarbeitung und nicht auf der Verwendung der Fahrsimulation liegen soll. Als Nachweis über die Erfüllung der an die Software gestellten Anforderungen, soll ein aktiver Spurhalteassistent umgesetzt werden, welcher über die Schnittstellensoftware mit der Simulation interagiert.

Da bei der Aufnahme von Videodateien große Datenmengen erzeugt werden, müssen diese mit Hilfe von Videocodecs komprimiert werden. Um die Effektivität dieser einschätzen zu können, werden die Ergebnisse einer Masterarbeit, die sich mit dem Thema befasst, zusammengefasst.

1.3 Abgrenzung

In der vorliegenden Arbeit erfolgt keine Gegenüberstellung von Simulationsprogrammen. Zudem wird die Simulationsgenauigkeit der Software "BeamNG.tech" nicht evaluiert. Eine Auseinandersetzung mit potenzieller Testhardware sowie die Erhebung von Daten an realen Fahrzeugen finden ebenfalls nicht statt. Ein praktischer Test und Vergleich der verschiedenen Videocodes erfolgt nicht.

1.4 Aufbau der Arbeit

Die Arbeit beginnt mit den technischen Grundlagen. Diese sollen eine Wissengrundlage für in der Arbeit genutzte Technologien und Software vermitteln. Des Weiteren soll der Ist-Zustand den aktuellen Fortschritt des Projekts aus 2022 beschreiben und die Mittel des aktuellen Kurses „Bildverarbeitung im Automobil" (Stand 2024) darlegen. Anschließend werden die Anforderungen an Schnittstellen- und Spurhalteassistenzsoftware aufgelistet und erläutert. Daraufhin wird ein Konzept zur Planung und Umsetzung der entsprechenden Anforderungen erstellt. Für die Schnittstellensoftware wird dabei die Softwarestruktur und das Bedienkonzept separiert. Anschließend befasst sich das Kapitel der Umsetzung mit der Realisierung des Konzepts. Schließlich folgt die Umsetzung des

Spurhalteassistenten und eine Einordnung der Funktionalität dessen. In einem abschließenden Fazit werden die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick in mögliche zukünftige Erweiterungen gegeben.

1.5 Videocodecs H.265 und AV1

Bezüglich des Vergleichs verschiedener Videokompressionsverfahren wird auf die Masterarbeit „Datenreduktion mit dem AV1 – Videocodec im Vergleich zu H.265“, Christoph Grubits, Wien, 31. Januar 2021, Bezug genommen. Dort erfolgte eine Gegenüberstellung der Verfahren H.265 und AV1. Der Autor kommt dabei zu dem Ergebnis, dass das AV1 – Verfahren insbesondere bei hohen Kompressionsraten ein artefaktfreies Bild erzeugt und daher dem H.265 – Verfahren bei entsprechenden Anwendungen vorzuziehen ist. (Grubits, 2021)

Abbildung 1 Vergleich von Video-Kompressionsverfahren anhand eines Bildausschnittes. Links Original, Mitte AV1 komprimiert 2Mb/s, rechts H.265 2Mb/s | Bildervergleich aus der Masterarbeit Christoph Grubits. Seite 94

In dem Vergleich aus Grubits Arbeit (Abbildung 1) können die entstandenen Artefakte an der rechten Hand der linken Person und an deren Schuhen erkannt werden. Diese Bildfehler können im Kontext der Bildverarbeitung im Automobil die Genauigkeit bei der Erkennung von Merkmalen verringern.

2 Technische Grundlage BeamNG.tech

BeamNG.tech/BeamNG.drive ist eine Fahrsimulationssoftware, die seit über 15 Jahren von der BeamNG GmbH aus Hamburg entwickelt wird. BeamNG.drive fokussiert sich dabei mehr auf den Videospiel-Markt und wird daher auch über Marktplätze, z.B. wie Steam vertrieben. BeamNG.tech, vormals: BeamNG.research, ist eine Erweiterung dieses Videospiels, welche Anpassungen für die Nutzung in Bildungs- und Industrieumgebungen beinhaltet.

Abbildung 2 Logo der BeamNG GmbH

Anders als standardmäßig in Rennspielen/Simulationen wird der gesamte Antriebsstrang vom Motor zum Reifen physikalisch berechnet. (Abbildung 3) Beispielsweise wird bei einem simulierten Eingreifen des *ABS* von einem simulierten *ABS*-Steuergerät erkannt, dass die Reifen blockieren. Folglich wird die Bremsstärke individuell für jedes Rad so angepasst, dass die Lenkfähigkeit erhalten bleibt. In einem „gewöhnlichen" Rennspiel, welches für Gelegenheitsspieler ausgelegt wurde, würde lediglich abgefragt werden: *„Frage: Würden bei dieser Bremsung die Räder blockieren? Antwort: Ja, dann kann das Fahrzeug, solange kein ABS verbaut wurde, nicht mehr lenken."* Durch die umfassende Simulation, die durch die BeamNG Physik-Engine durchgeführt wird, können auch komplexe Assistenz-Programme, wie z.B. bremsenbasierte Drehmomentverteilung (auf Englisch auch bekannt unter: „brake based tourge vectoring"), akkurat und mit Einbeziehung weiterer Variablen wie Bremstemperatur, simuliert werden. Für diese Arbeit ist diese genaue Widerspiegelung eines echten Fahrzeuges ein Grund dafür, dass BeamNG.tech als Kerntechnologie gewählt wurde. (BeamNG GmbH)

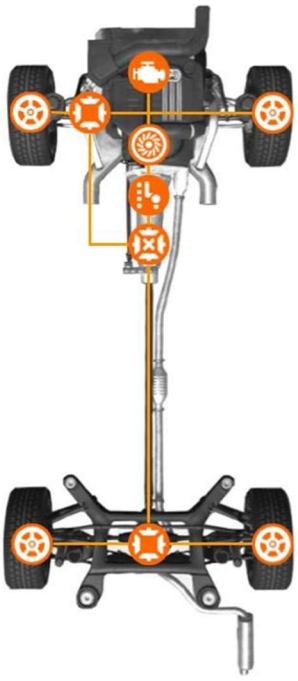

Abbildung 3 Schematische Darstellung des simulierten Antriebsstranges des ETK800 854t Modells Quelle: https://beamng.tech/blog/2021-06-21-beamng-tech-whitepaper/bng_technical_paper.pdf

Ein weiterer Grund für die Nutzung von BeamNG.tech ist die offene Schnittstelle BeamNG.py, mit der Python-Programme Zugriff auf die Simulationssoftware erhalten. Das beinhaltet unter anderem das Austauschen von Daten, Erstellen von Szenarien, Kontrollieren von Fahrzeugen oder auch das Platzieren und Abgreifen von Kameras am Fahrzeug. BeamNG.py macht die Entwicklung einer eigenen vereinfachten Schnittstelle erst

möglich. Ein weiterer Vorteil an BeamNG.py ist, dass es Open Source[1] ist. Durch diesen Umstand lässt sich aus dem Quellcode auf manche Funktionen schließen, bei denen die zwar umfangreiche, aber teils noch unvollständige Dokumentation, keine ausreichenden Informationen liefert.

3 Ist-Zustand

Der Ist-Zustand wird zwecks Übersichtlichkeit in drei Abschnitte unterteilt: Zunächst wird beschrieben mit welchen Mitteln die Studierenden aktuell in dem Modul „Bildverarbeitung im Automobil“ ihre entwickelte Software testen. Daraufhin wie mit diesen Daten eine Spurerkennung im Sommersemesters 2022 umgesetzt wurde und welche Ergebnisse diese liefert. Zuletzt wird der Prototyp der zu entwickelnden Software betrachtet, indem dessen Funktionen und der Aufbau aufgezeigt werden.

3.1 Testen ohne Simulation

Da zum Start des Sommersemesters 2022 noch keine Simulationen genutzt wurden, um Testbilder/Testvideos zu erzeugen, mussten diese auf andere Weise erlangt werden. Hierfür wurden zwei Wege genutzt, um an die Bildmaterialien zu kommen. Die erste Quelle waren die durch Professor Kleinsorge bereitgestellten Beispielbilder und -Videos. Diese umfassen vor allem Bilder mit Verkehrszeichen (Siehe Abbildung 4), Videos von Regenfahrten und Bilder von leeren Fahrspuren. Die genutzte Alternative dazu war es, eine Kamera im Auto eines Studierenden zu montieren und ein Video des Arbeitsweges zur Technischen Hochschule Wildau aufzunehmen. Die dabei erhobenen Daten wurden nur intern genutzt.

Abbildung 4 Beispielbild, das im Sommersemester 2022 genutzt wurde, um die entwickelten Programme zu testen

3.2 Spurerkennung Sommersemester 2022

Die Spurerkennung, implementiert in der Datei „modul_LDA.py“, wurde im Sommersemester 2022 von einem Kommilitonen programmiert. Sie war ein Teil einer Gruppenabgabe, bei der alle Kursteilnehmenden Funktionen zu einer Bildverarbeitungsbibliothek

beisteuerten. Das Modul *LDA*[6] beinhaltet zwei Funktionen: Die erste Funktion ist eine Fluchtpunkt-Erkennung, die in einem Bild den perspektivischen Punkt anzeigen soll, an dem alle real parallelen Linien zusammenlaufen. Die zweite Funktion ist eine Spurerkennung, die die erkannten Fahrbahnbegrenzungen auf einem verarbeiteten Bild einzeichnet. Da diese als Vergleich für den Funktionsbeweis gelten soll, wird anschließend erklärt, wie die Software ihre Ergebnisse erwirkt.

Um Spuren zu erkennen, wird zunächst das Quellbild in ein Graustufenbild umgewandelt und zusätzlich die Auflösung halbiert. Daraufhin wird die obere Hälfte des Bildes abgeschnitten und ein Gaußscher Weichzeichner gegen eventuelles Rauschen eingesetzt. Dann werden mit einem Canny-Algorithmus[2] Kanten erkannt und in einem weiteren Bild abgespeichert. Auf dieses Bild wird nun die Hough-Linien-Transformation[3] angewendet, um aus den Kanten Linien zu erkennen. Im darauffolgenden Schritt werden horizontale Linien eliminiert und favorisierte Linien hervorgehoben. Abschließend werden die erkannten Linien auf das Canny-Bild gezeichnet.

Unter Verwendung der ursprünglichen Standardkonfiguration wurden jedoch weder Fluchtpunkt noch die Fahrbahnbegrenzungen korrekt durch die Software erkannt. Auf dem mitgeliefertem Standard-Bild sehen die Ergebnisse der Erkennung wie folgt aus. (Siehe Abbildungen 5&6)

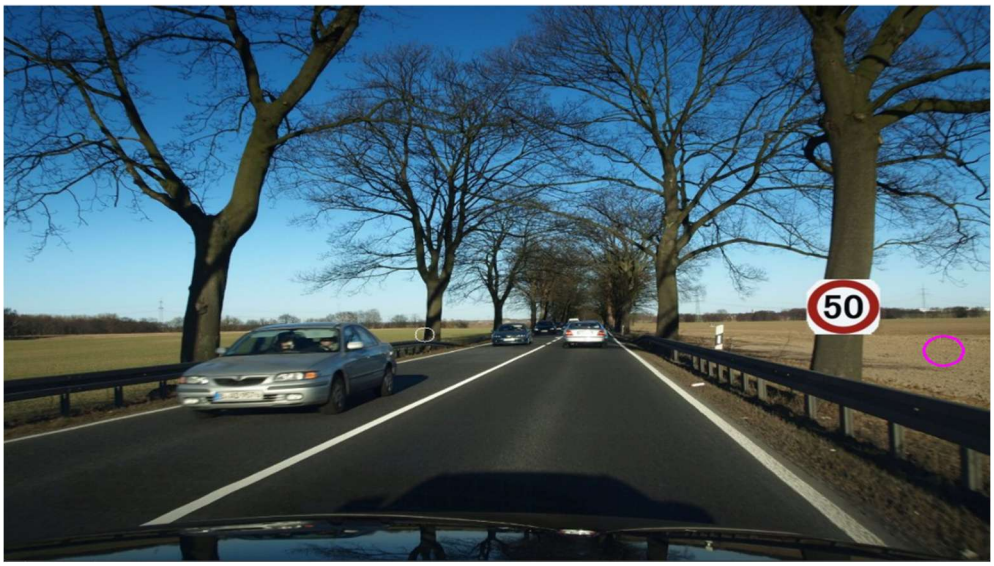

Abbildung 5 Bild des errechneten Fluchtpunktes (Pink) des LDA-Moduls aus dem Sommersemester 2022

Abbildung 6 Bild mit den erkannten Fahrbahnbegrenzungen (blau) des LDA-Moduls aus dem Sommersemester 2022

3.3 Prototyp Sommersemester 2022

Das „modul_VDS.py“, welches ebenfalls im Rahmen des Kurses „Bildverarbeitung im Automobil“ programmiert wurde, dient als Vorreiter für diese Arbeit. Als Hauptfunktion startet es ebenfalls die BeamNG.tech Fahrsimulation und lädt dort ein Szenario mit einem Fahrzeug, von dem es Daten abliest. Im Vergleich zu dieser Arbeit bietet das Modul einen Logging-Modus, welcher vergleichbar ist mit dem Recording-Modus, in dem er Fahrdaten und Bilder abspeichert und den Print-Modus, der die gesammelten Daten in einem Terminal ausgibt, sowie ein Fenster mit Bildern der montierten Frontkamera ausgibt.

```
simustart('l',5,10,'etk800','green','italy','beamCam')
```

Abbildung 7 Methodenaufruf mit Argumenten, um den Prototypen zu starten.

Optionen müssen als Argumente der „simustart()“ Methode mitgegeben werden. Dies ist auch der einzige Weg das Programm zu starten. Ein derartiger Methodenaufruf ist in der Abbildung 7 zu sehen, bei dem die Argumente folgendes bedeuten: Das erste Argument „l“ bestimmt den Modus. In diesem Fall Modus Logging. Die 5 und 10 stehen für Datenerhebungsintervalle pro Sekunde und die Länge der Programmlaufzeit in Sekunden. Das Argument „etk800“ ist der Name des Fahrzeuges, das simuliert werden soll und „green“ die Farbe dessen (grün). Argument sechs, „italy“ bestimmt die Karte, die geladen wird. Abschießend bestimmt „beamCam“ den Kameramodus, der verwendet werden soll.

Da dieser Prototyp für Version 0.24 von BeamNG.tech programmiert wurde und damals der eingebaute Kamerasensor auf den Entwicklungssystemen nur geringe Bildraten produzieren konnte, wurde ein zweiter Kameramodus entwickelt. Der Modus „winCam“ griff dabei das BeamNG.tech Fenster direkt ab. Zudem musste in der Simulation die Kameraeinstellung „Motorhaube“ genutzt werden, um verwendbare Aufnahmen aufzuzeichnen.

Weitere Einstellungen, wie das BeamNG-Nutzerverzeichnis oder die Position des Kamerasensors waren im Programmcode festgelegt. Häufig genutzte Werte standen als Kommentar über den jeweiligen Zeilen, damit die passende Zeile einkommentiert werden konnte, ohne sich die Werte merken zu müssen. Funktionen, wie eine Verzögerungserkennung für die Datenerhebungsintervalle waren bereits vorhanden, es fehlen allerding jegliche Fehlerbehandlungen oder weitere Komfortfunktionen.

4 Anforderungen

Da bei dieser Arbeit zwei Programme geschrieben werden, gibt es auch zwei separate Anforderungskataloge. Der erste Katalog beschreibt ausschließlich die Anforderungen an die zu entwickelnde Schnittstellensoftware, während der zweite Anforderungskatalog die Anforderungen für den Spurhalteassistent, der als Funktionsbeweis dienen soll, beinhaltet. Dabei wird in beiden Anforderungskatalogen zwischen *funktionalen* und *nicht-funktionalen* Anforderungen unterschieden. Diese sind durch das „F“ und respektive dem „NF“ in ihrem Bezeichner zu erkennen. Ebenfalls wird den jeweiligen Anforderungen eine Relevanz zugeordnet. Es wird hierbei zwischen den Relevanzstufen „Muss“, “Soll“ und „Kann“ unterschieden, welche in der Relevanz abfallend und zusätzlich farblich hinterlegt sind.

Da die Schnittstellensoftware vom Umfang her deutlich größer ist als der Funktionsbeweis, wird der erste Anforderungskatalog zusätzlich in vier Unterkategorien aufgeteilt. Die Anforderungen an die Grundfunktionalität haben einen Bezeichner ohne Zusatz. Anforderungen für den Modus „Recording“ haben am Ende ihres Bezeichners ein „R“ und für den Modus „Streaming“ ein „S“ angehangen. In der letzten Kategorie „Benutzeroberflächen“ wird zwischen den Zusätzen „T“ und „G“ unterschieden. „T“ beschreibt dabei Anforderungen an die textbasierte und „G“ für die grafische Benutzeroberfläche.

Nachdem die Anforderungen tabellarisch aufgezählt wurden, werden diese in einem Fließtext zusammengefasst. Dort werden in einem Satz erwähnte Anforderungen nach dem Satzzeichen in Klammern aufgeführt.

4.1 Funktionale Anforderungen

4.1.1 Anforderungen an die Grundfunktionalität (Tabelle)

Bezeichner	Beschreibung	Relevanz
F01	Die Software muss einen “Recording” Modus beinhalten.	Muss
F02	Die Software muss mit der Fahrsimulation BeamNG.tech interagieren können.	Muss
F03	Die Software muss die Fahrsimulationssoftware BeamNG.tech starten können.	Muss

F04	Die Software muss in BeamNG.tech eine Fahrsimulation starten können.	Muss
F05	Die Software muss eine Karte der Fahrsimulation laden können.	Muss
F06	Die zu ladende Karte muss einstellbar sein.	Muss
F07	Die Auswahl der eingestellten Karte muss persistent abgespeichert werden.	Muss
F08	Eine Liste unterstützter Karten muss angezeigt werden können.	Muss
F09	Die Liste unterstützter Karten kann durch die Nutzenden erweiterbar sein.	Kann
F10	Eine unterstützte Karte muss einen definierten Startpunkt besitzen.	Muss
F11	Die Software muss ein Fahrzeug in der geladenen Karte erscheinen lassen können.	Muss
F12	Es muss einstellbar sein, welches Fahrzeug erscheinen soll.	Muss
F13	Das eingestellte Fahrzeug muss persistent abgespeichert werden.	Muss
F14	Eine Liste mit unterstützten Fahrzeugen muss angezeigt werden können.	Muss
F15	Die Liste unterstützter Fahrzeuge kann durch die Nutzenden erweiterbar sein.	Kann
F16	Unterstützte Fahrzeuge müssen eine Kameraposition definiert haben.	Muss
F17	Die Software muss bei dem geladenen Fahrzeug Bilder von der Frontkamera abgreifen können.	Muss
F18	Das Sichtfeld der Frontkamera muss einstellbar sein.	Muss
F19	Das eingestellte Sichtfeld muss persistent gespeichert werden können.	Muss
F20	Die Auflösung der Frontkamera muss einstellbar sein.	Muss

F21	Die eingestellte Auflösung der Frontkamera muss persistent abgespeichert werden.	Muss
F22	Es muss Voreinstellungen für *VESA* anerkannte Auflösungsstandards geben.	Muss
F23	Sollte eine voreingestellte Auflösung eingestellt werden, muss persistent gespeichert werden, welche Voreinstellung gewählt wurde.	Muss
F24	Die Voreinstellungen kann durch die Nutzenden erweiterbar sein.	Kann
F25	Die Voreinstellungen müssen durch die Nutzenden überschreibbar sein.	Muss
F26	Die Software muss diese Bild-Daten auf dem Dateisystem des genutzten Rechners persistent abspeichern können.	Muss
F27	Der Speicherort der abgespeicherten Bilder muss einstellbar sein.	Muss
F28	Der eingestellte Speicherort muss persistent abgespeichert werden.	Muss
F29	Von dem geladenen Fahrzeug müssen Fahrzeugdaten ausgelesen werden.	Muss
F30	Von dem geladenen Fahrzeug muss die Radgeschwindigkeit ausgelesen werden.	Muss
F31	Von dem geladenen Fahrzeug sollte die Radgeschwindigkeit individuell für jedes Rad ausgelesen werden können.	Soll
F32	Von dem geladenen Fahrzeug muss die Gaspedalposition in % ausgelesen werden.	Muss
F33	Von dem geladenen Fahrzeug muss Gasstärke in % (Prozent) ausgelesen werden.	Muss
F34	Von dem geladenen Fahrzeug muss die Bremspedalstärke in % (Prozent) ausgelesen werden.	Muss

F35	Von dem geladenen Fahrzeug muss die Bremsstärke in % ausgelesen werden.	Muss
F36	Von dem geladenen Fahrzeug muss die Kupplungspedalposition in % (Prozent) ausgelesen werden.	Muss
F37	Von dem geladenen Fahrzeug muss die Kupplungsstärke in % (Prozent) ausgelesen werden.	Muss
F38	Von dem geladenen Fahrzeug muss die Stellung der Parkbremse in % (Prozent) ausgelesen werden.	Muss
F39	Von dem geladenen Fahrzeug muss die Stärke der Parkbremse in % (Prozent) ausgelesen werden.	Muss
F40	Von dem geladenen Fahrzeug muss der Motorstatus (Ein, Zündung, Aus) ausgelesen werden	Muss
F41	Von dem geladenen Fahrzeug muss die Motorlast in % (Prozent) ausgelesen werden.	Muss
F42	Von dem geladenen Fahrzeug muss die Motordrehzahl ausgelesen werden können.	Muss
F43	Von dem geladenen Fahrzeug muss der Gang des Getriebes ausgelesen werden.	Muss
F44	Von dem geladenen Fahrzeug muss der *ABS*-Status ausgelesen werden. (An/Aus, nicht vorhanden)	Muss
F45	Von dem geladenen Fahrzeug muss die *ABS*-Aktivität ausgelesen werden. (Passiv/Eingreifend)	Muss
F46	Von dem geladenen Fahrzeug muss der *ASR*-Status ausgelesen werden. (An/Aus, nicht vorhanden)	Muss
F47	Von dem geladenen Fahrzeug muss die *ASR*-Aktivität ausgelesen werden. (Passiv/Eingreifend)	Muss
F48	Von dem geladenen Fahrzeug muss der *ESP*-Status ausgelesen werden. (An/Aus, nicht vorhanden)	Muss
F49	Von dem geladenen Fahrzeug muss die *ESP*-Aktivität ausgelesen werden. (Passiv/Eingreifend)	Muss

F50	Von dem geladenen Fahrzeug muss der Scheinwerferstatus ausgelesen werden.	Muss
F51	Dieser Scheinwerferstatus sollte zwischen aus, Abblendlicht und Fernlicht unterscheiden.	Soll
F52	Von dem geladenen Fahrzeug muss der Blinker-Status ausgelesen werden.	Muss
F53	Von dem geladenen Fahrzeug muss die Fahrtrichtung ausgelesen werden.	Muss
F54	Von dem geladenen Fahrzeug muss die Position auf der gewählten Karte ausgelesen werden.	Muss
F55	Von dem geladenen Fahrzeug müssen die Lagewinkel ausgelesen werden.	Muss
F56	Von der Fahrsimulation muss die simulierte Zeit ermittelt werden.	Muss
F57	Die Software muss die Realzeit ermitteln.	Muss
F58	Die ausgelesenen Daten müssen auf dem Dateisystem des genutzten Rechners persistent abgespeichert werden können.	Muss
F59	Die Geschwindigkeit der Simulation muss durch die Nutzenden einstellbar sein.	Muss
F60	Die eingestellte Simulationsgeschwindigkeit muss persistent abgespeichert werden.	Muss
F61	Die simulierte Zeit muss durch die Nutzenden einstellbar sein.	Muss
F62	Die eingestellte Zeit muss persistent abgespeichert werden.	Muss
F63	Es muss möglich sein das Wetter der Simulation einzustellen.	Muss
F64	Nutzende müssen aus mehreren Wetter-Voreinstellungen auswählen können.	Muss

F65	Die eingestellte Wetter-Voreinstellung muss persistent abgespeichert werden.	Muss
F66	Die Nebeldichte muss konfigurierbar sein	Muss
F67	Die eingestellte Nebeldichte muss persistent abgespeichert werden.	Muss
F68	Die Software muss über eine textbasierte Benutzeroberfläche verfügen.	Muss
F69	Die Software muss über eine grafische Benutzeroberfläche verfügen.	Muss
F70	Die beiden Benutzeroberflächen sollten in ihren Funktionen übereinstimmen.	Soll
F71	Die Benutzeroberflächen müssen mehrere Anzeigesprachen unterstützen.	Muss
F72	Die eingestellte Anzeigesprache muss persistent gespeichert werden.	Muss
F73	Es soll die Möglichkeit geben alle Einstellungen nur zur Laufzeit der Software zu verändern.	Soll
F74	Die persistent gespeicherten Einstellungen müssen menschenlesbar formatiert sein.	Muss
F75	Die Software soll über eine geschützte Standardkonfiguration verfügen.	Soll
F76	Die Software soll erkennen, wenn die persistent gespeicherte Konfiguration beschädigt oder gelöscht wurde.	Soll
F77	Die Software soll eine beschädigte oder gelöschte Konfiguration mit der Standardkonfiguration ersetzen.	Soll
F78	Für die Software muss ein Benutzerhandbuch erstellt werden.	Muss
F79	In dem erstellten Benutzerhandbuch muss erklärt werden wie die Software installiert wird.	Muss

F80	In dem erstellten Benutzerhandbuch muss erklärt werden wie die Software initial eingerichtet werden muss.	Muss
F81	In diesem Benutzerhandbuch muss erklärt werden, welche Auswirkungen die einzelnen Einstellungen haben.	Muss
F82	Die Software muss einen „Streaming“ Modus beinhalten.	Muss

Tabelle 1 Funktionale Anforderungen an die Grundfunktionen der Schnittstellensoftware

4.1.2 Anforderungen an die Grundfunktionen (Textform)

Es muss eine Software entwickelt werden, welche mit der Fahrsimulationssoftware BeamNG.tech starten und mit dieser interagieren kann. (F02, F03) Die Software muss eine Fahrsimulation, mit einstellbarer Karte und einstellbarem Fahrzeug starten können. (F04, F05, F06, F11, F12) Welche Karte und welches Fahrzeug gewählt wurde, muss persistent abgespeichert werden. (F07, F13) Kompatible Karten haben dabei einen festen Startpunkt und es muss eine Liste der kompatiblen Karten angezeigt werden können. (F08, F10) Die Liste der kompatiblen Karten kann auch durch die Nutzenden erweiterbar sein. (F09) Ähnlich wie bei den Karten, muss auch eine Liste kompatibler Fahrzeuge ausgegeben werden. (F14) Diese Liste kann ebenfalls durch die Nutzenden erweiterbar sein, Fahrzeuge hingegen benötigen jedoch eine feste Kameraposition. (F15, F16)

Von dieser Kameraposition aus müssen dann Bilder mit einstellbarer Auflösung und einstellbarem Sichtfeld abgegriffen werden können. (F17, F18, F20) Beispiel-Auflösungen müssen aus einer Liste von *VESA* anerkannten Auflösungs-Voreinstellungen stammen. (F22) Den Nutzenden muss zusätzlich die Möglichkeit gegeben werden, eine eigene Kameraauflösung festzulegen, so wie sie auch die Fähigkeit haben können, die Liste der Voreinstellungen zu erweitern. (F24, F25) Die gewählte Konfiguration der Kamera muss anschließend persistent abgespeichert werden. (F19, F21) Bilder der Frontkamera müssen auf dem Rechner der Nutzenden unter einem einstellbaren Pfad abgespeichert werden können. (F26, F27) Der eingegebene Pfad muss ebenfalls persistent gesichert werden. (F28)

Während der laufenden Simulation müssen Fahrdaten des geladenen Fahrzeuges ausgelesen werden. (F29) Die auszulesenden Daten lauten wie folgt:

- Radgeschwindigkeit in m/s (F30)
- Gaspedalposition in % (F32)
- Gasstärke in % (F33)
- Bremspedalposition in % (F34)
- Bremsstärke in % (F35)
- Kupplungspedalposition in % (F36)
- Kupplungsstärke in % (F37)
- Stellung der Parkbremse (F38)
- Stärke der Parkbremse (F39)
- Motorstatus (An/Zündung/Aus) (F40)
- Motorlast in % (F41)
- Motordrehzahl (F42)
- Gang des Getriebes (F43)
- *ABS*-Status (An/Aus/ nicht vorhanden) (F44)
- *ABS*-Aktivität (Passiv/Eingreifend) (F45)
- *ASR*-Status (An/Aus/ nicht vorhanden) (F46)
- *ASR*-Aktivität (Passiv/Eingreifend) (F47)
- *ESP*-Status (An/Aus/ nicht vorhanden) (F48)
- *ESP*-Aktivität (Passiv/Eingreifend) (F49)
- Scheinwerferstatus (F50)
- Blinker-Status (Links, Rechts, aus) (F52)
- Fahrtrichtung (F53)
- Position auf der gewählten Karte (F54)
- Lagewinkel (F55)

Zusätzlich soll die Radgeschwindigkeit jedes individuellen Rades ausgelesen werden und der Scheinwerferstatus zwischen Abblend-, Fernlicht und ausgeschaltet unterscheiden. (F31, F51) Außerdem muss die Software sowohl die Realzeit als auch die Zeit in der Simulation ermitteln. (F56, F57) Alle erhobenen Daten müssen persistent auf dem Dateisystem der Nutzenden abgespeichert werden. (F58) Den Nutzenden muss es ebenfalls möglich sein die Zeit der Simulation einzustellen. Die eingestellte Zeit muss auch persistent

gespeichert werden. (F61, F62) Es muss zusätzlich möglich sein, die Simulationsgeschwindigkeit und weitere Umgebungsvariablen, wie beispielsweise das Wetter und die Nebeldichte einzustellen. (F59, F63, F66) Die vorstehend genannten Einstellungen müssen persistent abgespeichert werden. (F60) Zudem ist die Bereitstellung auswählbarer Voreinstellungen für die Wetterbedingungen erforderlich. (F64, F65, F67)

Die Software muss über zwei Benutzeroberflächen verfügen, eine textbasierte und eine grafische, deren Funktionen deckungsgleich sein sollten. (F68, F69, F70) Beide Oberflächen müssen multilingual umgesetzt und die ausgewählte Sprache persistent gespeichert werden. (F71, F72) Alle getätigten Einstellungen, die persistent gespeichert wurden, müssen in einem menschenlesbaren Format gesichert sein. (F74) Zudem soll erkannt werden, ob die gespeicherte Konfiguration fehlt oder beschädigt wurde und diese dann mit einer geschützten Standardkonfiguration ersetzt werden. (F75, F76, F77) Abschließend muss für die Software ein Benutzerhandbuch angelegt werden, welches die Installation, initiale Einrichtung, die Einstellungen und deren Auswirkungen erklärt. (F78, F79, F80, F81)

4.1.3 Anforderungen an den Modus Recording (Tabelle)

Bezeichner	Beschreibung	Relevanz
F83R	Im Modus „Recording“ müssen alle erhobenen Daten persistent gespeichert werden.	Muss
F84R	Die Daten müssen in einem menschenlesbaren Dateiformat abgespeichert werden.	Muss
F85R	Die Daten müssen in einem maschinen-iterierbaren Dateiformat abgespeichert werden.	Muss
F86R	Aus der Datei muss entnommen werden können, welcher Kategorie ein Wert angehört.	Muss
F87R	Aus der Datei kann entnommen werden, welche Spezifikationen der genutzte Rechner hat.	Kann
F88R	Aus der Datei kann entnommen werden, mit welcher Softwareversion die Daten aufgenommen wurden.	Kann
F89R	Die Daten müssen in Intervallen mit festen Abständen erhoben werden.	Muss

F90R	Der Abstand zwischen zwei Intervallen muss einstellbar sein.	Muss
F91R	Die Einstellung des Intervallabstands muss persistent gespeichert werden.	Muss
F92R	Die Software muss die Anzahl der vergangenen Datenerhebungsintervalle festhalten.	Muss
F93R	Die Software muss zu jedem erhobenen Datensatz aufzeichnen, in welchem Datenerhebungsintervall dieser erhoben wurde.	Muss
F94R	In jedem Datenerhebungsintervall muss ein Bild der Frontkamera des Fahrzeuges erhoben werden.	Muss
F95R	Dieses Bild muss dem Datenerhebungsintervall, in dem es erhoben wurde, eindeutig zuordenbar sein.	Muss
F96R	Die Software muss erkennen, wenn der eingestellte Intervall-Zeitraum nicht eingehalten werden kann.	Muss
F97R	Die Software muss erkennen, wenn die eingestellte Kamerabildrate nicht erreicht werden kann.	Muss
F98R	Die Software soll die Simulationsgeschwindigkeit verringern, wenn der Intervall-Zeitraum nicht eingehalten werden kann.	Soll
F99R	Die Software soll die Simulationsgeschwindigkeit verringern, wenn die eingestellte Kamerabildrate nicht erreicht werden kann.	Soll
F100R	Die Software muss die Nutzenden informieren, sollte der Intervall-Zeitraum nicht eingehalten werden können.	Muss
F101R	Die Software muss die Nutzenden informieren, sollte die eingestellte Kamerabildrate nicht erreicht werden können.	Muss
F102R	Die Software muss sicherstellen, dass die Bilderaufnahme während des zugeordneten Datenerhebungsintervall geschieht.	Muss

F103R	Die Software kann eine einstellbare Toleranz für den Intervall-Zeitraum besitzen.	Kann
F104R	Die Software kann eine einstellbare Toleranz für die Bildrate besitzen.	Kann
F105R	Die eingestellte Toleranz kann persistent abgespeichert werden.	Kann
F106R	Die Software soll Bilder vor dem Speichern komprimieren.	Soll

Tabelle 2 Funktionale Anforderungen and den Recording-Modus der Schnittstellensoftware

4.1.4 Anforderungen an den Modus Recording (Textform)

Es muss ein Recording-Modus existieren, bei dem alle erhobenen Daten in einem menschenlesbaren Format abgespeichert werden. (F01, F82, F83R, F84R) Um die Datenauswertung zu erleichtern, ist es erforderlich, dass das Format maschinen-iterierbar und die Datei so strukturiert ist, dass der Wert einer Kategorie eindeutig bestimmt werden kann. (F85R, F86R) Zusätzlich kann in der Datei vermerkt sein, welche Hardware und Softwareversion genutzt wurde, um die Daten aufzuzeichnen. (F87R, F88R) Alle Daten müssen in Intervallen mit fest einstellbaren Abständen erhoben werden. (F89R, F90R) Die eingestellte Intervalllänge muss persistent abgespeichert werden. (F91R) Die erhobenen Daten müssen dem Datenerhebungsintervall zugordnet werden, in dem sie ausgelesen wurden und die Anzahl dieser muss gezählt werden. (F92R, F93R)

Ebenfalls muss in jedem Datenerhebungsintervall ein Bild der Frontkamera des simulierten Fahrzeuges erzeugt und dem Datenerhebungsintervall zugeordnet werden. (F94R, F95R) Dabei muss die Software erkennen, ob die eingestellten Intervallabstände zwischen den Datenerhebungen und Bildaufnahmen eingehalten werden. Bei Nichteinhaltung soll die Simulationsgeschwindigkeit verringert werden. (F96R, F97R, F98R, F99R) Zusätzlich müssen die Nutzenden informiert werden, sollte einer der Intervallabstände nicht eingehalten werden können. (F100R, F101R) Ebenfalls ist durch die Software sicherzustellen, dass die Bildaufnahme während des zugeordneten Datenerhebungsintervalls geschieht. (F102R) Es kann eine einstellbare Toleranz für die Intervallabstände und Bildraten geben, die persistent abgespeichert wird. (F103R, F104R, F105R) Abschließend soll die Software aufgenommene Bilder vor dem Speichern komprimieren. (F106R)

4.1.5 Anforderungen an den Modus „Streaming“ (Tabelle)

Bezeichner	Beschreibung	Relevanz
F107S	Die Software muss, im Modus „Streaming“, alle erhobenen Daten über eine Schnittstelle verfügbar machen.	Muss
F108S	Die Software muss Nachrichten über diese Schnittstelle verarbeiten können.	Muss
F109S	Die Software muss auf eingehende Nachrichten antworten können.	Muss
F110S	Über diese Schnittstelle muss eine andere Software die textbasierten Daten abrufen können.	Muss
F111S	Als Antwort müssen alle textbasierten Daten in einer Antwort übertragen werden.	Muss
F112S	Die Daten in dieser Antwort müssen in einem festen Format übertragen werden.	Muss
F113S	Die Daten in dieser Antwort müssen maschinen-iterierbar sein.	Muss
F114S	Über die Schnittstelle muss es möglich sein, das aktuelle Bild der Frontkamera anzufragen.	Muss
F115S	Als Antwort muss das angefragte Bild über die Schnittstelle versendet werden.	Muss
F116S	Zusätzlich kann das letzte Bild davor gesendet werden.	Kann
F117S	Die Software soll erkennen, ob das zuletzt angefragte Bild sich von dem neu angefragten Bild unterscheidet.	Soll
F118S	Die Software soll die Nutzenden warnen, wenn das zuletzt angefragte Bild sich von dem neuen Bild unterscheidet.	Soll
F119S	Es muss möglich sein, über die Schnittstelle Steuereingaben an die Software zu senden.	Muss
F120S	Diese Steuereingaben müssen bei dem simulierten Fahrzeug angewendet werden.	Muss

F121S	Die Software muss die Steuereingabe „Beschleunigen“ bereitstellen.	Muss
F122S	Die Software muss die Steuereingabe „Bremsen“ bereitstellen.	Muss
F123S	Die Software muss die Steuereingabe „Lenken“ bereitstellen.	Muss
F124S	Die Steuereingaben sollen separat an die Software gesendet werden können.	Soll
F125S	Alle Arten der Steuereingabe sollen simultan gesendet werden können.	Soll
F126S	Manuell getätigte Steuereingaben der Nutzenden sollen die empfangenen Steuereingaben überschreiben können.	Soll
F127S	Die Software kann eine Bestätigung senden, wenn sie Steuereingaben erhalten hat.	Kann
F128S	Die Software soll Bilder für das Versenden vorladen.	Soll
F129S	Das Vorladen soll mit einer anpassbaren Rate passieren.	Soll
F130S	Die Software kann sich im Modus „Streaming“ mit einer laufenden Simulation verbinden.	Kann
F131S	Während dieser Verbindung kann die Software mit der laufenden Simulation interagieren, exakt so, als hätte die Software die Simulation selbst gestartet.	Kann
F132S	Es soll möglich sein, die Simulation über die Schnittstelle zu beenden.	Soll

Tabelle 3 Funktionale Anforderungen an den Modus Streaming der Schnittstellensoftware

4.1.6 Anforderungen an den Modus „Streaming“ (Textform)

Die Software muss über einen Streaming-Modus verfügen, der alle erhobenen Daten über eine Schnittstelle verfügbar macht (F82, F107S) und über diese Schnittstelle Nachrichten verarbeiten und beantworten. (F108S, F109S) Andere Software muss diese Schnittstelle nutzen können, um alle textbasierten Daten anzufragen, welche als Antwort in einem festen, maschinen-iterierbaren Format übertragen werden. (F110S, F111S, F112S, F113S) Ebenfalls muss die Schnittstelle ermöglichen das aktuelle Bild der Frontkamera des simulierten Fahrzeugs anzufragen, welches anschließend als Antwort übertragen wird. (F114S, F115S) Zusätzlich zum aktuellen Bild kann das zuletzt angefragte Bild in der Antwort mitgesendet werden. (F116S) Außerdem soll die Software überprüfen, ob das zuletzt gesendete und neu angefragte Bild identisch sind und die Nutzenden darüber warnen, sollte dies der Fall sein. (F117S, F118S) Bilder sollen mit einer anpassbaren Rate vorgeladen werden. (F128S, F129S)

Die Schnittstelle muss Steuereingaben akzeptieren und diese an das simulierte Fahrzeug weiterleiten. (F119S, F120S) Die möglichen Steuereingaben müssen „Beschleunigen“, „Bremsen“ und „Lenken“ umfassen. (F121S, F122S, F123S) Steuereingaben sollen getrennt oder kombiniert akzeptiert werden. (F124S, F125S) Die Nutzenden sollen stets die Möglichkeit haben, die empfangenen Steuereingaben mit manuellen Steuereingaben zu überschreiben. (F126S) Als Antwort auf die empfangenen Steuereingaben kann eine Bestätigung gesendet werden. (F127S) Die Software kann sich im Modus Streaming mit einer laufenden Simulation verbinden, mit der in gleicher Weise interagiert werden kann, wie mit einer durch die Software gestarteten Simulation. (F130S, F131S) Abschließend soll es möglich sein, die Simulation über eine Nachricht an die Schnittstelle zu beenden. (F132S)

4.1.7 Anforderungen an die Benutzeroberflächen (Tabelle)

Bezeichner	Beschreibung	Relevanz
F133T	Die textbasierte Benutzeroberfläche muss eine Funktion zur Anzeige der aktuellen Konfiguration umfassen.	Muss
F134T	Die textbasierte Benutzeroberfläche muss eine Funktion zur Änderung der aktuellen Konfiguration beinhalten.	Muss
F135T	Die Simulation muss über eine textbasierte Benutzeroberfläche gestartet werden können.	Muss
F136T	Der Software müssen beim Start über die Eingabeaufforderung Parameter mitgegeben werden können.	Muss
F137T	Über diese Parameter müssen alle Einstellungen konfigurierbar sein.	Muss
F138T	Die Software kann bei der Mitgabe eines festgelegten Parameters die Benutzeroberflächen überspringen und die Simulation sofort starten.	Kann
F139T	Die textbasierte Benutzeroberfläche muss den Nutzenden zeigen, wo sie sich eine BeamNG.tech Lizenz ausstellen lassen können.	Muss
F140G	Die grafische Benutzeroberfläche muss die Simulation mit der aktuellen Konfiguration starten können.	Muss
F141G	Die grafische Benutzeroberfläche muss die aktuelle Konfiguration anzeigen können.	Muss
F142G	Die grafische Benutzeroberfläche muss die aktuelle Konfiguration verändern können.	Muss
F143G	Die grafische Benutzeroberfläche muss den Nutzenden zeigen, wo sie sich eine BeamNG.tech Lizenz ausstellen lassen können.	Muss

Tabelle 4 Funktionale Anforderungen an die Benutzeroberflächen der Schnittstellensoftware

4.1.8 Anforderungen an die Benutzeroberflächen (Textform)

Die textbasierte Oberfläche soll die aktuelle Konfiguration sowohl anzeigen als auch deren Änderung ermöglichen. (F133T, F134T) Beim Start der Software über die Eingabeaufforderung müssen Parameter akzeptiert werden, mit denen alle Einstellungen in der Konfiguration verändert werden können. (F136T, F137T) Ebenso muss die Simulation über die textbasierte Oberfläche gestartet werden können. (F135T) Als zusätzliche Startoption kann ein weiterer festgelegter Parameter akzeptiert werden, welcher alle Oberflächen überspringt und die Simulation mit der gespeicherten Konfiguration sofort startet. (F138T)

Die grafische Benutzeroberfläche muss sowohl die Anzeige als auch die Modifikation der aktuellen Konfiguration unterstützen. (F141G, F142G) Die Simulation muss auch über die grafische Benutzeroberfläche gestartet werden können. (F140G) Abschließend müssen beide Oberflächen die Nutzenden darauf hinweisen, wo sie sich eine neue BeamNG.tech Lizenz ausstellen lassen können. (F139T, F143T)

4.2 Nicht-funktionale Anforderungen

4.2.1 Nicht-funktionale Anforderungen (Tabelle)

Bezeichner	Beschreibung	Relevanz
NF01	Die Software muss mit Windows 11 23H2 kompatibel sein.	Muss
NF02	Die Software kann mit Ubuntu 24.04.1 LTS kompatibel sein.	Kann
NF03	Von der Software soll eine ausführbare Binärdatei erstellt werden. (.exe)	Soll
NF04S	Die Latenz zwischen Anfrage und Antwort der textbasierten Daten muss im Streaming-Modus unter 5ms liegen.	Muss
NF05S	Die Latenz zwischen Anfrage und Antwort eines Frontkamerabildes (1280x720 Pixel, FOV=60°), muss im Streaming-Modus unter 10ms liegen.	Muss
NF06R	Auf dem Desktop-PC-Testsystem muss eine Aufnahme der Daten und der Frontkamera (1920x1080 Pixel, FOV=60°) mit 60 Intervallen die Sekunde möglich sein.	Muss
NF07R	Auf dem Fahrsimulator-Testsystem muss eine Aufnahme der Daten und der Frontkamera (1280x720Pixel, FOV=60°) mit 30 Intervallen die Sekunde möglich sein.	Muss
NF08R	Auf dem Laptop-Testsystem soll eine Aufnahme der Daten und der Frontkamera (1280x720Pixel, FOV=60°) mit 30 Intervallen die Sekunde möglich sein.	Soll

Tabelle 5 Nicht funktionale Anforderungen an die Schnittstellensoftware

4.2.2 Nicht Funktionale Anforderungen (Textform)

Die Software muss mit Windows 11 23H2 als Hauptziel entwickelt werden und damit auch voll kompatibel sein. (NF01) Eine Kompatibilität mit der Linux-Distribution Ubuntu 24.04.1 kann auch umgesetzt werden. (NF02) Um die Installation unter Windows zu vereinfachen, soll eine ausführbare Binärdatei (.exe) erstellt werden. (NF03) Im Streaming-Modus darf die Latenz zwischen Anfrage und Antwort bei textbasierten Daten nicht über 5ms und bei Bildern (1280x720 Pixel, FOV=60°) nicht über 10ms Liegen. Im Recording-Modus muss auf dem Desktop-PC-Testsystem eine Datenerhebungsintervallrate von 60 die Sekunde erreicht werden, während Bilder der Auflösung Full-HD und einem Sichtfeld von 60° gespeichert werden. (NF06R) Auf dem Testsystem müssen 30 Datenerhebungsintervalle mit HD als Kameraauflösung erreicht werden. (NF07R) Die gleichen Werte sollen auch bei dem Laptop-Testsystem erreicht werden. (NF08R)

4.3 Anforderungen Spurhalteassistenz

Bezeichner	Beschreibung	Relevanz
F01	Es muss eine Spurhalteassistenz entwickelt werden, die als Funktionsbeweis der zuvor entwickelten Schnittstellensoftware agiert.	Muss
F02	Die zu entwickelnde Spurhalteassistenz muss über die Schnittstelle des Streaming-Modus Nachrichten austauschen können.	Muss
F03	Über diese Schnittstelle müssen Bilder empfangen werden.	Muss
F04	Auf den empfangenen Bildern müssen Fahrbahnen mit durchgezogenen Linien als Fahrbahnmarkierungen erkannt werden.	Muss
F05	Die Fahrbahnerkennung kann auch mit gestrichelten Fahrbahnmarkierungen funktionieren.	Kann
F06	Die Spurhalteassistenz muss die Mitte der erkannten Fahrbahn finden.	Muss
F07	Die Spurhalteassistenz muss Steuerungsanweisungen über die Schnittstelle senden.	Muss

F08	Die Spurhalteassistenz muss das simulierte Fahrzeug zur Mitte der Fahrbahn lenken.	Muss
NF01	Die Spurhalteassistenz muss bei 10km/h für mindestens 30 Sekunden die Spur halten.	Muss
NF02	Die Spurhalteassistenz soll bei 20km/h für mindestens 30 Sekunden die Spur halten.	Soll

Tabelle 6 Anforderungen an den Spurhalteassistenten

Als Funktionsbeweis muss ein Spurhalteassistent entwickelt werden, der den Streaming-Modus der zuvor entwickelten Schnittstellensoftware nutzt. (F01) Über diese Schnittstelle müssen Nachrichten ausgetauscht und Bilder empfangen werden können. (F02, F03) Auf den empfangenden Bildern muss die Fahrbahn anhand von durchgezogenen Markierungen erkannt werden. (F04) Es können auch Fahrbahnen mit unterbrochenen Markierungen erkannt werden. (F05) Die Spurhalteassistentssoftware muss die Mitte der erkannten Fahrbahnen finden und das simulierte Fahrzeug über die Schnittstelle in Richtung dieser Mitte lenken. (F06, F08) Um dies zu erreichen, müssen Steuerungsanweisungen über die Streaming-Schnittstelle versendet werden. (F07) Der Spurhalteassistent muss die Spur bei einer Geschwindigkeit von 10km/h für 30 Sekunden halten können, soll dies aber auch bei 20km/h erreichen. (NF01, NF02)

5 Konzept

In diesem Abschnitt der Arbeit werden die Konzepte für das Schnittstellen-Programm, welches mit der BeamNG.tech Fahrsimulation kommuniziert und den als Funktionsbeweis entwickelten Spurhalteassistent beschrieben. Zuerst wird das Softwarekonzept der Schnittstellensoftware vorgestellt und daraufhin die einzelnen Teilprogramme in ihrer geplanten Funktionsweise. Abschließend wird eine Übersicht über den Spurhalteassistent präsentiert und die beiden Unterkomponenten näher betrachtet. Am Ende eines Absatzes werden zusätzlich die Anforderungen aufgezählt, die durch diesen erfüllt werden.

5.1 Konzept Softwarestruktur

Die Struktur der zu entwickelnden Software besteht aus fünf Python-Programmen, die untereinander kommunizieren und sich gegenseitig starten. Python wurde als Programmiersprache ausgewählt, da die BeamNG.tech-Schnittstelle BeamNG.py nur für Python

verfügbar ist. Der Programmablauf geschieht dabei stets linear. Das Startprogramm verarbeitet Startparameter und öffnet daraufhin die gewählte Benutzeroberfläche. Dort kann die Software konfiguriert und der Simulations-Connector gestartet werden, welcher dann mit der BeamNG.tech-Simulation kommuniziert. Texte, Inhalte und Konfigurationen werden dabei stehts vom Config-Dateiinterface verwaltet. (F01, F02, F03)

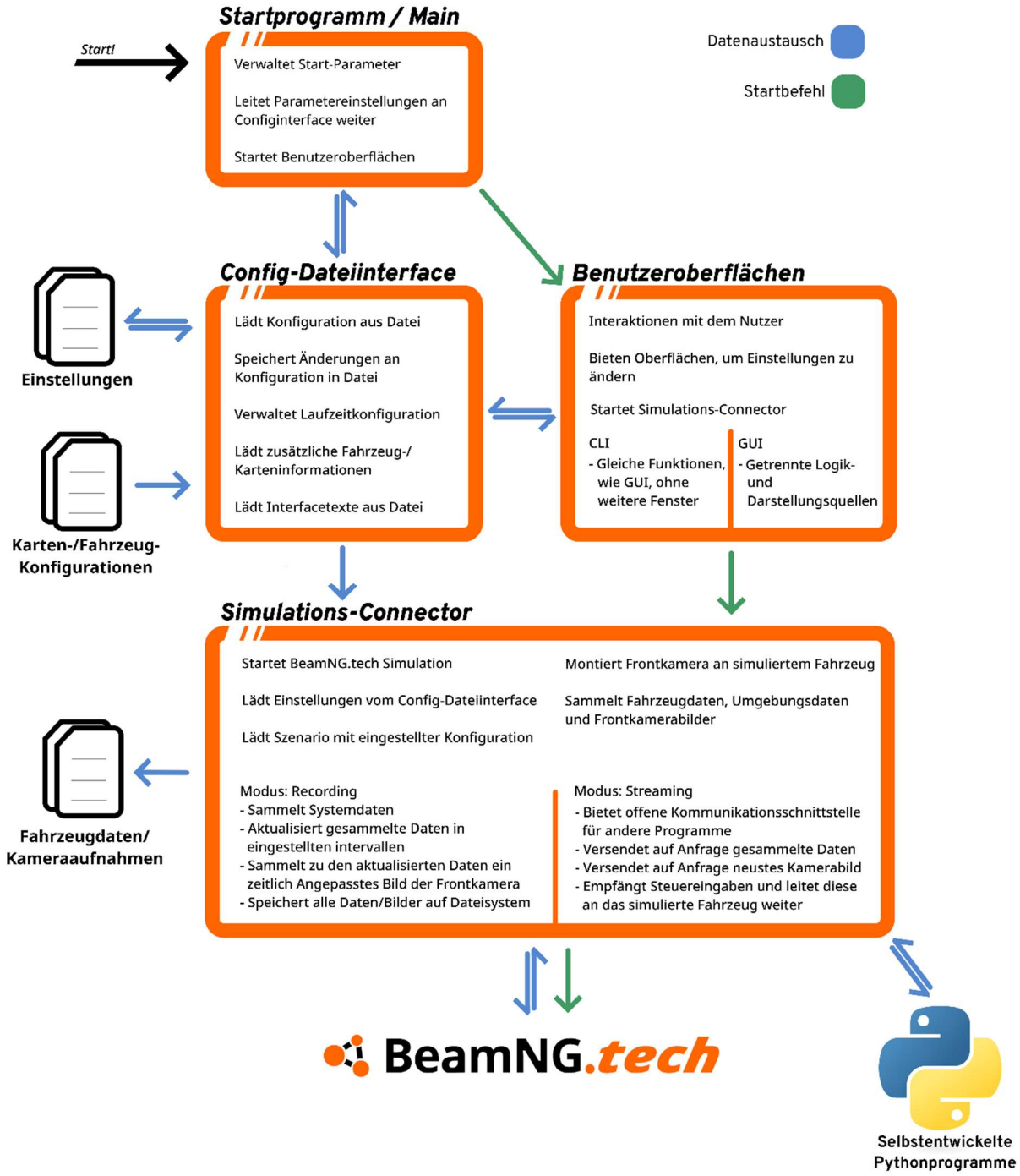

Abbildung 8 Übersichtsgrafik des Softwarestrukturkonzeptes.

5.1.1 Startprogramm / Main

Das Startprogramm ist der Einstiegspunkt für die gesamte Software. Diesem können beim Start Parameter mitgegeben werden, um die Konfiguration der Software anzupassen oder die Simulation direkt zu starten. Die angegebenen Parameter werden auf Fehlerfreiheit überprüft und dann an das Config-Dateiinterface weitergeleitet, um weiterverarbeitet zu werden. Sollte die Option des Direktstarts gewählt worden sein, wird dem Simulations-Connector der Befehl, zum Starten der BeamNG.tech Simulation, gegeben. Andernfalls wird die aktuell eingestellte Benutzeroberfläche gestartet. Zusätzlich wird den Nutzenden eine Erklärung der möglichen Parameter angezeigt, wenn sie das Programm mit „-h“ als Parameter aufrufen. Die Texte der Erklärung werden dabei in der eingestellten Sprache angezeigt und vom Config-Dateiinterface bereitgestellt. (F136T-F138T)

5.1.2 Config-Dateiinterface

Das Config-Dateiinterface verwaltet alle Dateien, die die Informationen für die Schnittstellensoftware beinhalten und betreffen: Eine Konfigurationsdatei, eine Datei mit kompatiblen Fahrzeugen und diese betreffenden Zusatzinformationen, eine Datei mit den unterstützten Karten und deren Startpunkten, Dateien in denen Szenarien abgespeichert sind und die Nachrichtendateien, die die Texte für Benutzeroberflächen in verschiedenen Sprachen beinhalten. Die Fahrzeugdatei beinhaltet, neben dem Fahrzeugmodell, auch Positionsdaten für die Frontkamera. Da die Dateien alle menschenlesbar und gleichzeitig maschineniterierbar sein müssen, wird das Dateiformat *JSON* gewählt. Eine *CSV*-Datei kann die Anforderungen auch erfüllen, da sich *JSON*-Dateien aber direkt in Python-Dictionary Datentypen umwandeln lassen und eine bessere Übersicht für die Nutzenden bieten, wird ein Abspeichern im *CSV*-Format nicht weiter betrachtet. Durch die Menschenlesbarkeit der Inhaltsdateien, wie der Karten- oder Fahrzeugdatei, können diese auch recht einfach von den Nutzenden selbständig erweitert werden. (F07, F09, F15, F16, F74)

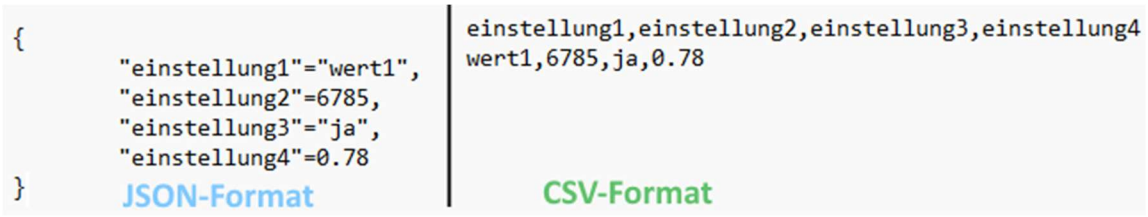

Abbildung 9 Vergleich Einstellungen im JSON- und CSV-Formats

In der Konfigurationsdatei müssen folgende Optionen gespeichert werden:

- Anzeigesprache der Benutzeroberflächen
- Pfad zur BeamNG.tech Installation
- Pfad zum BeamNG Nutzerordner
- Pfad zum Abspeichern von Aufzeichnungen
- Benutzeroberflächen-Modus
- Fahrzeug
- Karte
- Zeit
- Wettervoreinstellung
- Nebel
- Gewähltes Szenario
- *VESA*-Auflösungsvoreinstellung für die Kamera
- Kameraauflösung Breite
- Kameraauflösung Höhe
- Kamerasichtfeld
- Intervalle pro Sekunde
- Verzögerungstoleranz
- Simulationsgeschwindigkeit

(F06, F07, F12, F13, F18-F23, F27, F28, F59-F67, F72, F90R, F103R-F105R)

Einige der gespeicherten Optionen überschreiben andere wenn sie gesetzt werden. Das Einstellen einer eigenen Kameraauflösung verhindert, dass die *VESA*-Auflösungsvoreinstellung genutzt wird. Sollte ebenso ein Szenario angegeben sein, werden die Einstellungen für Karte, Zeit, Wetter und Nebel ignoriert und dem Szenario entnommen. (F25)

Für die Fahrzeug-, Karten- und Szenario-Dateien sind einfache Methoden ausreichend, denen ein gewünschter Name mitgegeben wird. Als Rückgabe werden dann die dem Namen zugehörigen Daten aus den jeweiligen Dateien gelesen und zurückgegeben. Für die Einstellungsdatei sind allerdings ausgeprägtere Werkzeuge erforderlich. Zuerst muss es eine Methode geben, die die Einstellungsdatei einliest und als Python-Dictionary zurückgibt. Jedoch muss es auch möglich sein, den gesamten Inhalt der Einstellungsdatei zu überschreiben. Diese Methode ist notwendig, sollte die Einstellungsdatei fehlen oder beschädigt sein. Ist die Einstellungsdatei beschädigt oder fehlt, wird eine im Config-Dateiinterface gespeicherte Standardkonfiguration verwendet, um die Einstellungsdatei

wiederherzustellen. Zusätzlich zu der Konfigurationsdatei wird eine Laufzeitkonfiguration geführt, die bei jedem Programmstart leer ist. Die Nutzenden können dann einstellen, welche Konfiguration sie nutzen möchten. (F73, F75-F77)

Damit nicht bei jeder Änderung die komplette Konfiguration überschreiben werden muss, existieren Methoden, um jede Option direkt anzusprechen. Diese Methoden beachten dabei, ob der Konfigurationsmodus auf Laufzeit oder Datei gesetzt ist. Weil die Laufzeitkonfiguration bei jedem Programmstart leer ist, wird bei nicht gesetzten Optionen die Konfigurationsdatei als Backup genutzt. Beim Setzen von einzelnen Optionen wird zusätzlich überprüft, ob die Typen der Eingabe mit den ursprünglichen Typen übereinstimmen. Diese Überprüfung soll bei einem Überschreiben der gesamten Konfiguration nicht nötig sein, da dies nur im Falle einer Regeneration geschieht.

Zusätzlich zu Konfigurations- und Inhaltsdateien liest das Config-Dateiinterface auch die Sprachdateien für die gewählte Anzeigesprache aus. Diese werden ebenfalls als *JSON*-Dateien abgespeichert mit jeweils anderen Dateinamen, die die Sprache repräsentieren. Dieser Ansatz erleichtert die Erweiterung der Sprachen in der Zukunft. (F71)

5.1.3 Benutzeroberflächen

Für das BeamTechEasyInterface werden eine textbasierte und eine grafische Benutzeroberflächen entwickelt. Beide ermöglichen die Konfiguration der Software und den Start der BeamNG.tech Simulation und sind funktional identisch. Die Texte beider Oberflächen werden in der selektierten Sprache vom Config-Dateiinterface geladen. (F68-F71)

Grafische Benutzeroberfläche

Wählen die Nutzenden die grafische Benutzeroberfläche, sehen sie als erstes das Hauptmenü, welches in Abbildung 10 dargestellt ist. Dort wird zunächst ein Hinweis angezeigt, wo sie sich eine BeamNG.tech Lizenz ausstellen lassen können und eine Übersicht welches Fahrzeug und welche Karte ausgewählt sind. Rechts davon befindet sich eine Ansammlung von vier Knöpfen, von denen die oberen Beiden die Simulation starten, der Dritte ein Einstellungsmenü öffnet und der Letzte, der das Programm und die Simulation beendet. Der erste der beiden Startknöpfe gibt dem Simulations-Connector den Befehl die Simulation mit dem Streaming-Modus zu starten. Die Schaltfläche darunter leitet die gleiche Anweisung an den Simulations-Connector mit dem Unterschied weiter, dass der Recording-

Modus genutzt wird. Die restlichen Einstellungen werden nicht verändert. (F140G, F143G)

Abbildung 10 Konzeptgrafik der Hauptansicht der grafischen Benutzeroberfläche

Dafür ist der dritte Knopf konzipiert, welcher ein Einstellungsmenü öffnet, in dem alle Einstellungen geändert werden können. Dort werden zunächst die alten Werte angezeigt, welche dann durch die Nutzenden überschrieben werden können. Zusätzlich überprüft die Logik des Einstellungsmenüs die Eingaben auf Typenkonformität und besitzt eine Schaltfläche, mit der die Einstellungen ausschließlich zur Laufzeit gespeichert werden. Um die Einstellungen zu bestätigen, gibt es einen dedizierten „Speichern"-Knopf, sie können aber auch durch das Schließen des Menüs oder das Drücken eines „Abbrechen"-Knopfs verworfen werden. (F141G, F142G)

Textbasierte Benutzeroberfläche

Die textbasierte Benutzeroberfläche ist als ein interaktiver Menü-Dialog gestaltet, der den Nutzenden mögliche Antworten und deren Bedeutung vorschlägt. Die Nutzenden antworten darauf mit der Funktion, die sie verwenden möchten. Eine dieser Funktionen zeigt den Nutzenden die aktuelle Konfiguration an. Diese Funktion berücksichtigt auch, ob aktuell eine Laufzeitkonfiguration genutzt wird und zeigt dort nicht gesetzte Einstellungen an. Eine weitere Funktion ermöglicht es den Nutzenden Einstellungen zu verändern. Dabei wird der aktuelle Wert insofern überprüft, ob der neue Wert typenkonform ist. Den

Nutzenden ist es ebenfalls damit möglich zu einer Laufzeitkonfiguration zu wechseln. Weitere Menüpunkte zeigen die kompatiblen Karten und Fahrzeuge an. (F133T, F134T) Zusätzlich existiert ein Menüpunkt, mit dem die Simulation mit den aktuellen Einstellungen gestartet werden kann. Wenn diese Option gewählt wird, überprüft die Oberfläche vor dem Start die gesetzten Pfade und fordert die Nutzenden zur Neusetzung dieser auf, sollten sie fehlerhaft sein. Ebenso werden die Nutzenden über eine abgelaufene BeamNG.tech Lizenz informiert und wo sie eine neue beantragen können. Abschließend schließt die letzte Funktion das Programm und die Simulation. (F135T, F139T)

5.1.4 Simulations-Connector

Der Simulations-Connector ist das Herzstück des gesamten Projektes und baut auf den Funktionen des 2022 entwickelten Prototypen auf. Die Kernfunktionen, wie das Starten

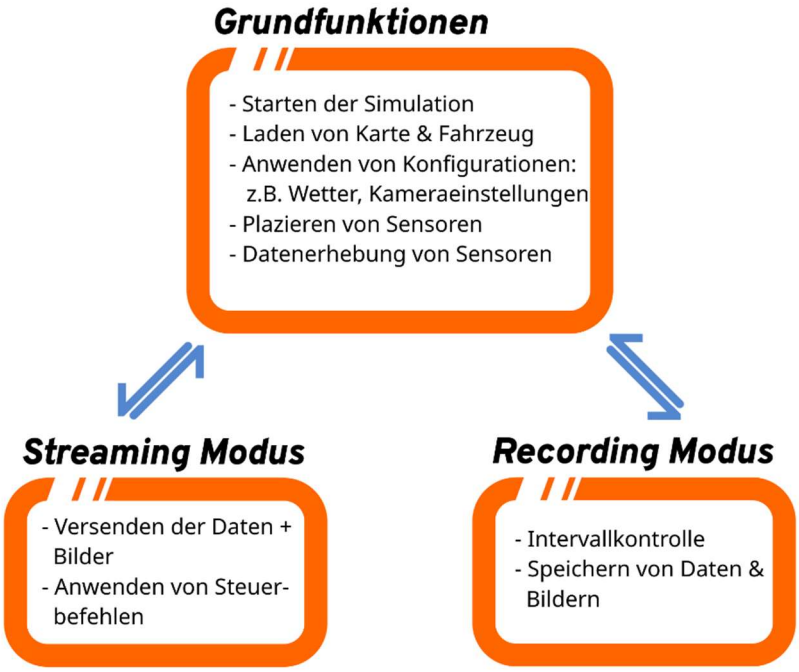

Abbildung 11 Konzeptgrafik Simulations-Connector Aufbau

der Simulation, Erheben von Daten und die Extraktion/Weitergabe, sind ebenfalls vorhanden. Sie wurden allerdings um mehrere Zusatzfunktionen erweitert und von der Handhabung verbessert. Es existieren nun die beiden Modi Streaming und Recording, die die gesammelten Daten unterschiedlich verarbeiten. Da die beiden Modi trotzdem auf viele gemeinsame Funktionen zugreifen müssen, wurde der Simulations-Connector in drei Teilen konzipiert. Die Grundfunktionen, die immer genutzt werden und die unterschiedlichen Datenausgabemodi. (F82)

Grundfunktionen

Die Grundfunktionen haben zusammengefasst drei Aufgaben: Das Starten der Simulation, das Anwenden der nutzerspezifizierten Konfigurationen und das Erheben der Fahrdaten. Um die BeamNG.tech Simulationssoftware starten zu können, werden das BeamNG Installationsverzeichnis und der BeamNG Nutzerordner vom Config-Dateiinterface angefragt. Mit diesen beiden Pfaden wird versucht die Simulation zu starten. Dabei können zwei Fehler auftreten: Der erste Fehler zeigt sich, wenn die Nutzenden das BeamNG Installationsverzeichnis falsch angegeben haben. Der weitere Fehler wird ausgelöst, wenn die genutzte Programm-Lizenz abgelaufen ist. Bei beiden Fehlern werden die Nutzenden, je nach eingestellter Oberfläche informiert und die Software geschlossen. (F02, F03)

Wenn BeamNG.tech erfolgreich gestartet wurde, muss das Fahrzeug zum Laden vorbereitet werden. Für das Fahrzeug werden vom Config-Dateiinterface das eingestellte Modell und die Kameraposition für dieses erfragt. Daraufhin wird das Fahrzeug geladen, sodass die Kamera an diesem befestigt werden kann. Um ein Kameraobjekt zu erstellen, benötigt dieses noch drei Parameter, die vom Config-Dateiinterface geladen werden: die Auflösung, in der die Bilder erzeugt werden, das Sichtfeld der Kamera und die Bildrate. Bei einer manuell angegebenen Auflösung, werden dabei die *VESA*-Voreinstellungen überschrieben. Nachdem das Kameraobjekt erfolgreich konfiguriert und erstellt wurde, wird es am Fahrzeug angebracht. Zusätzlich werden die Fahrdaten des Fahrzeuges durch das Anbringen eines Elektroniksensors zugänglich gemacht. (F29, F128S, F129S)

Anschließend wird die zu ladende Karte und das Fahrzeug mit seinem Startpunkt BeamNG.tech übergeben und die Simulation gestartet. Nachdem die Simulation vollständig geladen wurde, werden weitere Umgebungseinstellungen angewandt. Darunter in welcher Geschwindigkeit die Simulation laufen soll, die Simulation der jeweiligen Uhrzeit sowie des Wetters. Ebenfalls wird die eingestellte Nebeldichte an die Simulation übergeben. Die Initialisierungsphase der Simulation ist damit abgeschlossen und der gewählte Modus wird gestartet. (F04, F05, F11, F59, F63, F66)

Da beide Modi Fahrzeugdaten benötigen, ist die Erhebung dieser ebenfalls in den Grundfunktionen erhalten. Dies verhindert, dass Programmcode doppelt abgespeichert wird und erhöht dessen Lesbarkeit. Für die Erhebung und Abspeicherung der Fahrzeugdaten wird das *CSV*-Format gewählt. Da potenziell 60 Datensätze die Sekunde erhoben werden, würden andere Formate wie *JSON* zu einem höheren Datenüberhang führen. Ebenfalls lassen sich Datensätze im *CSV*-Format gut in Tabellenkalkulationsprogramme importieren. (F84R, F85R, F112S, F113S)

Von der Simulation werden folgende Daten aufgezeichnet:

- Simulierte Zeit (F56)
- Fahrtrichtung (F53)
- Position auf der gewählten Karte (F54)

Von dem am Fahrzeug angebrachten Elektroniksensor werden folgende Daten aufgezeichnet:

- Radgeschwindigkeit in m/s (F30)
- Individuale Radgeschwindigkeiten
- Gaspedalposition in % (Prozent)(F32)
- Gasstärke in % (Prozent) (F33)
- Bremspedalposition in % (Prozent) (F34)
- Bremsstärke in % (Prozent) (F35)
- Kupplungspedalposition in % (Prozent) (F36)
- Kupplungsstärke in % (Prozent) (F37)
- Stellung der Parkbremse (F38)
- Stärke der Parkbremse (F39)
- Motorstatus (An/Zündung/Aus) (F40)
- Motorlast in % (Prozent) (F41)
- Motordrehzahl (F42)
- Gang des Getriebes (F43)
- *ABS*-Status (An/Aus/nicht vorhanden) (F44)
- *ABS*-Aktivität (Passiv/Eingreifend) (F45)
- *ASR*-Status (An/Aus/nicht vorhanden) (F46)

- *ASR*-Aktivität (Passiv/Eingreifend) (F47)
- *ESP*-Status (An/Aus/nicht vorhanden) (F48)
- *ESP*-Aktivität (Passiv/Eingreifend) (F49)
- Scheinwerferstatus (F50)
- Blinker-Status (Links, Rechts, aus) (F52)
- Lagewinkel (F55)

Diese erhobenen Daten werden anschließend durch Kommas separiert und als Text zurückgegeben.

Modus Recording

Die Aufgabe des Recording-Modus ist es, die erhobenen Fahrzeugdaten und die Bilder der Frontkamera in festgelegten Intervallen abzuspeichern. Dafür nutzt es die aus den Grundfunktionen gestartete BeamNG.tech-Instanz und dessen Datenerhebungsfunktion. Bevor dies geschehen kann, muss der Recording-Modus zusätzliche Daten sammeln, die als Kopfzeile der zukünftigen Log-Datei genutzt werden. In dieser Kopfzeile werden zunächst die Spezifikationen des ausführenden Rechners und die Softwareversion der Simulation festgehalten. Darunter werden die Kategorien der aufzunehmenden Daten geschrieben, damit die aufgenommenen Werte von den Nutzenden und Programmen besser zugeordnet werden können. (F01, F84R, F86R-F88R)

Um die eingestellte Intervallrate, in der Daten erhoben werden sollen, zu erreichen und deren Einhaltung zu überprüfen, erfolgt eine Umrechnung von Intervallen pro Sekunde in Sekunden pro Intervall. Mit dieser Metrik wird verglichen, ob das letzte Intervall die festgelegte Intervallzeit überschritten hat, welches eine Warnung der Nutzenden zufolge hätte oder ob die Zeit eingehalten wurde. Dabei wird von dem Config-Dateiinterface die eingestellte Verzögerungstoleranz geladen und auf die Sekunden pro Intervall addiert. Zusätzlich zu der Warnung wird zudem die Geschwindigkeit der Simulation verringert. Beides geschieht ebenfalls sollte die Erhebung von Bildern verzögert geschehen. Dies kann überprüft werden, indem das letzte Bild mit dem aktuell erhobenen verglichen wird, da BeamNG.tech immer das intern zuletzt erhobene Bild zurückgibt. Kommt die Simulation nicht mit dem Generieren der Bilder hinterher, wird zwei Mal das gleiche Bild ausgegeben. Das Konzept zur Verzögerungserkennung ist zudem auch nochmal in Abbildung 12 dargestellt. (F89R, F90R, F96R-F101R, F103R, F104R)

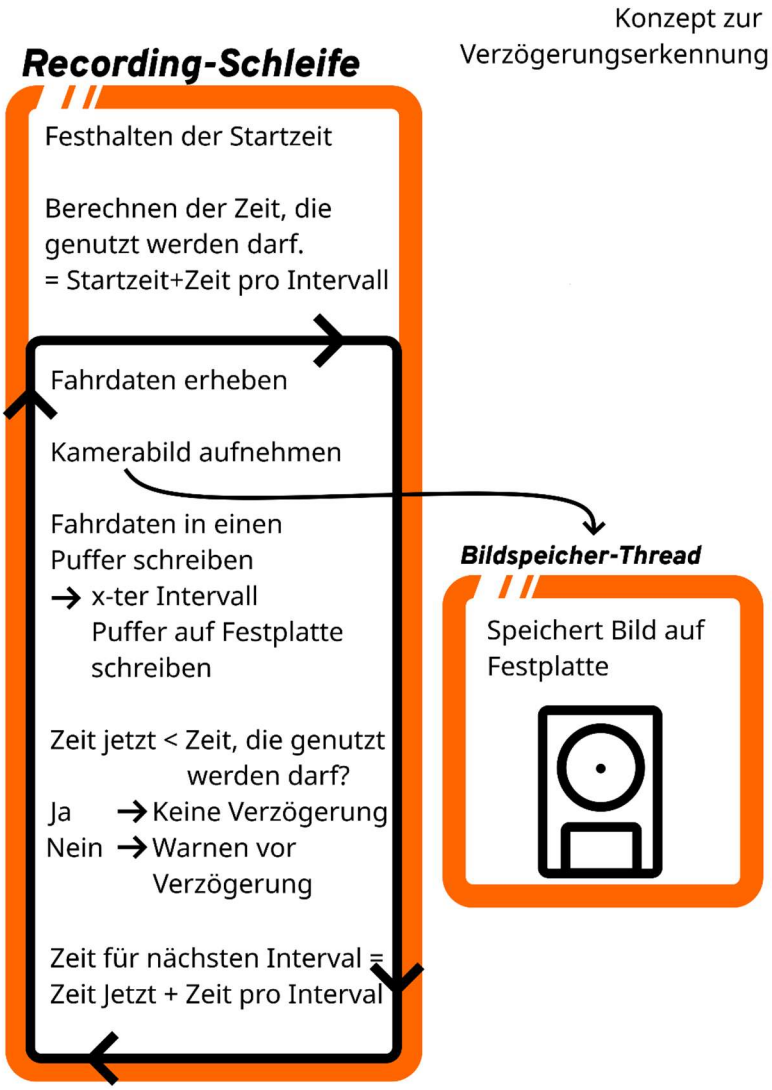

Abbildung 12 Konzeptgrafik Verzögerungserkennung Recording-Modus

Zusätzlich zu den durch die Grundfunktionen erhobenen Daten, erweitert der Recording-Modus die erhaltenen Informationen um die Echtzeit, die vergangene Zeit und die Intervallnummer. Die gezählte Intervallnummer wird ebenfalls für die Dateinamen der aufgenommenen Bilder genutzt, damit diese später dem Datenerhebungsintervall zugeordnet werden können. Da die Bilder in dem gleichen Schleifendurchlauf wie die Daten erhoben werden, wird auch sichergestellt, dass ein Bild mit der Intervallnummer x auch zum Intervall x gehört. Das Speichern der Bilder erfolgt in einem separaten Thread, damit diese ohne Leistungseinbußen komprimiert werden können. Die erhobenen Daten werden vorerst in einen Puffer geschrieben, welcher nur jeden x-ten Intervall in die erstellte Logdatei schreibt, um die Last auf das Dateisystem zu verringern. (F83R, F92R-F95R, F102R)

Modus Streaming

Der Streaming-Modus ist nach dem „Frage-Antwort"-Prinzip konzipiert. Programme, die den Streaming-Modus benutzen, können über die Schnittstelle Informationen anfragen, worauf eine Antwort mit den geforderten Daten gesendet wird. Über diesen Austausch können die Fahrdaten, die mit Hilfe der Grundfunktionen erhoben wurden, angefragt werden. Diese werden zusätzlich mit der Uhrzeit der Erhebung und der Intervallanzahl erweitert und versendet. Des Weiteren kann ein Bild der Frontkamera angefragt werden, welches daraufhin über die Schnittstelle verschickt wird. Zudem kontrolliert das Programm, ob das neuangefragte Bild sich von dem zuletzt gesendeten Bild unterscheidet. Sollte dies der Fall sein, werden die Nutzenden gewarnt, dass die eingestellte Datenerhebungsintervallrate nicht gehalten werden kann. (F107S-F111S, F114S-F118s)

Eine weitere Funktion des Streaming-Modus ist es, Steuerdaten empfangen und an das simulierte Fahrzeug weiterleiten zu können. Anders als bei den Datenanfragen sendet das verbundene Programm eine Anfrage, um Steuerdaten senden zu können. Darauf antwortet der Streaming-Modus nicht, sondern bereitet sich auf das Empfangen der Steuerdaten vor. Mit dem Versenden der Steuerdaten kann das verbundene Programm die Gas- und Bremseingaben verändern sowie den Lenkwinkel bestimmen. Dabei können alle drei Werte zusammen verändert oder die einzelnen Eingaben individuell angepasst werden. Nachdem die Steuerbefehle auf das simulierte Fahrzeug angewendet wurden, antwortet der Streaming-Modus mit einer Bestätigungsnachricht. Manuelle Eingaben, die die Nutzenden über BeamNG.tech direkt tätigen, überschreiben dabei die gesendeten Steuerdaten. (F119S-F127S)

Anders als der Recording-Modus kann der Streaming-Modus sich zusätzlich mit bereits laufenden Simulationen verbinden. Dafür verbindet sich dieser über den Standard-Kommunikations-Port mit BeamNG.tech und übernimmt das Fahrzeug mit dem Standardnamen. Von diesem Fahrzeug können dann ebenfalls Daten und Bilder erhoben und Steueranweisungen gesendet werden. Abschließend kann der Streaming-Modus über ein verbundenes Programm einen Schlussbefehl empfangen, nachdem die Simulation und das BeamEasyTechInterface geschlossen wird. (F130S-F132S)

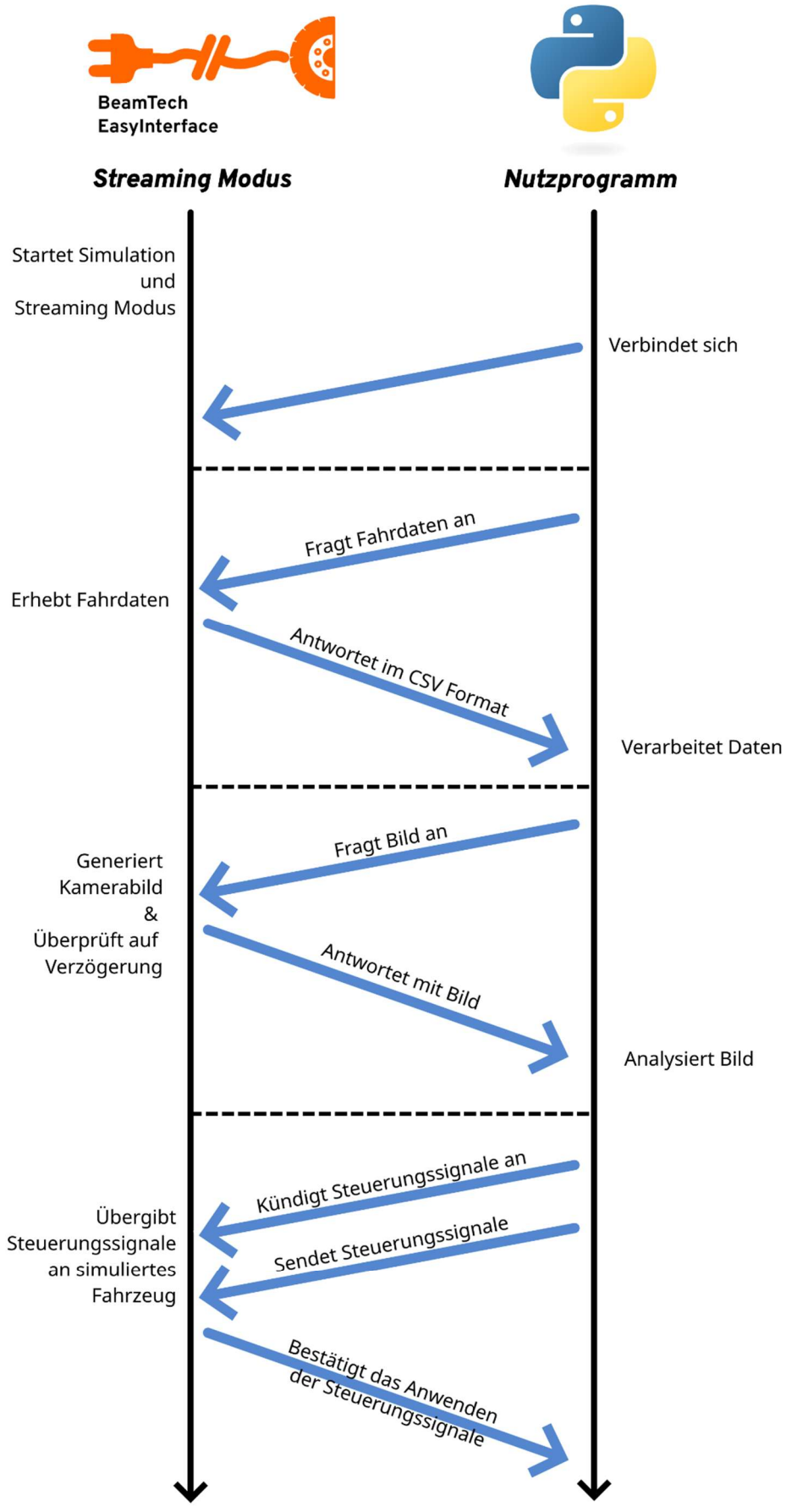

Abbildung 13 Kommunikationskonzept Modus Streaming

5.2 Nutzungskonzept

Zur Veranschaulichung der Nutzung des BeamEasyTechInterface durch Anwendende, wird im Nutzungskonzept ein Beispiel für die zukünftige Anwendung gezeigt. Bei dem beschriebenen Szenario handelt es sich um einen Studierenden, der von seiner Lehrkraft eine Hausaufgabe bekommen hat.

Der Studierende hat folgende Aufgabe gestellt bekommen:

„Nutzen Sie das BeamTechEasyInterface, um eine Python-Anwendung zu entwickeln, die Bilder einer Fahrzeug-Frontkamera in Echtzeit auf Verkehrsschilder analysiert. Sollte ein Verkehrsschild erkannt werden und das Fahrzeug zu schnell sein, bremsen Sie es automatisch auf die angegebene maximale Geschwindigkeit ab. Geben Sie den aktuellen Status ihres Programmes auf der Konsole aus.“

Der Nutzungsprozess des Studierenden:

Im ersten Schritt startet der Studierende das BeamTechEasyInterface. Daraufhin legt er entsprechende Pfade für die Installation, den betreffenden User und das Zielverzeichnis für die Recording-Dateien fest. Hiernach erfolgt die Einstellung der Umgebungsvariablen, für das Fahrzeug und die Karte, die er verwenden möchte. Der nächste Schritt beinhaltet die Konfiguration der verwendeten Fahrzeugkamera bei Auflösung, Sichtfeld und Datenerhebungsintervall. Der Studierende entscheidet sich für eine Auflösung von 1280x720p, ein Sichtfeld von 40° und eine Aufnahmerate von 20 Bildern die Sekunde. Ist dieses erfolgt, nimmt der Studierende unter Verwendung des Recording-Modus zunächst eine erste Fahrt zu Testzwecken auf. Mit den erhobenen Daten entwickelt er einen ersten Prototyp, der die erkannten Verkehrsschilder auf der Konsole ausgibt. Durch den Prototyp versteht der Studierende die Grundlagen der Bildverarbeitung zur Verkehrsschilderkennung. Die dadurch erreichten Erkenntnisse werden nun zur Entwicklung eines Programms, welches in der Lage ist, in Echtzeit über den Streaming-Modus Verkehrsschilder zu erkennen, genutzt. Zusätzlich erweitert er das Programm, um die Radgeschwindigkeit des simulierten Fahrzeugs zu analysieren. Diese wird anschließend mit der erkannten Geschwindigkeitsbegrenzung verglichen und bei einer Überschreitung ein Bremssignal über die Streaming-Schnittstelle versendet.

5.3 Konzept Spurhalteassistenz

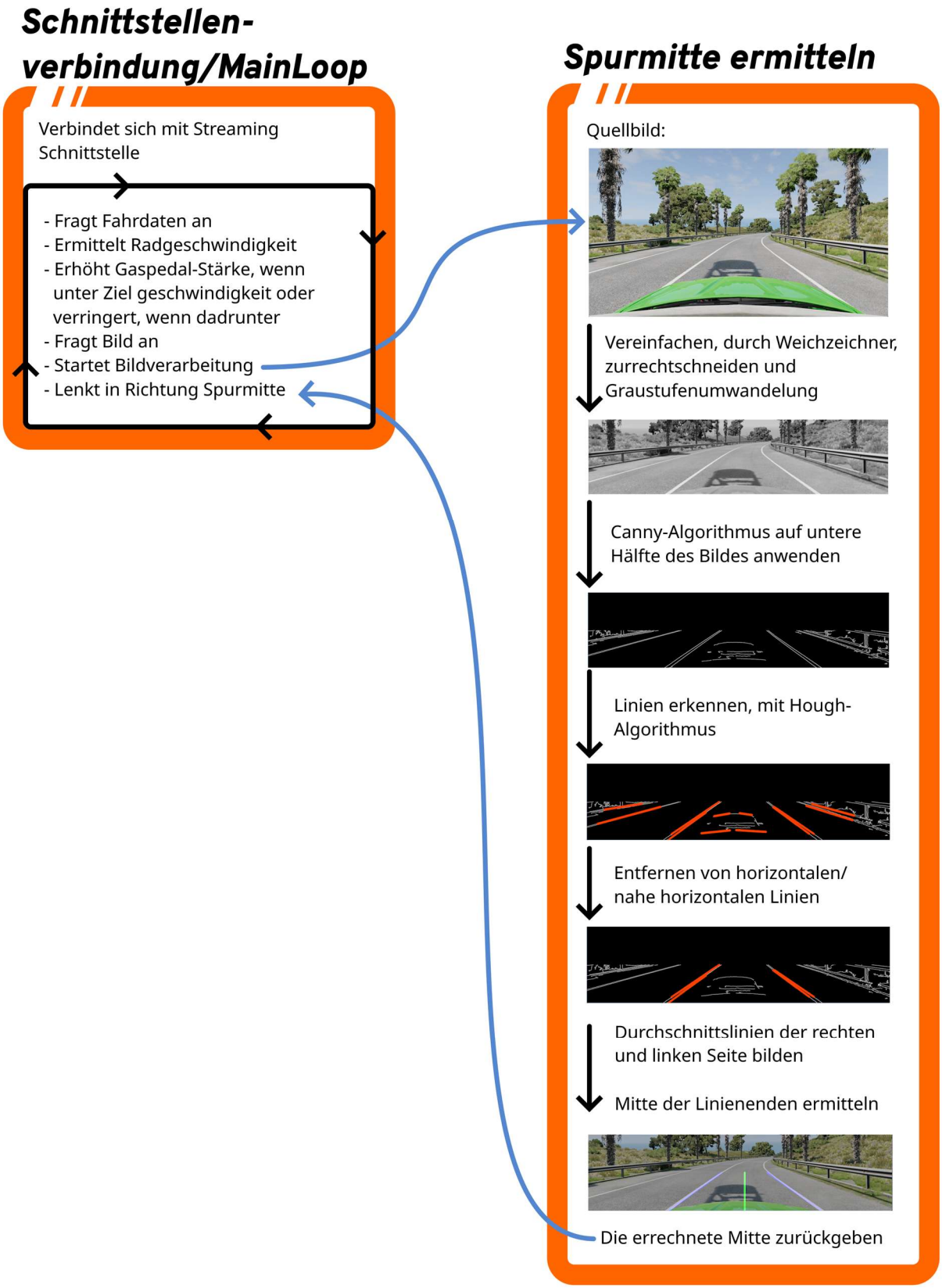

Abbildung 14 Konzept des Bildverarbeitungsvorganges des Spurhalteassistenten

Da der Spurhalteassistent als Funktionsbeweis für das entwickelte Schnittstellenprogramm dient, muss dieser mit den betreffenden Funktionen arbeiten, um ein zukünftiges Anwendungsszenario umzusetzen. Da es sich allerdings nur um einen Funktionsbeweis und nicht um ein auszulieferndes Produkt handelt, können bei der Ergebnisqualität Abstriche, bei z.B. der Handhabung, gemacht werden.

Um die Kommunikationslogik und die Bildverarbeitung zu trennen, wird der Spurhalteassistent in zwei Teilen entwickelt: zuerst die Schnittstellenverbindung, die Daten mit dem Streaming-Modus austauscht und zuletzt ein Bildverarbeitungsmodul, dass ein Bild als Eingabe erhält und die Mitte der Spur ausgibt.

Schnittstellenverbindung

Wie in Abbildung 14 links zu sehen ist, verbindet sich die Schnittstellenverbindung zunächst mit der Streaming-Schnittstelle des Hauptprogrammes. Nach dem eine Verbindung hergestellt wurde, wird in eine Schleife eingetaucht, in der Daten angefragt, verarbeitet und mit Steuereingaben geantwortet wird. In dieser Schleife existieren zwei Variablen, die über die Durchläufe hinweg modifiziert werden, von denen eine später zur Kontrolle des Gaspedals und die andere zur Kontrolle der Lenkung genutzt wird. Zusätzlich zu den Lenkanweisungen, um das Fahrzeug zur Mitte der Spur zu lenken, werden die Gaspedalanweisungen genutzt, um eine konstante Geschwindigkeit zu halten. Dies verringert die Komplexität der Anwendung, da sie keine Rücksicht auf variable menschliche Eingaben machen muss. (F02)

In der Schleife selbst werden zuerst die Fahrdaten vom Schnittstellenprogramm angefragt und empfangen. Aus den Fahrdaten wird anschließend die Radgeschwindigkeit ermittelt und mit einem Sollwert verglichen. Wenn die Radgeschwindigkeit vom Sollwert abweicht, wird die Gaseingabe entsprechend angepasst. Daraufhin wird ein Bild der Frontkamera angefragt und an die Bildverarbeitung weitergegeben. Kann die Bildverarbeitung die Spurmitte erkennen, wird diese zurückgegeben. Anschließend wird verglichen, ob die Spurmitte von der Bildmitte abweicht und die Lenkeingabe dementsprechend angepasst. Dabei wird davon ausgegangen, dass die montierte Frontkamera mittig am Fahrzeug angebracht wurde. Abschließend werden die berechneten Steuereingaben über die Streaming-Schnittstelle zurückgesendet und die Schleife beginnt von vorn. (F03, F07, F08)

Bildverarbeitung Spurerkennung

Die Bildverarbeitung ähnelt, vom Konzept her, dem *LDA*-Modul aus dem Jahr 2022, berechnet jedoch die Spurmitte und nicht den Fluchtpunkt der Fahrbahnbegrenzungen. Dieser Ansatz wird gewählt, um die Implementationskomplexität möglichst gering zu halten. Als erster Schritt wird das Quellbild vereinfacht. Dafür werden ungebrauchte Teile des Bildes, wie beispielsweise der Himmel und die Motorhaube, weggeschnitten. Ebenfalls wird das Bild von einem Farbbild in ein Graustufenbild umgewandelt und mit einem Weichzeichner eventuelles Rauschen entfernt. Dies ist bei echtzeitgenerierten Computergrafiken besonders wichtig, da es dort zu Artefakten von z.B. Antialiasing oder niedrig aufgelösten Schatten kommen kann. Von diesem vereinfachten Bild wird nun nochmals die obere Hälfte abgeschnitten und die Canny-Transformation[2] zur Kantenerkennung angewendet. Es wird dabei nur die untere Hälfte des Bildes genutzt, um sich auf die Straße unmittelbar vor dem Fahrzeug zu konzentrieren. Die ermittelten Kanten werden nun auf einem separaten Bild eingezeichnet, um sie von dem Original zu trennen. Auf dieses Kanten-Bild wird anschließend die Hough-Linien-Transformation[3] angewandt, um eine Liste von erkannten Winkeln zu erhalten. Die Hough-Linien-Transformation[3] wird dabei so konfiguriert, dass sie nah aneinander liegende Linien kombiniert, um auch unterbrochene Fahrbahnbegrenzungen zu erkennen. Da diese Liste auch Geraden enthält, die horizontal oder nahezu horizontal sind, werden Geraden mit unrealistischen Winkeln aussortiert. Die Linien werden zusätzlich in rechte und linke Linien unterteilt. Daraufhin wird von beiden Seiten eine Durchschnittslinie berechnet, die eine Annäherung der jeweiligen Spurenbegrenzung abbilden soll. Zum Schluss wird der Mittelpunkt der beiden Linienenden berechnet. Dieser Punkt repräsentiert die Mitte der Spur und wird an die Hauptschleife zurückgegeben. (F04, F05, F06)

6 Umsetzung

Im Kapitel der Umsetzung wird die Realisierung des zuvor ausgearbeiteten Konzeptes behandelt. Dabei wird der Aufbau der entwickelten Software beschrieben und wie geforderte Funktionen implementiert wurden. Für die erschaffene Schnittstellensoftware wird die Logik des Programmes getrennt von den Nutzeroberflächen betrachtet. Abschließend wird die Implementation des Spurhalteassistenten vorgestellt und bewertet, ob mit diesem die Funktion des Schnittstellenprogramms gezeigt werden konnte. Der umgesetzte Programmcode ist im Git-Lab der Technischen Hochschule Wildau unter folgender Adresse zu finden. https://git.th-wildau.de/maha2541/beamtecheasyinterface

Abbildung 15 Logo des entwickelten BeamEasyTechInterface

6.1 Schnittstellensoftware Logik und Aufbau

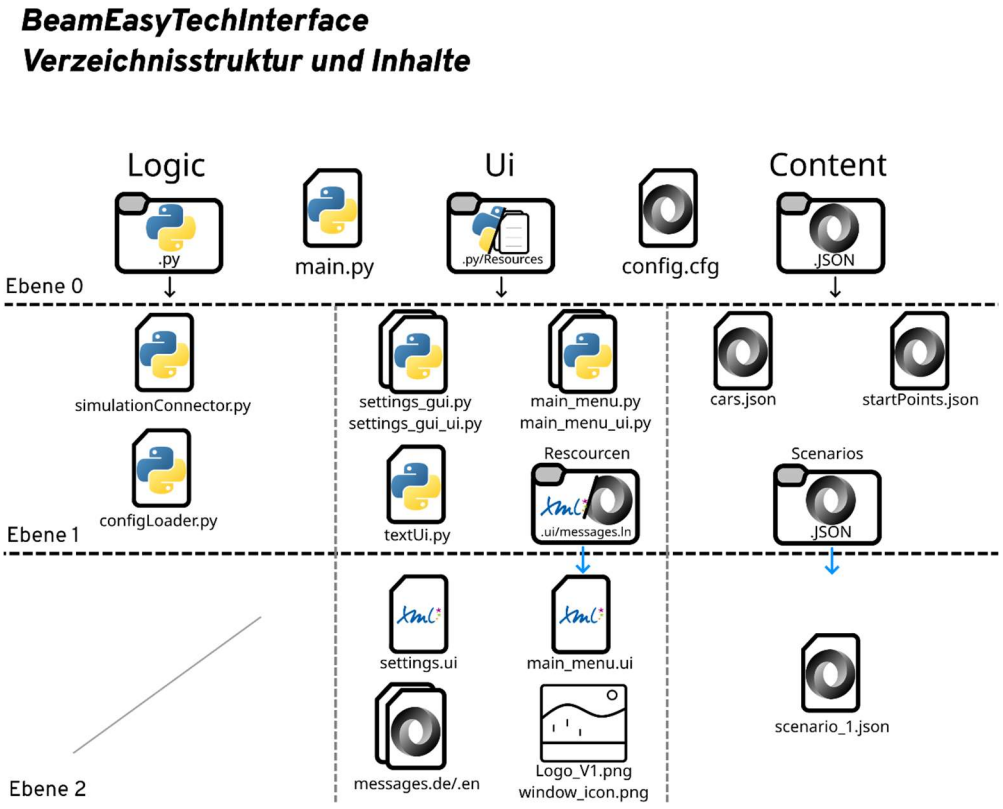

Abbildung 16 Übersichtsgrafik der angefertigten Verzeichnisse und Dateien

6.1.1 Simulationsinteraktion

Für die Interaktion mit der Simulationssoftware BeamNG.tech wurde das Python Programm „simulationConnector.py“ entwickelt. Dieses arbeitet mit der BeamNG.py Bibliothek, welche die verschiedenen Elemente der BeamNG.tech Software, als Python-Objekte zugänglich macht.

6.1.1.1 Verbinden, Konfigurieren und Starten der Simulation

Damit sich der „simulationConnector.py“ mit der BeamNG.tech Simulation verbinden kann, wird zunächst ein BeamNGpy-Objekt erstellt. Dieses liest die Installations- und Nutzerverzeichnisse sowie eine Verbindungsadresse als Parameter ein. Dem Objekt kann anschließend der Befehl gegeben werden, die Simulation zu starten.

```
42. beamNG = BeamNGpy(
        'localhost',
        64256,
        home=get_single_conf("beam_home"),
        user=get_single_conf("beam_user"))
43. beamNG.open()
```

Codeausschnitt aus dem „simmulationConnector.py“

Um nun eine Simulation zu starten, muss ein Scenario-Objekt erstellt werden, welches als Argumente die zu ladende Karte und einen Namen benötigt. Als Name wird der statisch programmierte Name „main_Scenario“ verwendet und die eingestellte Karte über den „configLoader.py“ geladen. Anschließend wird ein Vehicle-Objekt erstellt, dass das simulierte Fahrzeug darstellt. Zusätzlich werden an dem Vehicle-Objekt der erste von zwei Sensoren platziert. Dieser Sensor, welcher die Fahrdaten des Fahrzeuges sammelt, ist ein Electrics-Objekt, welches über den Methodenaufruf „`vehicle.attach_sensor('electrics', electrics)`“ angebracht wird. Daraufhin wird das Vehicle-Objekt dem Scenario-Objekt, inklusive Startpunkt hinzugefügt. Der Startpunkt wird dabei für jedes Fahrzeug individuell angepasst, damit diese nicht innerhalb des Bodens erscheinen. (Siehe Abbildung 17)

Abschließend kann das Szenario kompiliert werden und über das BeamNGpy-Objekt geladen werden. Dies startet die Simulation.

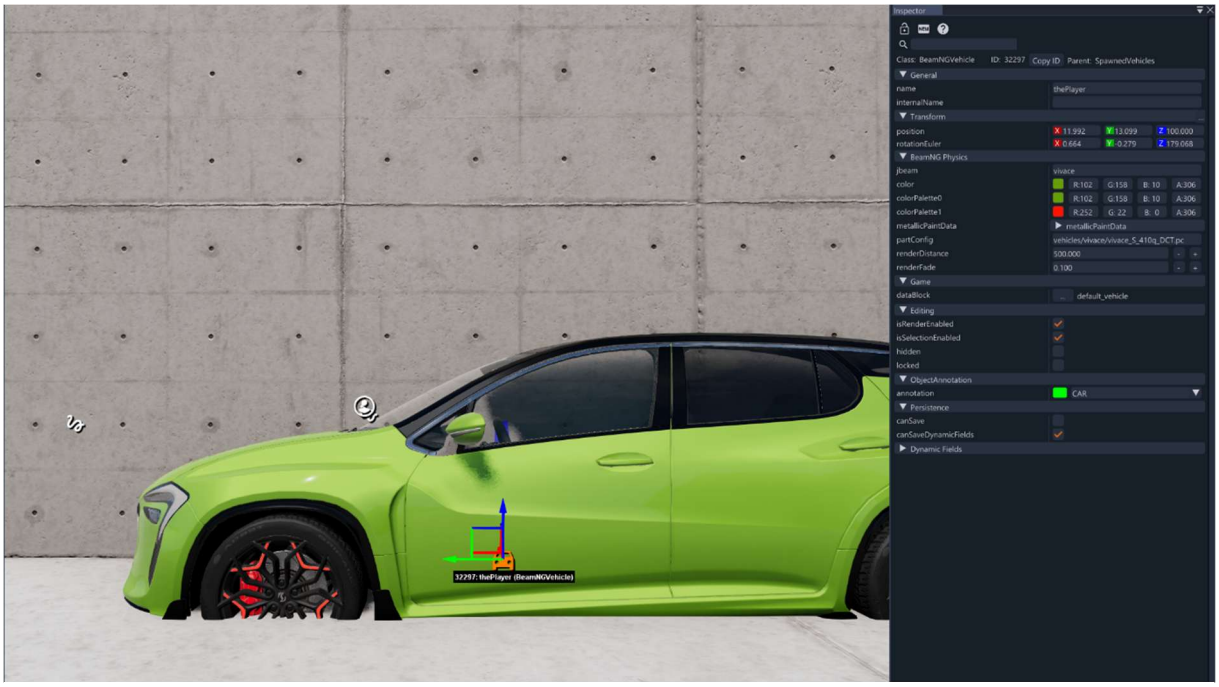

Abbildung 17 Fahrzeug auf Z-Koordinate 0 ohne Höhenanpassung

Einige Konfigurationen und Objekte können erst nach dem Start der Simulation angewandt werden. Darunter das Camera-Objekt, welches die Bilddaten zugänglich macht. Anders als bei dem Electrics-Sensor, wird die Verbindung zum Fahrzeug durch die Mitgabe des Vehicle-Objekts bei der Definition der Kamera hergestellt. Zudem werden bei der Erstellung auch die fahrzeugspezifische Position und die eingestellte Konfiguration der Kamera mitgegeben. Ebenfalls werden weitere Parameter, wie die Render-Art und die Aktualisierungsrate, mit der die Kamera versucht Bilder vorzuladen, festgelegt.

```
59. camera = Camera(
60.             name="main_cam",
61.             bng=beamNG,
62.             vehicle=vehicle,
63.             pos=camPos,
64.             dir=camRot,
65.             field_of_view_y=get_single_conf("sim_cam_fov"),
67.             resolution=cam_res,
68.             is_render_colours=True,
69.             is_using_shared_memory=True,
70.             is_streaming=True,
71.             is_render_instance=True,
72.             near_far_planes=(0.1,500),
73.             is_static=False,
74.             requested_update_time=log_time)
```

Codeausschnitt der Kameraerstellung aus dem „simmulationConnector.py“ Zeile 59-74

Anschließend wird über die Umgebungsschnittstelle des BeamNGpy-Objekts das Wetter und die Uhrzeit festgelegt. Danach wird der gewählte Modus gestartet. Sollten das Nutzer- oder Installationsverzeichnis falsch eingestellt sein, wird der Nutzer – je nach ausgewählter Benutzeroberfläche – über ein kleines Fenster oder eine Textausgabe darüber informiert. Eine andere Warnung wird ausgelöst, sollte die BeamNG.tech Lizenz ausgelaufen sein.

6.1.1.2 Erheben und Formatierung von Daten

Das BeamTechEasyInterface extrahiert zwei unterschiedliche Typen an Daten aus der BeamNG.tech-Simulation. Die textbasierten Daten werden hauptsächlich mit der Methode

```
get_current_car_data(beamNG: BeamNGpy ,vehicle : Vehicle, sensors : Sensors ,log_time :float)
```

erhoben. Diese löst nach dem Aufruf ein Aktualisieren der Fahrzeugsensoren aus und speichert deren Werte in einer String-Variable, die wie eine *CSV*-Datei formatiert ist. Umgebungsdaten, wie die simulierte Uhrzeit werden vom BeamNGpy-Objekt angefragt. Die Fahrzeugposition und -richtung, werden ebenfalls ohne die Nutzung von Sensoren aufgenommen, indem das Fahrzeug-Objekt diese selbst ausgibt.

Aus der Richtung, der Zeit seit dem letzten Datenerhebungsintervall und dem letzten Gier-Winkel[6] werden anschließend die aktuellen Lagewinkel des Fahrzeuges berechnet. Der Programmcode dafür, welcher aus dem Sommersemester 2022 Prototypen stammt, wurde von Prof. Dr. Alexander Kleinsorge verfasst und ist auch dementsprechend markiert.

Anschließend werden die Werte des angebrachten Elektroniksensors abgefragt und der String-Variable angehangen. Gleitkommazahlen werden dabei auf zwei Stellen gerundet. Für die Fahrassistenzprogramme muss zusätzlich ein Schlüsselfehler abgefangen werden, welcher geworfen wird, falls eine Technologie nicht vorhanden ist. Dieses Verhalten spricht gegen die Dokumentation, nach der bei einem Nichtvorhanden sein des z.B. *ESP*s eine Null zurückgegeben werden soll. Zusätzlich wird von dem Programm abgefangen, wenn die Verbindung zur Simulation abbricht oder beendet wird. Daraufhin wird der String-Variable der Text „end of sim“ angehangen. Abschließend wird die String-Variable mit den gesammelten Daten an die ausführende Methode zurückgegeben.

6.1.1.3 Datenspeicherung im Recording-Modus

Der Recording-Modus wird durch die Methode "`start_log(log_time :float ,vehicle : Vehicle ,camera :Camera, beamNG: BeamNGpy)`" umgesetzt. Diese bekommt zum Start, neben den erstellten Simulationsobjekten, auch eine „log_time" als Parameter. Diese Log-Zeit beschreibt, wie viel Zeit für einen Datenerhebungsintervall aufgebracht werden darf. Als Beispiel beträgt die Log-Zeit für 30 Aufnahmen pro Sekunde 33 Millisekunden. Zusätzlich werden die Einstellungen Verzögerungstoleranz, Speicherort für Logs und Speicherort für Bilder geladen.

Alle erhobenen Daten werden, bevor sie in eine Datei geschrieben werden, in die String-Variable „log_buffer_string" geschrieben. Dieser Buffer wird zunächst mit einer Legende über die gesammelten Daten erweitert. Anschließend werden die Systemspezifikationen über das BeamNGpy-Objekt angefragt und in den „log_buffer_string" geschrieben. Die letzte Zeile ohne Simulationsdaten ist die genutzte Konfiguration des BeamEasyTechInterfaces.


```
{'power': {'batteryFullLifeTimeSecs': -1.0, 'batteryCharging': True, 'ACOnline': True, 'batteryState': 'high', 'batteryLifeTimeSecs': -1.0, 'batteryPresent': True}, 'tech': True, 'gpu': {'memoryMB': 3938.0, 'version': '', 'name': 'NVIDIA GeForce GTX 1650 (D3D11)'}, 'type': 'GetSystemInfo', 'os': {'buildNumber': 26100.0, 'versionMinor': 0.0, 'shortname': 'Microsoft Windows 11 ', 'type': 'windows', 'fullname': 'Microsoft Windows 11  (v10.0) (build 26100),1.64-bit', 'gamearch': 'x64', 'bits': 64.0, 'versionMajor': 10.0}, 'cpu': {'coresLogical': 16.0, 'features': 'SSE,SSE2,SSE3,SSSE3,SSE4.1,SSE4.2,SSE4a,AVX,AVX2,SHA,BMI1,BMI2', 'arch': 'x64', 'measuredSpeed': 3.1939213275909424, 'l2Count': 8.0, 'numaCount': 1.0, 'packageCount': 1.0, 'coresPhysical': 8.0, 'name': 'AMD Ryzen 7 5800H with Radeon Graphics', 'l1Count': 16.0, 'l3Count': 1.0, 'vendor': 'AuthenticAMD', 'clockSpeed': 3194.0}}
```


Gesammelte Systemspezifikationen des Testsystems Laptop

Nach dem der „log_buffer_string" vorbereitet wurde, werden die Dateinamen für die Bilder und die Log-Datei festgelegt. In beiden Dateinamen wird der Zeitpunkt der Aufnahme festgehalten und die Bilder später um den Logintervall erweitert. Dies ist notwendig, um die Bilder den Fahrdaten zuzuordnen. Ebenfalls wird die Startzeit festgehalten und daraufhin die Datenerhebungsschleife gestartet.

In der Datenerhebungsschleife wird zunächst ein neuer Log-String erstellt, welcher zur Speicherung der Daten dieses Durchlaufes genutzt wird. Daraufhin wird die Nummer des aktuellen Intervalls und die vergangene Zeit im Log-String gespeichert. Folgend wird die Methode „get_current_car_data(…)" aufgerufen und die erhobenen Fahrdaten in den String geschrieben. Um die Last auf das Dateisystem zu verringern, wird der Log-String

zunächst an den „log_buffer_string“ angehangen. Dieser wird alle 100 Schleifenaufrufe geleert und der Inhalt in die Log-Datei geschrieben.

Für die Aufnahme und das Speichern der Frontkamerabilder wird anschließend ein neuer Thread geöffnet, der parallel zur Datenerhebungsschleife läuft. In diesem Thread wird die Methode „`write_to_FS(camera: Camera, recordings_path: str, frame_count: int):`“ aufgerufen, welche zunächst das Camera-Objekt nach dem neusten Bild abfragt. Dieses Bild wird anschließend mit dem zuletzt empfangenden Bild verglichen, um mögliche Verzögerungen zu erkennen. Wird erkannt, dass die beiden Bilder gleich sind, wird eine Verzögerung festgestellt und die Nutzenden werden gewarnt. Folgend wird das neue Bild in den globalen Speicher des alten Bildes kopiert und von einem PIL-Image (Bildformat, dass von BeamNG.py genutzt wird) in ein *CV2*-Bild des richtigen Farbraumes umgewandelt. Das umgewandelte Bild wird anschließend als *.jpg* komprimiert abgespeichert.

In der Datenerhebungsschleife wird währenddessen überprüft, ob es zu einer Verzögerung in der Erhebung der Textdaten kam. Dafür wird die Startzeit des Intervalls mit der „log_time“ aufaddiert und mit der aktuellen Zeit verglichen. Ist die aktuelle Zeit kleiner als die gewährte Zeit, wird die restliche Zeit des Intervalls gewartet. Das Warten ist notwendig, damit die Abstände zwischen den Datenerhebungsintervallen gleich bleiben. Ist die gemessene Zeit größer als die erlaubte Zeit, liegt aber noch innerhalb der Verzögerungstoleranz, wird der nächste Schleifendurchlauf sofort gestartet. Ist mehr Zeit verstrichen, als die „log_time“ und die Verzögerungstoleranz genehmigt, werden die Nutzenden gewarnt und ein Vermerk für diesen Logintervall in die Datei geschrieben. Daraufhin startet der nächste Schleifendurchlauf.

Bricht die Verbindung zur Simulation ab, werden die restlichen Daten aus dem „log_buffer_string“ an die Log-Datei angehangen, damit keine Daten verloren gehen. Ebenfalls werden die Nutzenden darüber informiert.

6.1.1.4 Datenaustausch im Streaming-Modus

Der Datenaustausch des Streaming-Modus, mit dem Fahrdaten und Bilder gesendet sowie Steuereingaben empfangen werden, wurde zuerst mit einem Webserver, der Nachrichten über Get-Anfragen austauscht, realisiert. Es stellte sich jedoch heraus, dass durch den Nachrichtenüberhang keine 30 Nachrichten in der Sekunde übertragen werden konnten.

Deswegen bietet der Streaming-Modus eine offene Socket[4]-Schnittstelle, welche deutlich performanter ist. Diese ist lokal über den Port 64300 erreichbar. Sollte sich ein Klient erfolgreich mit dem Server verbinden, werden die Nutzenden benachrichtigt und eine Schleife zur Nachrichtenverarbeitung gestartet.

In der Schleife werden einkommende Nachrichten empfangen und in eine String-Variable umgewandelt. Enthält die empfangene Nachricht ein „request_data“, wird die Methode „`get_stream_data(last_data_stream_time : float, stream_data_interval_count :int)`“ gestartet. Diese erweitert die Daten der „get_current_car_data(…)“ Methode, um die aktuelle Uhrzeit und gibt anschließend eine String-Variable im *CSV*-Format zurück. Diese wird daraufhin über die Socket[4]-Verbindung verschickt.

Alternativ wird auf den Befehl „request_img“ mit dem Versenden eines Bildes geantwortet. Dieses wird, wie im Recording-Modus, umgewandelt und komprimiert. Daraufhin wird das Bild, mit Hilfe der Pickle-Bibliothek von einem Python-Objekt in ein Byte-Array umgewandelt. Dieses wird anschließend über den Socket[4] , gefolgt von einem „EOF“, verschickt. Das „EOF“ („End Of Field“, englisch für Ende des Bereiches) markiert dem Empfänger, dass das Bild vollständig übertragen wurde. Dies ist notwendig, da die Bilder, je nach Inhalt, unterschiedliche Größen haben und in mehreren Teilen eingelesen werden. (Siehe 6.4.1)

Wird der Befehl „send_control“ empfangen, bereitet sich der Streaming-Modus auf das Empfangen von Steuerdaten vor. Nachdem diese eingetroffen sind, wird der dekodierte Inhalt an die Methode „`send_control (control_message : str)`“ weitergegeben. Da die Nachricht mit den Steuerdaten wie ein *JSON* formatiert ist, wird diese in ein Python-Dictionary umgewandelt. Anschließend werden hintereinander die Werte der Kategorien Lenkung, Gaspedal und Bremse ausgelesen und an das Vehicle-Objekt weitergegeben. Dieses steuert anschließend das simulierte Fahrzeug in BeamNG.tech.

Abschließend wird beim Empfangen der Nachricht „quit“ das Programm und die Simulation beendet.

6.1.2 Konfigurations- und Inhaltsmanagement

Die Konfiguration und Inhaltsinformationen werden vom „configLoader.py“ geladen und verwaltet. Sie sind im *JSON*-Format angelegt, da dies mit dem Python-Dictionary Datentyp kompatibel ist und damit die Inhalte direkt über den Namen angesprochen werden können. Dieses Verhalten wird über die Laufzeit wiederholt genutzt, um z.B. Werte der Konfiguration zu ändern oder Informationen über ein Fahrzeug zu erhalten.

```
68 def set_single_runtime_conf(setting_name:str, new_value:any):
69     runtime_Configuration[setting_name] = new_value
```

Codeausschnitt bei dem ein Wert der Laufzeitkonfiguration über den Einstellungsnamen geändert wird | configLoader.py Zeile 68-69

Die Laufzeitkonfiguration, das Auflösungsverzeichnis und die geschützte Standardkonfiguration, die nicht in Dateien abgespeichert sind, werden ebenfalls durch Python-Dictionarys im „configLoader.py“ abgebildet. Dies ist so realisiert, damit diese vor den Nutzenden geschützt und nicht zu verändern sind. Dies ist besonders bei der Standardkonfiguration wichtig, da diese genutzt wird, um eine beschädigte oder fehlende Konfiguration zu ersetzen.

Eine weitere Einstellung determiniert, ob das Programm „configLoader.py“ die Laufzeitkonfiguration oder die Dateikonfiguration verändern oder laden soll. In dem „configuration_write_mode“ wird festgehalten, welche der beiden Konfigurationen aktiv genutzt wird.

```
 1. default_configuration = { #is used to restore the config file
 2.     "language" : "en",
 3.     "configuration_write_mode" : "configuration_file",
 4.     "beam_home" : None,
 5.     "beam_user" : None,
 6.     "recordings_path" : None,
 7.     "logging_path" : None,
 8.     "ui" : "GUI", #GUI (graphic) or CLI (commandline)
 9.     "mode" : "recording", #recording or streaming
10.     "sim_speed" : 100, #percent 1-200
11.     "resolution_preset" : "hd", #is used if no width or hight are 0
12.     "sim_resolution_width" : 0, #pixel
13.     "sim_resolution_hight" : 0, #pixel
14.     "sim_cam_fov" : 60, #degrees
15.     "slowdown_tolerance" : 10, #ms
16.     "frame_rate": 30, #frames per second
17.     "use_scenario" : "no", #yes/no | if no, interface will try to connect to running sim
18.     "scenario" : "scen_1", # possible scenarios: TODO
19.     "car" : "etk800", # possible cars: TODO
```

```
20.     "weather" : "sunny", #sunny, rainy, snow
21.     "fog" : 50,
22.     "time" : "12:00:00",
23.     "map" : "grid" #is not used, when scenario is selected
24. }
```

Geschützte Standardkonfiguration aus dem „configLoader.py“

Weitere Inhalte, die vom „configLoader.py“ geladen werden, sind die zusätzlichen Informationen für unterstützte Fahrzeuge. Diese befinden sich in der Datei „cars.json“ und beinhalten neben der Kameraposition auch eine Höhenverschiebung für das Erscheinen der Fahrzeuge. Beide Werte werden mithilfe des BeamNG.tech Welteneditors auf der Karte Gridmap V2 bestimmt. Auf dieser gibt es mehrere 8m^3 Würfel, auf deren Seiten Abstände von 25cm markiert sind. Über ein Bild mit den Würfeln als Referenz und dem Fahrzeug, wird in dem Bildbearbeitungsprogramm Paint.Net die Kameraposition vor dem Rückspiegel ermittelt. (Siehe Abbildung 18)

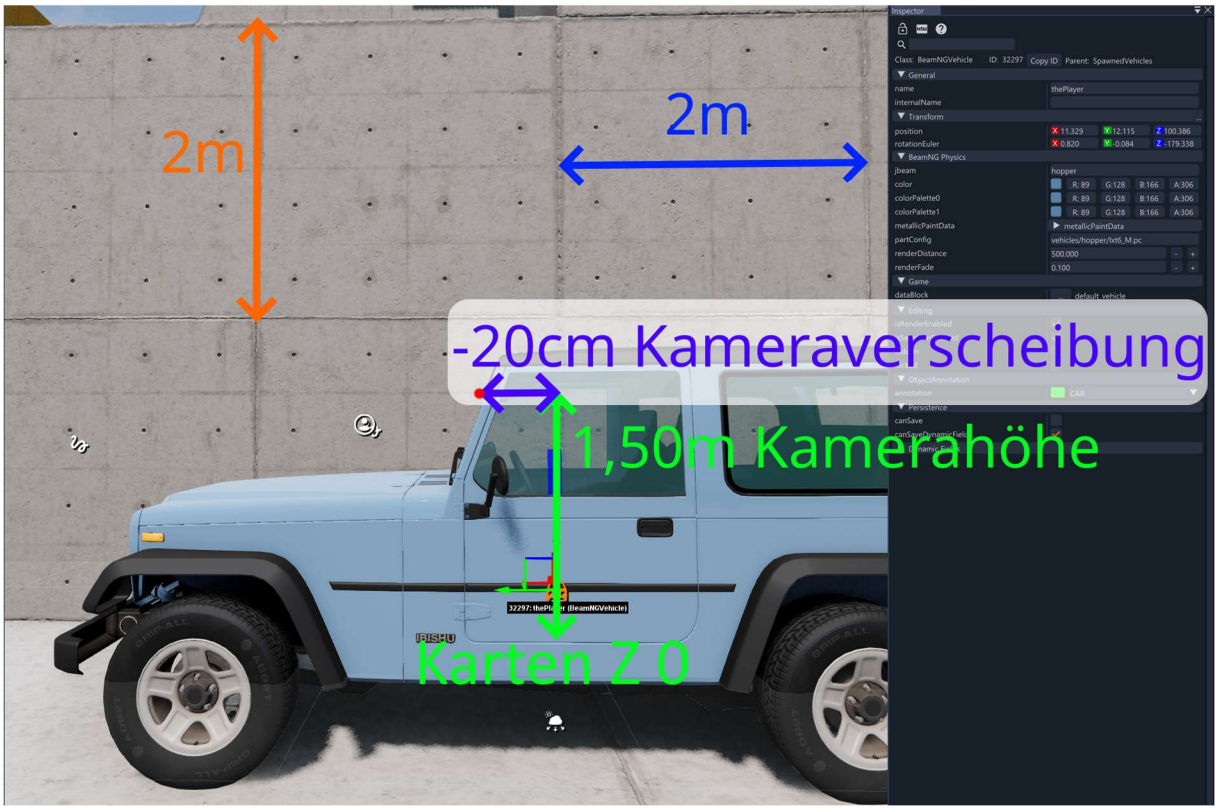

Abbildung 18 Ermittlung der Kameraposition mit Hilfe des Welteneditors

Die Höhenverschiebung verhindert ein Erscheinen des Fahrzeuges im Boden, wie es in Abbildung 17 dargestellt ist. Dies tritt auf, da der Nullpunkt der Höhe des Fahrzeuges am untersten Punkt des Chassis liegt. Die Höhe der ausgefederten Achsen und die Hälfte der Räder ergeben die Höhenverschiebung, die beim Laden auf den Startpunkt aufgerechnet werden muss. Um diesen Abstand zu ermitteln, wird ebenfalls die Karte Gridmap V2 und

der Welteneditor genutzt. Dabei wird die Eigenschaft der Karte verwendet, dass die Fahrebene genau auf der Z-Koordinate 100 liegt. Daraufhin wird das Fahrzeug im Welteneditor ausgewählt und die Z-Koordinate des Chassis ausgelesen. Da Fahrzeuge voll ausgefedert erscheinen, wird nun die Schwerkraft der auf die des Plutos verringert. Die Differenz zur 100 ist die ermittelte Höhenverschiebung.

Abbildung 19 Ermittlung der Höhenanpassung für Fahrzeuge. (Rote Linien = Z Koordinate 0 der Karte | Blaue Linien = Z Koordinate 0 des Fahrzeuges)

Diese Daten liegen für drei Fahrzeuge vor: Erstens der Vivace, ein Sport-Kompaktwagen, der eine Mischung aus dem Peugeot 308 und dem Volkswagen Golf VIII R darstellt. Zweitens der ETK800, das Standardfahrzeug der BeamNG.tech Simulation, welches als Diesel-Kombi dem 2017er BMW 320d Touring ähnelt. Drittens der SUV Ibishu Hopper, der mit einem 2001er Jeep Wrangler vergleichbar ist. Die Auswahl an Fahrzeugen ist so ausgewählt, dass möglichst unterschiedliche Fahrzeugklassen vertreten sind.

Die unterstützen Karten und Startpunkte sind in der Datei „startingPoints.json“ gespeichert und werden ebenfalls vom „configLoader.py“ geladen. Es werden ebenfalls drei Karten unterstützt. Die Karte „italy“ stellt eine italienische Insel dar, auf der es mit Landstraßen, Autobahnen und Kleinstätten viele unterschiedliche Umgebungen zum Testen gibt. Weitere Merkmale der Karte sind Kreisverkehre, Tunnel und europäische Verkehrsschilder. Die zweite unterstütze Karte ist „east_coast_usa“, die die Straßenverhältnisse an der Ostküste der USA nachahmt. Sie unterscheidet sich durch alternative Fahrbahnmarkierungen, Geschwindigkeitsbegrenzungen in Meilen und dichteren Wälder, die für schattige Fahrbahnen sorgen. Abschließend sind Startkoordinaten für die „gridmap_v2“

gespeichert. Dies ist ein Testumgebung, die eine Ansammlung an unterschiedlichen Fahrbahnen, Untergründen und Steigungen bietet, aber keine realistische Welt nachstellt.
Die Inhaltsdateien der Karten und Fahrzeuge werden so ausgelesen, dass sie durch die Nutzenden erweiterbar sind. Die hinzugefügten Einträge stellen die Nutzenden dann über die textbasierte Oberfläche oder in der Konfigurationsdatei ein.

6.2 Schnittstellensoftware Nutzeroberfläche

Für das BeamTechEasyInterface wurden zwei Nutzeroberflächen entwickelt. In diesem Abschnitt werden diese in ihrer Nutzung und Funktion erklärt.

6.2.1 Grafische Benutzeroberfläche

Starten die Nutzenden das BeamEasyTechInterface mit eingestellter grafischer Benutzeroberfläche, öffnet sich zuerst das Hauptmenü, welches in Abbildung 20 dargestellt ist. Dort befinden sich links vier Schaltflächen, mit denen die Simulation gestartet, das Einstellungsmenü geöffnet oder das Programm beendet werden kann. Am unteren Rande befindet sich ein Hinweis, der auf die Registrierungsseite der BeamNG.tech Simulation verweist.

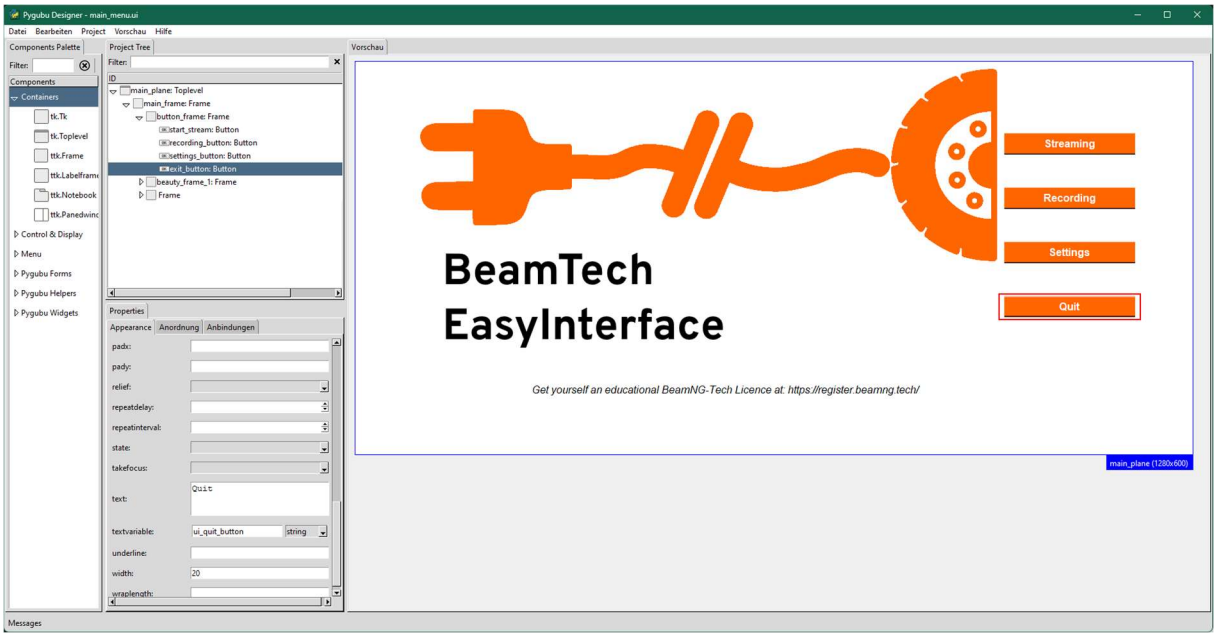

Abbildung 20 Umgesetzte Hauptübersicht der grafischen Benutzeroberfläche im Pygubu Designer

Sämtliche grafischen Oberflächen wurden mit dem Programm Pygubu Designer erstellt. Pygubu Designer ermöglicht es Tkinter- (Python Bibliothek für grafische Benutzeroberflächen) Oberflächen über eine eigene grafische Benutzeroberfläche zu erstellen. Diese

werden in einer Baumstruktur angelegt und Veränderungen unmittelbar nach der Eingabe in einer Vorschau angezeigt. Des Weiteren wird der benötigte Programmcode, um die erstellten Oberflächen in einem Python-Projekt zu nutzen, vom Pygubu Designer generiert. Dabei werden pro Oberflächen drei Dateien erzeugt. Eine „*.ui“ Datei, die den Aufbau und die Konfiguration der Oberflächenelemente in einem *XML*-Format abspeichert und zwei Python-Dateien. Die erste Python-Datei, die ihren Namen mit einem „_ui“ beendet, beinhaltet die grundliegende Logik, um die Oberfläche anzuzeigen. Dort werden zudem die Variablen für Texte und Eingabefelder zugänglich gemacht. Das BeamTechEasyInterface lädt hier auch die Texte vom „configLoader.py“, damit diese der eingestellten Sprache entsprechen. Ebenfalls werden in dieser Datei, die für das Einstellungsmenü „settings_gui_ui.py heißt, die aktuellen Konfigurationswerte in die Eingabefelder geladen. (Siehe Abbildung 21)

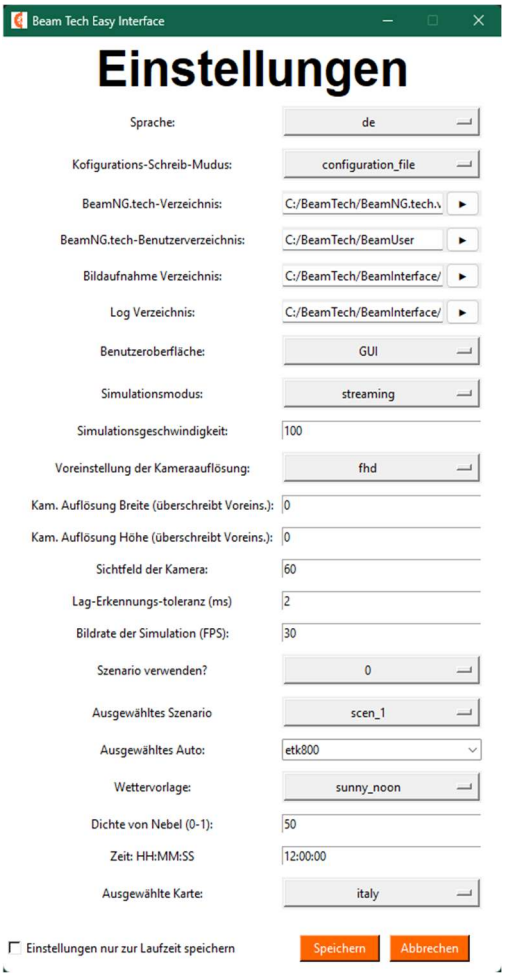

Abbildung 21 Umgesetztes Einstellungsmenü der Grafischen Benutzeroberfläche

In der zweiten generierten Python-Datei wird die Logik der Nutzeroberflächen hinterlegt. Dies umfasst in diesem Projekt hauptsächlich die Logik der Knöpfe. Für das Einstellungsmenü wird bei der Speichern-Schaltfläche zunächst eine temporäre Konfiguration aus den Werten der Eingabefelder generiert. Mit dieser wird, je nach Status der Laufzeit-Checkbox links, die Konfigurationsdatei oder die Laufzeitkonfiguration überschrieben. Daraufhin werden die neu vergebenen Pfade kontrolliert und bei einem Fehler die Nutzenden mit einem Pop-Up dazu aufgefordert, diese richtig zu setzen. Sollten die Pfade valide sein, so wird das Fenster, wie beim „Abbrechen“ Knopf, geschlossen.

Im Hauptmenü setzen die beiden oberen Knöpfe den jeweiligen Modus in der Laufzeitkonfiguration fest, bevor sie den „simulationConnector.py“ starten. Der „Beenden“ Knopf schließt das gesamte Programm und versucht, eine verbundene BeamNG.tech Instanz zu terminieren.

6.2.2 Textbasierte Oberfläche

Die textbasierte Oberfläche besteht aus zwei Teilen. Der erste Teil, der durch das „main.py“ Programm verwaltet wird, stellt Parameteroptionen für den Programmstart bereit. Diese Funktion wird mit Hilfe der „argpars“ Bibliothek umgesetzt, über die die möglichen Parameter, inklusive Hilfstext und Eingabeeinschränkungen, definiert werden. Dies ermöglicht es den Nutzenden das Programm mit dem Parameter „-h“ aufzurufen, um eine Auflistung mit allen Optionen und Erklärungen zu diesen ausgegeben zu bekommen. Die beim Start mitgegebenen Parameter werden nach dem Programmstart mit einem Übersetzungs-Dictionary, dem „configLoader.py“, verständlich beschrieben und die Werte entsprechend in die Konfigurationen geschrieben. Daraufhin werden die angegeben Pfade auf Validität überprüft und eine Eingabemaske angezeigt, wenn diese beschädigt sind. Wird die Option „`--start=y`“ von den Nutzenden mitgegeben, überspringt das Programm die Nutzeroberflächen und startet die Simulation.

```
PS C:\Users\maxih\OneDrive\Dokumente\Uni\MasterArbeit\Software\beamtecheasyinterface> python .\main.py -h
usage: main.py [-h] [--cwm {rt,cf}] [--ln {en,de}] [--bh BH] [--bu BU] [--rp RP] [--lp LP] [--ui {GUI,CLI}] [--mode {str,rec}]
               [--sisp SISP] [--rpst {vga,svga,hd,sxga,fhd}] [--srw SRW] [--srh SRH] [--fov FOV] [--st ST] [--fps FPS] [--sc SC]
               [--car CAR] [--weather WEATHER] [--fog FOG] [--time TIME] [--map MAP] [--start START]

Dies ist das BeamTechEasyInterface. Mit diesem Programm können Fahrdaten aus der BeamNG.tech Simulation gespeichert oder an andere
Programme übertragen werden. Weitere Infos unter: https://git.th-wildau.de/maha2541/beamtecheasyinterface

options:
  -h, --help            show this help message and exit
  --cwm {rt,cf}         configurationWriteMode | 'rt'/'cf' | Definieren Sie, wo Parameteroptionen geschrieben werden sollen: Laufzeit
                        (RunTime) oder Kofigurations-Datei (ConfigFile)
  --ln {en,de}          Sprache | 'de'/'en' | Definieren Sie die Anzeigensprache
  --bh BH               beamHome | Definieren Sie das beam home Verzeichnis
  --bu BU               beamUser | Definieren Sie das beam User Verzeichnis
  --rp RP               recordingPath | Definieren Sie das Verzeichnis, in dem Aufnahmen gespeichert werden
  --lp LP               loggingPath | Definieren Sie das Verzeichnis, in dem Logs gespeichert werden
  --ui {GUI,CLI}        userInterface | 'GUI'/'CLI' | Definieren Sie die Art der Benutzeroberfläche
  --mode {str,rec}      Modus | 'str'/'rec' | Definieren Sie den Programm Modus Streaming oder Recording
  --sisp SISP           simSpeed | Definieren Sie die Geschwindigkeit der Simulation (1-200)
  --rpst {vga,svga,hd,sxga,fhd}
                        resPreset | Nutzen einer VESA-Standardauflösung (VGA/SVGA/HD/SXGA/FHD)
  --srw SRW             simResWidth | Definieren Sie die Breite der Aufnahme. Ungleich 0 überschreibt diese die VESA-Standardauflösung
  --srh SRH             simResHight | Definieren Sie die Höhe der Aufnahme. Ungleich 0 überschreibt diese die VESA-Standardauflösung
  --fov FOV             fieldOfView | Definieren Sie das Sichtfeld der Kamera
  --st ST               slowdownTolerance | Definieren Sie die Toleranz für die Lag-Erkennung/Bildrate in ms
  --fps FPS             Frames pro Sekunde | Definieren Sie die Ziel FPS/Datenerhebungsrate für die Simulation
  --sc SC               scenario | Optional: Bestimmen Sie ein Szenario. Wenn verwendet wird werden Karten- und Umgebungsvariablen ggf.
                        überschrieben
  --car CAR             Auto | Definieren Sie das Auto
  --weather WEATHER     Wetter | Definieren Sie das Wetter
  --fog FOG             Nebel | Definieren Sie die Nebelintensität
  --time TIME           Zeit | Definieren Sie die Zeit
  --map MAP             Karte | Definieren Sie die verwendete Karte
  --start START         'y' Nutzen Sie diese Option, um die Simulation sofort und ohne Benutzeroberfläche zu starten
```

Abbildung 22 Ausgegebener Hilfstext der textbasierten Oberfläche

Alternativ dazu wird nun das Hauptmenü angezeigt und die Nutzenden zur Eingabe aufgefordert. (Siehe Abbildung 23) Optionen Drei bis Fünf geben die unterstützten Inhalte der jeweiligen Kategorie wieder. Manuelle Erweiterungen der Inhaltsdateien werden dabei nicht beachtet. Menüeintrag Zwei gibt die aktuell genutzte Konfiguration wieder und Eins startet die Simulation.

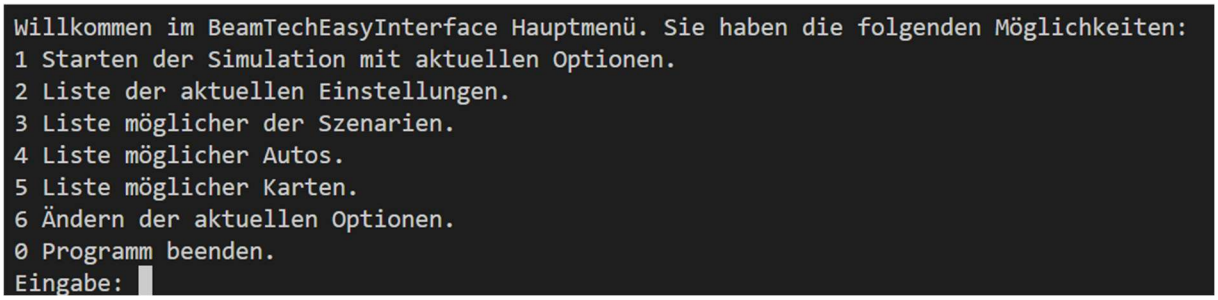

Abbildung 23 Hauptmenü der textbasierten Oberfläche

Wählen die Nutzenden die Option Sechs aus, wird ein Dialog zum Ändern der Konfiguration gestartet. Zuerst wird angezeigt, ob die aktuelle Konfiguration eine Laufzeitkonfiguration oder eine Konfigurationsdatei ist. Daraufhin fordert das Programm zur Angabe der zu verändernden Option auf. Anschließend wird der aktuelle Wert angezeigt und auf die Eingabe des neuen Werts gewartet. Nach dem Abschicken dieser Information erfolgt eine abschließende Abfrage. Dort ist durch die Nutzenden anzugeben, in welche Zielkonfiguration der neue Wert eingetragen werden soll. Ein beispielhafter Ablauf dieses Dialogs ist in Abbildung 24 dargestellt.

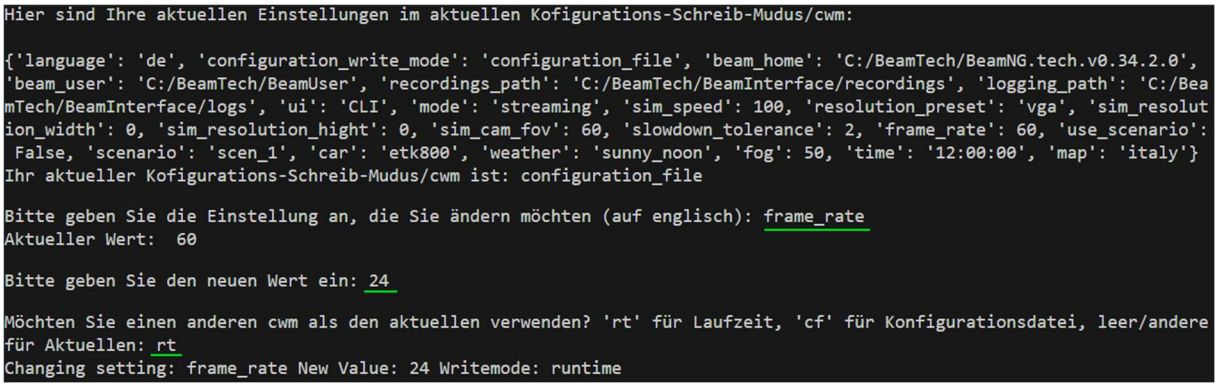

Abbildung 24 Beispielhafter Optionsdialog, beidem die Bildrate/Datenerhebungsrate der Laufzeitkonfiguration von 60 auf 24 gewechselt wird

Abschließend beendet der Menüpunkt Null das Programm und versucht verbundene BeamNG.tech Instanzen zu terminieren.

6.3 Umsetzung der Spurhalteassistenz

Der Spurhalteassistent wurde als zusätzliches Modul mit der Bezeichnung „LaneDepartureExample.py“ umgesetzt. Er nutzt den Streaming-Modus des BeamTechEasyInterface, um ein Fahrzeug zu kontrollieren und in der Mitte der Spur zu halten. In diesem Abschnitt wird zunächst die Umsetzung des Modulsoftware selbst erklärt und anschließend dessen Funktion bewertet. Bei der Softwareumsetzung wird die Bildverarbeitungskomponente getrennt von der Kommunikationsfunktion betrachtet. Der Programmcode ist im gleichen Git-Projekt wie das BeamTechEasyInterface als Beispielcode zu finden. https://git.th-wildau.de/maha2541/beamtecheasyinterface

6.3.1 Softwareumsetzung

Wird der Spurhalteassistent gestartet, verbindet er sich zunächst mit dem offenen Socket der laufenden BeamTechEasyInterface-Streaming-Instanz. Ist die Verbindung erfolgreich hergestellt, wird auf die Eingabe des Nutzenden gewartet. Diese Abfrage existiert, um den Nutzenden Zeit zu geben das simulierte Fahrzeug manuell von dem Standardstartpunkt an einen anderen Platz zu fahren.

Hauptschleife

Wird eine Eingabe registriert, startet eine unendliche Schleife, in der zuerst die Fahrdaten der Simulation mit einem „request_data“ angefragt werden. Die daraufhin empfangenen Daten werden von einer einzigen Textvariable in eine Liste umgewandelt, um die Werte einzeln anzusprechen. Eine Legende für die Reihenfolge der Werte in der Liste befindet sich in einem Kommentar am Anfang des Quellcodes. Anschließend werden empfangene Radgeschwindigkeit, Gaspedal- und Lenkradposition auf der Konsole ausgegeben.

Als nächstes wird die Radgeschwindigkeit mit dem Soll-Wert 10m/s (36km/h) verglichen. Ist die Geschwindigkeit zu niedrig, wird die Gaspedalposition um 1% erhöht. Ist sie zu hoch, wird das Gas komplett weggenommen. Folgend wird das Bild der Frontkamera mit der Nachricht „request_img“ angefragt. Da dies zu groß für ein einzelnes Datenpaket ist, wird eine Schleife gestartet, die den Inhalt mehrerer Pakete zusammensetzt. Die Abbruchbedingung für diese Schleife ist, dass ein Datenpaket die Bytes für die Zeichen „EOF“ beinhalten, da damit das Ende des Bilds markiert ist. Im Anschluss werden die empfangenen Daten von einer Ansammlung roher Bytes in ein *CV2*-kompatibles Bildobjekt umgewandelt. Dieses wird daraufhin der Bildverarbeitungsmethode

„detect_middle_of_lane(image)“ mitgegeben, welche die Mitte der Spur als Pixelposition zurückgibt. (Die Funktionsweise der „detect_middle_of_lane(image)“ Methode wird im nächsten Kapitel erklärt.)

Ist die errechnete Spurmitte ungleich 0, ist dies das Zeichen, dass keine Spur erkannt wurde. Anschließend der wert sie mit der Mitte des Bildes verglichen. Ist die Spurmitte links der Bildmitte, wird die Lenkradposition um einen Prozent nach links verändert. Andernfalls wird die Lenkposition nach rechts angepasst. Beim Vergleich werden der Spurmitte dabei 30 Pixel Toleranz eingeräumt, um ein Aufschwingen des Fahrzeuges zu verhindern.

```
239. image_mid = image_width/2
240. #if middle of road is to the left turn left and otherwise
241. if (mid_of_lane < (image_mid -15)):
242. if (steering> -1.0):
243.     steering -= 0.01
244.     print("Steer left")
245. elif (mid_of_lane > (image_mid + 15)):
246.     if (steering < 1.0):
247.         steering += 0.01
248.         print("Steer right")
249. else:
250.     steering = 0.0
251.     print ('normal')
```

Codeausschnitt zum Lenkvergleich aus LaneDepartureExample.py

Die dadurch ermittelte Lenkrad- und Gaspedalpositionen werden folgend in eine *JSON*-konforme Nachricht eingetragen und nach der Ankündigung „send_control“ als Steuerdaten übertragen. Die bestätigende Antwort des Streaming-Modus wird abschließend empfangen und ignoriert. Anschließend startet die Schleife von vorn.

Bildverarbeitung

Die hier angegeben Pixelwerte für z.B. das Zuschneiden des Bildes oder der Linienlänge in der Hough-Transformation sind auf ein Eingabebild mit der Auflösung 1280x720 Pixel optimiert.

Die Bildverarbeitung des Spurhalteassistenten findet in der Methode „detect_middle_of_lane()“ statt. Im ersten Schritt schneidet das Programm das Bild so zu, dass 125 Pixel vom oberen Rand und 200 Pixel vom unteren Rand entfernt werden. Anschließend wird das Bild in Graustufe umgewandelt und mit einem Gaußschen Weichzeichner verwischt. Auf dieses vereinfachte Bild wird nun der Canny[2]-Algorithmus angewendet und

in einer neuen Variable abgespeichert. Die eingestellten Schwellenwerte von 50-150 haben sich während mehreren Praxistests als Optimum zur Kantenerkennung herausgestellt. Daraufhin wird eine schwarze Maske mit identischer Größe wie das Canny[2]-Ergebnis erstellt. Anschließend wird ein Polygon definiert, das die untere Hälfte des Bildes abdeckt. Dieses Polygon wird nun so auf die Maske angewendet, dass die untere Hälfte weiß gefüllt wird. Folgend wird die „cv2.bitwise_and“ auf die schwarz-weiße Maske und das Canny[2]-Ergebnis angewandt.

Abbildung 25 Vergleich Quellbild (oben) und Ergebnis "cv2.bitwise_and" von Poligon und Canny Ergebnis (unten)

Auf das untere Bild in der Abbildung 25 wird nun die Hough[3]-Linien-Transformation angewendet, um markante Linien zu finden. Die verwendete minimale Linienlänge von 80 Pixeln und der maximale Abstand von 20 Pixeln zwischen Liniensegmenten wurde ebenfalls nach mehreren Tests als optimal für die Auflösung von 1280x720 Pixel befunden. Die gefundenen Linien werden zunächst in einer Liste abgespeichert und anschließend anhand der Winkel, horizontalen und nahe horizontale Linien, aussortiert. Die verbliebenen Linien werden daraufhin der Lage nach in linke und rechte Linien unterteilt. Aus diesen beiden Ansammlungen wird nun jeweils die durchschnittliche linke und rechte Linie berechnet. Abschließend wird der Mittelwert der X-Koordinaten der beiden Linien berechnet, um den Mittelpunkt der Spur zu erhalten.

6.3.2 Bewertung der Funktion

Um die Funktion des implementierten Spurhalteassistenten bewerten zu können, wurde ein Video des Einsatzes auf der Karte „italy“ aufgenommen. Dieses Video kann auf der Webseite YouTube unter Folgendem Link abgerufen werden:

(https://www.youtube.com/watch?v=qtkoTLbMS0Q)

Das BeamTechEasyInterface hat als kamerarelevante Einstellungen ein Sichtfeld von 60°, eine Bildrate von 60 Bilder die Sekunde und die Auflösungsvoreinstellung von VGA (640x480 Pixel) angewandt. Die Fahrzeuggeschwindigkeit ist auf 36km/h festgelegt.

In dem Video ist ab Sekunde 55 zu sehen, dass der Spurhalteassistent das Fahrzeug zuerst mit einem leichten Rechtsdrall in den Fahrbahnbegrenzungen hält. (Siehe Abbildung 26)

Abbildung 26 Ausschnitt aus dem Demonstrationsvideo, Sekunde 57

Daraufhin tastet sich der Spurhalteassistent immer weiter in Richtung Mitte der Fahrbahn, bis er dann ab 1:02 komplett mittig in der Spur bleibt. (Siehe Abbildung 27)

Abbildung 27 Ausschnitt aus dem Demonstrationsvideo, 1 Minute 2 Sekunden

Jedoch fällt ab 1:17 im Video auf, dass der Übergang auf eine Straße mit zwei Spuren nicht richtig erkannt wird und das Fahrzeug auf der mittleren Spurmarkierung fährt. (Siehe Abbildung 28) Ab 1:52 verliert der Spurhalteassistent die Fahrbahn komplett und fährt das Auto in die linke Leitplanke.

Abbildung 28 Ausschnitt aus dem Demonstrationsvideo, 1 Minute 34 Sekunden

Da der Spurhalteassistent als Funktionsbeweis für das BeamTechEasyInterface konzipiert wurde, ist das erzielte Ergebnis ausreichend. Es zeigt auf, dass das BeamTechEasyInterface genutzt werden kann, um aktive Fahrassistenzsysteme zu entwickeln.

7 Fazit

In diesem abschließenden Kapitel werden die erreichten Ergebnisse zusammengefasst. Außerdem wird ein gewerteter Ausblick auf mögliche zukünftige Erweiterungen gegeben.

7.1 Zusammenfassung Ergebnisse

In dieser Arbeit wurde anfangs die Akquirierung von Testmaterialien des Moduls - Bildverarbeitung im Automobil im Sommersemester 2022 - beschrieben. Zusätzlich wurde eine zu dieser Zeit entwickelte Spurerkennung, die diese Materialien nutzt, in ihrer Funktion vorgestellt. Daraufhin wurde ein ebenfalls 2022 implementierter Prototyp der Schnittstellensoftware analysiert und dessen Nachteile aufgezeigt.

Folgend wurden die Anforderungen an eine neu programmierte Schnittstellensoftware aufgestellt und beschrieben. Ebenfalls wurden Anforderungen an einen Spurhalteassistenten gestellt, der die Funktion der Schnittstellensoftware beweist. Aus diesen Anforderungskatalogen wurden anschließend zwei Konzepte erarbeitet, die die Programme in ihrer Funktion und ihrem Aufbau beschreiben. Des Weiteren wurde ein Beispiel für die zukünftige Nutzung der Schnittstellensoftware dargelegt.

Diese Konzepte wurden abschließend genutzt, um zwei Programme zu implementieren. Die Schnittstellensoftware BeamTechEasyInterface, die die BeamNG.tech Fahrsimulation nutzt, um Fahr- und Bilddaten zu erheben, speichert diese oder macht sie anderen Programmen zugänglich. Zudem ein Spurhalteassistent, der das BeamTechEasyInterface nutzt, um ein simuliertes Fahrzeug zwischen zwei Spuren zu halten.

Anforderungsrelevanz	**Anzahl**	**Anzahl vollständig umgesetzt**	**Anzahl teilweise umgesetzt**	**Prozent umgesetzt**
Alle (Schnittstellenpro.)	151	136	4	92,7%
Muss	117	110	1	94,9%
Soll	20	16	1	85%
Kann	16	10	2	75%
Spurhalteassistenz	10	9	0	90%

Tabelle 7 Übersicht der Umgesetzten Anforderungen

In Tabelle 7 sind die zuvor in Kapitel 4 aufgestellten Anforderungen an das Schnittstellenprogramm und die Spurhalteassistenz aufgezählt und nach ihrer Erfüllung aufgeschlüsselt. Die Umsetzung des BeamTechEasyInterfaces erfüllt von den anfangs aufgezählten Anforderungen 92,7%. Dabei wurden von der höchsten Relevanzstufe „Muss“ 94,9% der Anforderungen umgesetzt. Die Implementierung des Spurhalteassistenten konnte neun von zehn Anforderungen erfüllen. Somit konnte er zeigen, dass das BeamTechEasyInterface zur Entwicklung von aktiven Fahrassistenzsystem genutzt werden kann.

7.2 Mögliche Erweiterungen und Verbesserungen

Die für den Einsatz relevanteste Erweiterung, welche das Projekt des BeamTechEasyInterfaces benötigt, ist ein Ausbau des Benutzerhandbuchs und der Installationsanleitung. Da die Software im Rahmen der Lehre von Studierenden genutzt werden soll, ist eine ausführliche, jedoch leicht verständliche Dokumentation unabdinglich.

Eine zusätzliche, potenziell nützliche Erweiterung, wäre die Unterstützung von Szenarien. Ob hierfür die in BeamNG.tech vorhandene Szenario-Funktion, welche zwar sehr umfangreich, aber auch komplex ist oder ein eigene Lösung implementiert wird, bedarf weiterer Untersuchung.

Auch die Stabilität des Streaming-Modus kann in der Zukunft noch optimiert werden. Mit dem implementierten Spurhalteassistenten bricht die Verbindung, trotz eingebauter Gegenmaßnahmen, sporadisch ab. Der Recording-Modus hingegen hat sich als sehr stabil erwiesen und zeichnet auch lange Fahrten problemlos auf.

Da die BeamNG.tech Software selbst noch im Alpha[5]-Stadium ist, können manche Fehler nur im Dialog mit deren Entwicklern behoben werden. Darunter zählen die Falschanzeige transparenter Texturen, wie in der Abbildung 28 zu sehen ist oder auch das verschobene Darstellen von Schatten, die in Abbildung 14 sichtbar sind.

7.3 Schlusswort

Abschließend kann das in dieser Arbeit erreichte Ergebnis als Erfolg verzeichnet werden. Entsprechend den aufgestellten Anforderungen wurde Software konzipiert und entwickelt, die in einem realen Umfeld eingesetzt werden kann und dabei einen wirklichen Mehrwert bietet. Studierenden soll es hiermit einfacher gemacht werden, Bildverarbeitung realitätsnah zu lernen und bis hin zur Steuerung von Fahrzeugen in einem virtuellen Umfeld experimentieren zu können. Ich hoffe, mit dieser Arbeit einen Mehrwert für die Lehre der Bildverarbeitung im automobilen Umfeld zu liefern und durch Erweiterungen an dieser Software einen Beitrag zu leisten.

Literaturverzeichnis

BeamNG GmbH. (kein Datum). *BeamNG.Tech Technical Paper.* Abgerufen am 10. Februar 2025 von BeamNG.Tech: https://beamng.tech/

Grubits, C. (13. Januar 2021). *Masterarbeit: Datenreduktion mit dem AV1-Videocodec im Vergleich zu H.265.* Abgerufen am 19. Februar 2025 von Christoph Neuwirth: https://christoph-neuwirth.at/projekt/masterarbeit-datenreduktion-mit-dem-av1-videocodec-im-vergleich-zu-h-265/

Jennings, N. (07. Dezember 2024). *Socket Programming in Python (Guide)*. Abgerufen am 22. Februar 2022 von Real Python: https://realpython.com/python-sockets/

Simon, B. (kein Datum). *50 Jahre Bosch ABS-Geschichte | Bosch Global*. Abgerufen am 01. Februar 2025 von Bosch Global: https://www.bosch.com/de/stories/anfaenge-des-abs/

Stefan, B., & Andrea, R.-K. (2015). Fahrzeugdynamik, Mechanik des bewegten Fahrzeugs. In B. Stefan, & R.-K. Andrea, *Fahrzeugdynamik* (S. 181). Springer Vieweg.

Anhang

In diesem zusätzlichen Abschnitt werden Abbildungen und Tabellen, die im Hauptteil der Arbeit zu viel Platz einnehmen oder den Lesefluss stören würden.

Spezifikationen Testsysteme

Um Referenzpunkte für Leistungstests zu geben, werden hier die Spezifikationen der Testsysteme aufgeführt. Wenn Leistungsergebnisse in der Arbeit genannt werden, wird das genutzte Testsystem dazu genannt. Da sich unter den Testsystemen ein Laptop befindet und die Geschwindigkeit mobiler Komponenten durch die Konfiguration der Leistungsaufnahme stark variiert, wird die Leistungsaufnahme des Prozessors und der Grafikkarte zusätzlich in Klammern genannt.

Testsystem: [Laptop] Lenovo IdeaPad Pro 16 ACH 2021

Prozessor/ CPU	AMD Ryzen 7 5800H (45 Watt)
Arbeitsspeicher/ RAM	16GB DDR4 3200MT/s Dualchannel (gelötet)
Grafikkarte/ GPU	Nvidia GTX 1650 (Max-Q) (50 Watt)
Speichermedium	Samsung SSD 970 Evo Plus 2TB

Tabelle 8 Spezifikationen des Testsystems [Laptop]

Testsystem: [Desktop PC] Eigenbau basierend auf der Gigabyte Z690 GamingX DDR4 Hauptplatine

Prozessor/ CPU	Intel Core i7 12700k
Arbeitsspeicher/ RAM	32GB DDR4 3600MT/s Dualchannel
Grafikkarte/ GPU	Nvidia RTX 4070Ti Super (270 Watt)
Speichermedium	Samsung SSD 980 Pro 1TB

Tabelle 9 Spezifikationen des Testsystems [Desktop PC]

Testsystem: [Fahrsimulator] Eigenbau basierend auf dem

Prozessor/ CPU	Intel Core i5 12400F
Arbeitsspeicher/ RAM	16 GB DDR4
Grafikkarte/ GPU	Nvidia RTX 4060
Speichermedium	Sata SSD

Tabelle 10 Spezifikationen des Testsystems [Fahrsimulator]