Dokumentation Integrationsprojekt

im Studiengang Information Engineering

von

Studiengruppe HFP424

Prüfer: Prof. Dr. Joel Greenyer

Hannover, 2. September 2025

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende selbständig verfasst und keine anderen als die angegebenen Quellen verwendet habe. Alle Textstellen und andere Inhalte wie Bilder, Quellcode oder Daten, die wörtlich oder sinngemäß aus anderen Quellen entnommen sind, habe ich eindeutig mit korrekten Quellenangaben versehen. Über Zitierrichtlinien bin ich schriftlich informiert worden. Diese Arbeit wurde bisher in gleicher oder ähnlicher Form, auch nicht in Teilen, keiner anderen Prüfungsbehörde vorgelegt.

Falls ich während der Erstellung dieser Arbeit generative KI oder andere KI-gestützte Technologien verwendet habe, die über grundlegende Werkzeuge für Übersetzungen und zur Überprüfung von Grammatik oder Rechtschreibung hinausgehen, erkläre ich, nach der Nutzung dieser Tools den Inhalt meiner überprüft und bearbeitet zu haben und übernehme die volle Verantwortung für die diesbezüglichen Ausführungen. Eine Verwendung solcher Werkzeuge habe ich in meiner Arbeit dokumentiert (bspw. im Abschnitt „Struktur der Arbeit, Methodik“ oder durch Erläuterungen zur Nutzung im Anhang).

Hannover, den 2. September 2025

Studiengruppe HFP424

Inhaltsverzeichnis

Kapitel 1

Motivation

Autonomes Fahren ist in der heutigen Zeit keine ferne Vision mehr, sondern bereits Realität. Unternehmen wie Waymo in San Francisco, Baidu in Wuhan oder Volkswagen in Hamburg zeigen, dass fahrerlose Fahrzeuge am Straßenverkehr teilnehmen können. Hierdurch wachsen jedoch auch die Herausforderungen, die Sicherheit solcher Systeme zu gewährleisten und dementsprechend zu validieren. Die Vielzahl potenzieller Verkehrssituationen ist praktisch unbegrenzt, von einfachen Überholmanövern bis hin zu komplexen Interaktionen mit Fußgängern, Radfahrern, Ampeln und mehreren Fahrzeugen gleichzeitig. Hinzu kommen länderspezifische Verkehrsregeln und Regularien, die eine vollständige Erprobung im Realverkehr nahezu unmöglich machen.

Simulationen stellen daher einen unverzichtbaren Baustein in der Validierung autonomer Fahrfunktionen dar, da reale Testfälle nicht in der notwendigen Breite und Tiefe skalierbar sind. Mit Standards wie ASAM OpenScenario existieren bereits Ansätze, Szenarien in strukturierter Form zu beschreiben. Allerdings sind die XML-basierten Szenarien sehr aufwändig zu modellieren und müssen für jedes Testziel individuell erstellt werden. Neuere domänenspezifische Sprachen, wie die auf Constraints basierende OpenScenario DSL, versprechen zwar leichtere Handhabung, sind aber noch nicht weit verbreitet und erfordern ebenfalls manuelle Anpassungen. Industrielle Lösungen wie Fortellix Foretify nutzen Constraint-Solver und Planungsalgorithmen, erweisen sich in der Praxis jedoch häufig als starr und schwer automatisierbar.

Vor diesem Hintergrund ist die Entwicklung neuer, flexiblerer Ansätze zu automatisierten Szenariengenerierung von zentraler Bedeutung. Behavioral Programming bietet hierfür eine attraktive Grundlage. Es erlaubt eine modulare Modellierung von Szenarien und bildet konkurrierende Verhaltensweisen auf natürliche Weise ab. Ergänzend eröffent Reinforcement Learning die Möglichkeit, aus abstrakten Szenariobeschreibungen, etwa ein Auto überholt das VUT und bremst anschließend, konkrete, ausführbare

Szenarien zu generieren, die den abstrakten Vorgaben genügen.

Dieses Projekt verfolgt das Ziel, die Kombination von Behavioral Programming und Reinforcement Learning als neuartigen Ansatz zur automatisierten Szenariengenerierung zu untersuchen. Dabei sollen nicht nur Methoden zur Modellierung abstrakter und konkreter Szenarien entwickelt werden, sondern auch Verfahren, wie die ausgeführten Szenarien geloggt und visuell aufbereitet werden können, um Test-Engineers eine transparente Analyse zu ermöglichen. Vor diesem Hintergrund ergibt sich die zentrale Forschungsfrage:

> **Wie kann BPpy zur Modellierung ausführbarer Testszenarien im autonomen Fahren eingesetzt werden ?**

Zur Beantwortung dieser Frage weden folgende Unterfragen untersucht:

- Wie können Szenarien in BPpy modelliert werden und welche grundsätzlichen Unterschiede bestehen im Vergleich zu OpenSCENARIO?
- Welche Möglichkeiten der Komposition bietet BPpy für Szenarien, etwa sequenziell oder parallel?
- Wie kann BPpy in Kombination mit RL-Agenten eingesetzt werden, sodass auf Basis abstrakter Szenarien konkrete Testszenarien generiert werden?
- Wie können ausgeführte Szenarien geloggt und visualisiert werden, um eine transparente Analyse zu ermöglichen?

Das Vorhaben versteht sich als Machbarkeitsstudie. Es geht nicht darum, ein industrietaugliches Framework zu entwicklen, sondern vielmehr um die grundlegende Frage, ob sich durch die Kombination von Behavioral Programming und Reinforcement Learning ein flexibler, automatisierbarer Ansatz zur Generierung konkreter Testszenarien aus abstrakten Vorgaben realisieren lässt.

Kapitel 2

Verwandte Arbeiten

Kapitel 3

Grundlagen

3.1 Testumgebungen im autonomen Fahren

- kurze Abgrenzung VIL, HIL, SIL
- zur Einordung, welche unterschiedlichen Testebenen es gibt

3.2 Klassifikation von Testszenarien im autonomen Fahren

- Abgrenzung zwischen konkreten und abstrakten Szenarios
- am Beste mittels eines Beispiels z.B. Überhol-Manöver beschreiben

3.3 OpenSCENARIO als Standard zur Beschreibung von Testszenarien

- OpenScenario 1.x Standard (XML-basiert)
- OpenScenario DSL
- jeweils Beispiel zeigen, wie so etwas aussehen könnte
- Herausforderungen herausstellen

3.4 Eingesetzte Simulations-Engines im Rahmen dieser Arbeit

Simulationsumgebungen sind ein zentrales Werkzeug im Bereich des autonomen Fahrens. Sie ermöglichen es, Verkehrssituationen unter kontrollierten Bedingungen nachzustellen und die entwickelten Verfahren zu testen.

Abhängig vom jeweiligen Einsatzgebiet unterscheiden sich die Umgebungen vor allem im Grad der Abstraktion, in den bereitgestellten Funktionen und im benötigten Rechenaufwand. In den folgenden Abschnitten werden mit HighwayEnv sowie SUMO und dem dazugehörigen TraCI drei Umgebungen vorgestellt, die in unserem Projekt verwendet werden.

3.4.1 HighwayEnv

HighwayEnv ist eine Open-Source-Simulationsumgebung für autonomes Fahren, welches auf der Basis von Gymnasium beziehungsweise OpenAI Gym entwickelt wurde. Der Fokus liegt auf einer schnellen abstrahierten Simulation auf mehrspurigen Straßen und Autobahnen. Das Ziel dieser Simulationsumgebung ist die Entwicklung, Training und Evaluierung von Reinforcement-Learning-Algorithmen und Planungsverfahren im Bereich des autonomen Fahrens.

Technisch ist es mit Python implementiert und folgt einem modularen Aufbau, welche sich an der klassichen Agent-Environment-Schnittstelle aus dem Bereich des Reinforcement-Learnings bedient. Die einzelnen Szenarien werden durch Environment-Klassen definiert, etwas highway-v0 (ein klassisches Autobahn-Szenario) oder roundabout-v0 (ein Kreisverkehr). Damit eine effiziente Simulation stattfinden kann nutzt es mittels einem kinematischen Modells ein vereinfachte Fahrzeugdynamiken.

Bei der Fahrzeugmodellierung beschränkt sich HighwayEnv auf wesentliche Parameter, wie Position, Geschwindigkeit und Heading. Die Umgebungsmodellierung bietet mehrspurige Straßen, Fahrspurwechsel sowie unterschiedliche Verkehrsdichten. Ein Agent kann beschleunigen, bremsen, die Spur wechseln oder seine aktuelle Bewegung beibehalten. Die im Reinfocement-Learning benötigte Belohnungsfunktion kann je nach Zielstellung beliebig angepasst werden, z.B. auf Sicherheit oder Effizienz.

Der Fokus von HighwayEnv liegt vor allem auf abstrakten Szenarien und nicht hochrealistischen Simulationen. Es ist somit kein Ersatz für High-Fidelity-Simulatoren, wie CARLA. Zudem bietet es keine realistische Physik. Dadurch werden beispielsweise Fahrdynamiken oder Wetterbedingungen vernachlässigt. Der Vorteil von HighwayEnv liegt in den geringen Rechenkosten und ist somit geeignet für Experimente und Prototyping.

3.4.2 Sumo und Traci

Die Simulation of Urban MObility, kurz SUMO, ist ein Open-Source, mikroskopischer, straßenverkehrsbasierter Simulator, welcher vom Deutschen Zentrum für Luft- und Raumfahrt (DLR) entwickelt wird. Das Ziel ist es Verkehrsflüsse und - dynamiken in realistischen Straßennetzwerken zu simulieren. Es untsertützt Einzel- sowie Massensimulationen von Fahrzeugbewegungen in Städten, Autobahnen und Mischverkehr.

Die Implementierungssprache ist im Kern C++. Zudem stehen Python-APIs zur Verfügung mit denen Erweiterungen möglich sind. Mikroskopisch bedeutet in diesem Fall, dass jedes Fahrzeug einzeln simuliert wird. Die Straßennetze können aus realen Karten, wie OpenStreetMap importiert werden, sodass eine hohe Flexibilität geboten ist. Des Weiteren werden verschiedene Fahrzeugtypen von LKWs und Bussen über den PKW bis hin zum Fahrrad oder Fußgänger.

Für die Fahrdynamik werden Car-Following-Modelle wie das Krauß-Modell genutzt. Darüber hinaus finden Lane-Change-Modelle Anwendung. Innerhalb der Verkehrssteuerung können Amplen, Zuflussregelungen oder auch Stauszenarien simuliert werden. Des weiteren gibt es eine eingebaute Routenplanung sodass Fahrzeuge entweder feste Routen folgen oder auf Basis von dynamischen Routing-Algorithmen gesteuert werden können. Im Bereich der Auswertungen werden verschiedene detaillierte Statistiken im Bezug auf Reisezeiten, Emissionen oder Staus geliefert.

Die Einsatzbereiche von SUMO sind reichen von der Verkehrsforschung, wobei Verkehrsflüsse, Engpässe oder Infrastrukturmaßnahmen untersucht werden bis hin zur Stadtplanung und Mobilitätskonzepten bei denen Szenarien wie Car-Sharing oder ÖPNV simuliert werden. Zudem kann es auch im Bereich des Autonomen Fahrens als Testumgebung für Entscheidungs- und Koordinationsstrategien dienen.

Im Gegensatz zu HighwayEnv benötigt SUMO bei sehr großen Netzen eine hohe Rechenlast. Es bietet zudem auch keine High-Fidelity, da auch hier ein vereinfachtes Fahrzeugmodell verwendet wird.

Mit dem Traffic Control Interface (TraCI) bietet das DLR eine Client-Server-Schnittstelle zur Echtzeitsteuerung und Abfrage von SUMO-Simulationen. Dies erlaubt die dynamische Interaktion mit einer laufenden SUMO-Simulation. Dabei kann ein lesender Zugriff zur Abfrage von Zuständen etwa der Position, Geschwindigkeit und Ampelphasen geschehen, aber auch eine Manipulation der Simulation indem beispielsweise Fahrzeuge hinzugefügt oder Routen verändert werden. Dies erlaubt die Einbindung von externen Steuerungsalogrithmen z.B. Reinforcement-Learning oder Verkehrsmanagementsysteme.

3.5 Behavioral Programming als Programmierparadigma

- kurze Beschreibung des Programmierparadigma und welche Ideen dahinter stecken
- Besonderheiten herausstellen - Fokus auf Verhalten statt Kontrolle
- B-Threads

- Ereignisgesteuerte Synchronisation
- Modularität und Erweiterbarkeit

Kapitel 4

Problemanalyse

Kapitel 5

Umsetzung

5.1 Gesamt-Architektur

- aktuelles Bild noch entsprechend aufbereiten
- sprich BProgram zentral in der Mitte, daneben Simulation-Engines
- Logging wird in konkreten Szeanrien genutzt - Visualisierung nutzt Daten aus dem Logger
- RL nutzt auch BProgram? -> aktuell noch nicht, aber sollte es eigentlich

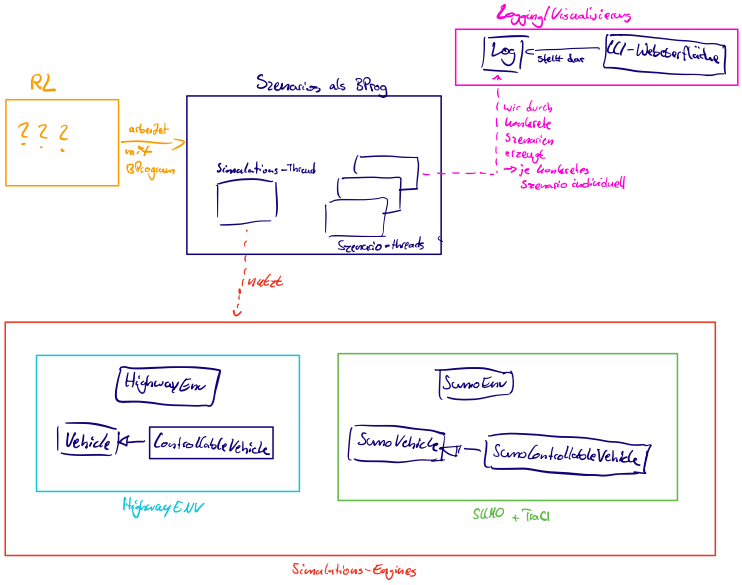

Abbildung 5.1: Gesamtarchitekturp

5.2 Simulationsumgebungen

5.2.1 HighwayEnv

Konfiguration von HighwayEnv

- z.B. Multi-Agent-setting
- und andere Besonderheiten, die wir in Q1 erarbeitet haben

Integration in das BProgram

- B-Thread für HighwayEnv, wie aufgebaut und warum?

VUT-Ansätze

- Training eines VUT, um es als Testfahrzeug für unsere Testszenarien zu nutzen

Ansätze zum Zugriff auf Fahrzeuginformationen

- Observation-Wrapper-Ansatz
- Vehicle-Based Ansatz
- Vergleich der beiden Ansätze

5.2.2 Sumo

- Warum SUMO anstelle von HighwayEnv?

SumoEnv zur Einbindung in das BProgram

- Klasse SumoEnv erklären
- implementiert Gym Env-Interface, wie auch HighwayEnv
- zur einfacheren Einbindung in unsere bestehende Arbeit

Abbildung der Fahrzeugeigenschaften

- da sich das Vehicle-Konstrukt als praktikabler herausgestellt hat wurde es analog auf Sumo übertragen
- somit muss in die bestehenden Szenarien nur die Fahrzeug-Klassen anders instanziiert werden, der Rest bleibt gleich

5.3 Modellierung von Szenarien mit BPpy

- Erklärung der einzelnen BThreads -> Simulation, abstraktes Szenario, konkretes Szenario
- am Beispiel von follow-behind-Szenario

5.4 Visualisierung von ausgeführten konkreten Szeanrien

5.5 Reinforcement Learning

Hier wird die Struktur und Funktionalität des Projekts, das ein Reinforcement-Learning-Modell (RL) für Überholszenarien trainiert, speichert und nutzt beschrieben. Die Hauptkomponenten des Projekts sind in mehreren Python-Dateien organisiert.

`src/envs/overtake_env.py` Diese Datei definiert die Umgebung für das Überholszenario, bei dem das VUT zum Überholen des vom Modell gesteuerten Fahrzeugs angeregt werden soll. Sie basiert auf einer Kernumgebung und erweitert diese um spezifische Logik für das Training eines RL-Agenten. Analog existiert die Datei `src/envs/intersection_env.py`, bei dem eine Umgebung für ein Kreuzungsszenario definiert wird.

- `__init__(render_mode, **config_overrides)`: Initialisiert die Umgebung mit dem übergebenen Rendermodus. Zusätzlich kann die Konfiguration der Umgebung über den `config_overrides` Parameter überschrieben werden.
- `reset(**kwargs)`: Setzt die Umgebung zurück, initialisiert die Position und Geschwindigkeit des Agenten sowie des zu überholenden Fahrzeugs (VUT).
- `step(action)`: Führt einen Schritt in der Umgebung aus, basierend auf der Aktion des Agenten. Berechnet Belohnungen anhand des Zustandes der Simulation, unter anderem der Position der Fahrzeuge im Vergleich zueinander. Abschließen wird überprüft, ob die Episode beendet ist.
- `render()`: Rendert die Umgebung.

`src/main.py` Diese Datei enthält den Einstiegspunkt für die Ausführung des Projekts und die Konfiguration der Umgebung.

- `create_env(config: Dict[str, Any])`: Erstellt eine neue Umgebung basierend auf der übergebenen Konfiguration.

- `set_config()`: Definiert die Konfiguration der Umgebung, einschließlich Fahrspuren, Fahrzeugpositionen und Beobachtungs-/Aktionsräume.
- `main()`: Führt die Hauptlogik aus, einschließlich der Initialisierung der Umgebung und der Simulation von Aktionen.

`src/training/train_overtake_agent.py` Diese Datei ist für das Training des RL-Modells verantwortlich.

- `make_env(rank: int)`: Erstellt eine Instanz der Überholumgebung für das Training.
- `RenderCallback`: Eine Callback-Klasse, der die Umgebung während des Trainings in regelmäßigen Abständen rendert. Ihre `_on_step()` Methode wird vom Modell während des Trainings aufgerufen um zu bestimmen ob die aktuelle Episode gerendert werden soll.
- `main()`: Führt das Training des RL-Modells durch. Es initialisiert das Modell und definiert die Traingsumgebung, trainiert es mit der `DQN`-Methode und speichert das trainierte Modell.

5.5.1 Trainieren des Modells

Das Training erfolgt in der Datei `train_overtake_agent.py`. Die Hauptschritte sind:

1. Initialisierung der Umgebung mit `DummyVecEnv`, welchem das zu lernende Szenario übergeben wird.
2. Definition des RL-Modells mit `DQN`. Dabei werden diverse Parameter wie der Diskontierungsfaktor für das Lernen des Modells festgelegt.
3. Start des Trainings mit `model.learn()`.

5.5.2 Speichern des Modells

Das trainierte Modell wird mit einem Zeitstempel versehen und im Verzeichnis `models/` gespeichert:

```
model.save("models/overtake_dqn.zip" + current_time)
```

5.5.3 Nutzen des Modells

Das gespeicherte Modell kann später geladen und für Inferenz oder weitere Trainingsschritte verwendet werden:

```
from stable_baselines3 import DQN
model = DQN.load("models/overtake_dqn.zip")
```

Kapitel 6

Evaluierung

Kapitel 7

Fazit

Kapitel 8

Ausblick

Anhang A

Anhang

Die Zitate habe ich nur hinzugefügt, weil ich sonst einen Error bekomme, dass ich keine citations nutze... [FHD18, FHD21]

Der Anhang stellt eine Ergänzung zur eigentlichen Arbeit dar. Denkbare Einsatzmöglichkeiten sind:

- Dokumentation von Materialien, die dem Leser üblicherweise nicht zugänglich sind (Herstellung der Zitierfähigkeit): Dieser Punkt kommt bspw. zum Tragen, wenn in der Arbeit unternehmensinterne Unterlagen eingearbeitet werden (z.B. Organigramme, Auszüge aus Organisationshandbüchern, Texte von Betriebsvereinbarungen, Verkaufsunterlagen, Produktbeschreibungen).
- Detaillierte Beschreibungen von hergestellten Artefakten, z.B. ausführliche Anwendungsfallbeschreibungen, wenn im Hauptteil Anwendungsfälle schon grob beschrieben sind und dies für das primäre Verständnis ausreicht, oder ausführliche Klassendiagramme, die übersichtsartige Diagramme im Hauptteil ergänzen. Hier können auch Code-Listings, von Algorithmen oder Skripten, die Sie entwickelt haben.
- Ergebnisse von Gesprächen mit Experten in Form von Gesprächsnotizen oder Interviewprotokolle (z. B. in Frage-Antwort-Form).

Ein Anhang kann eine sinnvolle Ergänzung der eigentlichen Arbeit sein. Material sollte jedoch nicht beliebig angehängt werden, sondern durch geeignete Beschreibungen mit dem Inhalt der Arbeit in Verbindung gebracht werden. Es gilt auch hier: Beschränken Sie sich auf Wichtiges. Im Regelfall sollte der Anhang nicht 50% des Seitenumfangs des Inhaltsteils überschreiten. Im Zweifel kann eine Arbeiten auch ohne Anhang auskommen.

Der Anhang ist insbesondere kein Ort für Abbildungen, die aus dem Hauptteil ausgelagert werden – insbesondere nicht zu dem Zweck, dadurch den vorgegebenen Seitenumfang des hauptinhaltlichen Teils einzuhalten. Es erschwert unnötig die Lesbarkeit der Arbeit, wenn zwischen

Hauptteil und Anhang hin- und hergeblättert werden muss. **Die Arbeit muss auch ohne Anhang vollständig und verständlich sein!**

Literaturverzeichnis

[FHD18] FHDW Hannover. Modulhandbuch Bachelor (mit Studienbeginn ab Oktober 2018). Interne Quelle der FHDW Hannover, 2018. Stand 17.2.2020.

[FHD21] FHDW Hannover. Modulhandbuch Master Information Engineering (Modulhandbuch HFP). Interne Quelle der FHDW Hannover, 2021. Stand Mai 2021.