Entwicklung einer Simulationsumgebung für die automatisierte Modellkonfiguration zur Auslegung und Absicherung KI-basierter Fahrfunktionen

Or Aviv Yarom*, Xiaobo Liu-Henke

Institut für Mechatronik, Ostfalia Hochschule für angewandte Wissenschaften, Salzdahlumer Str. 46/48, 38302 Wolfenbüttel, Deutschland; *o.yarom@ostfalia.de

Abstract. Die Weiterentwicklung des autonomen Fahrbetriebs erfordert den vermehrten Einsatz innovativer und intelligenter Algorithmen. Um diese effektiv und effizient entwickeln zu können, sind geeignete Entwicklungsmethoden und -werkzeuge erforderlich. Deshalb wird in diesem Beitrag die Entwicklung einer Simulationsumgebung für die automatisierte Modellkonfiguration zur Auslegung und Absicherung KI-basierter Fahrfunktionen vorgestellt. Auf Basis des aktuellen Standes der Technik werden die Konzeption inklusive Anforderungsdefinition und Realisierung der Simulationsumgebung detailliert beschrieben. Zusätzlich wird die Simulationsumgebung in einer Anwendung zur automatisierten Fahrzeugführung mit Künstlichen Neuronalen Netzen validiert.

Einleitung

Der vom Europäischen Fonds für regionale Entwicklung (EFRE) geförderte Innovationsverbund *autoMoVe* (*Dynamisch konfigurierbare Fahrzeugkonzepte für den nutzungsspezifischen autonomen Fahrbetrieb*), hat die Entwicklung eines autonomen, modularen und elektrischen Fahrzeugkonzeptes zum Ziel. Durch den Austausch anwendungsspezifischer Module zur Laufzeit soll eine Vielzahl von Anwendungen vom innerbetrieblichen Gütertransport bis zur Personenbeförderung im Straßenverkehr autonom realisiert werden. Im Rahmen dieses Forschungsvorhabens fokussiert das Teilprojekt der Ostfalia *autoEVM* (*Ganzheitliches elektronisches Fahrzeugmanagement für autonome Elektrofahrzeuge*) die modellbasierte Entwicklung innovativer intelligenter Algorithmen und Funktionen für den autonomen Fahrbetrieb.

Mit einer höheren Automatisierung des Fahrbetriebs geht auch ein Anstieg der Anforderungen an das Fahrzeug bzw. die automatisierten Fahrfunktionen einher. Aktuelle Funktionen und Algorithmen, die auf Methoden der Regelungstheorie oder der klassischen Informationsverarbeitung basieren, können diesen nicht mehr in vollem Umfang gerecht werden [1]. Deshalb stellt Künstliche Intelligenz (KI) in diesem Vorhaben bzw. für viele Domänen, die an der Entwicklung und Nutzung intelligenter, automatisierter Fahrzeuge beteiligt sind eine Schlüsseltechnologie dar [2].

Unabhängig von der Art der Informationsverarbeitung sind moderne Fahrzeuge und Fahrfunktionen komplexe mechatronische Systeme mit einem hohen internen und externen Vernetzungsgrad. Zur Beherrschung dieser Komplexität im Entwicklungs- und Absichrungsprozesss, wird auf eine in der Mechatronikforschung etablierte Entwurfsmethodik zurückgegriffen. Diese durchgängige und verifikationsorientierte Methodik basiert auf digitalen Modellen und Simulationen, um den Entwurfs- und Absicherungsprozess komplexer mechatronischer Systeme im vernetzten Umfeld einfacher, schneller und sicherer zu gestalten. [3]

Die zu diesem Zweck aktuell verfügbaren Entwicklungs- und Simulationsumgebungen für intelligente Fahrzeugfunktionen sind zwar grundsätzlich sehr umfangreich, konzentrieren sich aber weitestgehend auf konventionelle Algorithmen zur Informationsverarbeitung. Der Einsatz von KI-Funktionen bei Entwicklung und Absicherung ist dabei nicht oder nur mit hohem Aufwand möglich. Umgekehrt bieten aktuelle Entwicklungsumgebungen für KI-Algorithmen nicht die Vorzüge bzw. den Funktionsumfang von Tools, die speziell auf automatisierte Fahrfunktionen ausgerichtet sind. [4]

Deshalb wird in diesem Beitrag die Entwicklung einer Simulationsumgebung für die automatisierte Modellkonfiguration zur Auslegung und Absicherung KI-basierter Fahrfunktionen vorgestellt. Diese Simulationsumgebung ist einerseits mit ihrem Funktionsumfang auf die Entwicklung automatisierter Fahrfunktionen ausgerichtet und bietet andererseits die Möglichkeit nicht nur konventionelle sondern auch KI-Algorithmen einzusetzen.

1 Entwurfsmethodik

Die Komplexität moderner Fahrzeuge steigt durch den höheren internen und externen Vernetzungsgrad sowie die wachsende Anzahl intelligenter und leistungsfähiger Hard- und Softwarekomponenten stetig an. Zur Beherrschung der Systemkomplexität und frühzeitigen Fehlervermeidung bei der Auslegung der Informationsverarbeitung, ist eine ganzheitliche Entwurfsmethodik unverzichtbar. Daher hat sich die durchgängige, verifikationsorientierte, modellbasierte Entwurfsmethodik, basierend auf dem Rapid Control Prototyping (RCP) und Model-in-the-Loop-(MiL), Software-in-the-Loop-(SiL) und Hardware-in-the-Loop-(HiL) Simulationen etabliert. [3]

Die Methodik baut auf funktionsorientierten physikalischen Modellen einer Regelstrecke auf. Die Regelungsfunktion wird anschließend in Abhängigkeit des Systemverhaltens simulativ ausgelegt und frühzeitig in MiL-Simulationen validiert. Um manuelle Programmierung zu umgehen, werden Modell und Regelungsfunktion in blockschaltbildbasierten Programmiersprachen entwickelt. Der anschließend automatisch generierte Funktionscode wird in SiL-Simulationen erneut gegen das Streckenmodell getestet. HiL-Simulationen dienen zur weiteren Validierung und Optimierung der Informationsverarbeitung mit echtzeitfähigen Simulationsmodellen und realen Teilkomponenten des zu regelnden Systems.

Der verifikationsorientierte und iterative Ansatz dieser Methodik unterstützt den Entwicklungsprozess auch in der herausfordernden Aufgabe der Absicherung. Die Methodik adressiert die Schwächen der klassischen auf physischen Prototypen basierenden Validierung, wie einem hohem Ressourcenaufwand oder Sicherheitsrisiken für Mensch, Maschine und Umwelt. Durch ihren virtuellen Charakter sparen MiL-, SiL- und HiL-Simulationen Zeit und Kosten [5]. Sie ermöglichen jederzeit durchführbare und reproduzierbare Tests ohne direkte Abhängigkeit von physischen Prototypen, Tageszeiten oder menschlichen Experten. So lassen sich Simulationsdurchläufe, für verschiedene Funktionsvarianten oder Szenarien automatisiert durchführen. Damit eignet sich diese Methodik auch besonders für das Training von KI-Algorithmen. Denn das sogenannte maschinelle Lernen verläuft, bis auf seltene Ausnahmen, immer iterativ.

Virtuelle Entwurfsmethoden wie diese bilden die Grundlage für viele intelligente Systeme, wie hochautomatisierte Fahrzeuge. Mit prototypenbasierten Tests wären die benötigten Hunderttausenden Testkilometer nicht mit vertretbarem Zeit- und Kostenaufwand zu leisten. [5]

2 Stand der Technik

2.1 Intelligente Fahrfunktionen

Beim automatisierten Fahren werden dem menschlichen Fahrer einzelne Fahraufgaben von sogenannten Fahrerassistenzsystemen (FAS) abgenommen. Solche FAS, z.B. zur Geschwindigkeitsregelung oder Spurhalteassistenz sind schon seit längerem in Serie verfügbar [6]. FAS verarbeiten von Fahrzeug- und Umgebungssensoren erfasste Daten und berechnen dadurch Fahrbefehle, welche anschließend mittels geregelter Aktorik umgesetzt werden. Mit steigender Anzahl und Vernetzung der FAS delegiert der menschliche Fahrer sukzessive Fahraufgaben an das Fahrzeug bis er beim autonomen Fahrbetrieb schließlich zum Passagier wird. Dann spricht man nicht mehr von FAS, sondern von (automatisierten) Fahrfunktionen.

Eine ansteigende Automatisierung des Fahrbetriebs bedeutet eine extreme Steigerung der Anforderungen an die Fahrfunktionen. So werden nicht nur mehr, sondern auch verschiedene Sensoren zur Umfeldperzeption benötigt, deren inhomogene Rohdaten mehrfach, intensiv und mit höchsten Echtzeitanforderungen verarbeitet und umgesetzt werden müssen. Kommen Informationen aus der internen Bus- oder externen Vehicle-to-Everything-(V2X-) Kommunikation hinzu, steigt die Komplexität weiter an. Dadurch geraten konventionelle Algorithmen zur Regelung und Datenverarbeitung an ihre Grenzen. [1] Deshalb werden bereits heute KI-Algorithmen für automatisierte Fahrfunktionen eingesetzt. Besonders vielversprechend sind dabei Künstliche Neuronale Netze (KNN) in Kombination mit dem Maschinellen Lernen (ML). Prominente Anwendungen sind die bildbasierte semantische Segmentierung der Fahrumgebung [7] oder die automatisierte Fahrzeugführung [8]. KI-Algorithmen sind aufgrund ihrer Leistungsfähigkeit, Robustheit und Anpassungsfähigkeit vielversprechend für die Weiterentwicklung autonomer Fahrzeuge [9]. Dabei bietet sich auch außerhalb der automatisierten Fahrzeugführung Potential für intelligente Fahrzeugfunktionen, z.B. für das-Batteriemanagement [10]. Auf niedrigen Ebenen der Informationsverarbeitung z.B. der lokalen Regelung der Aktorik werden hingegen häufig weiterhin konventionelle Ansätze verwendet [1].

2.2 Grundlagen Künstlicher Neuronaler Netze und Maschinellen Lernens

Der Begriff KI umfasst eine Vielzahl unterschiedlichster Methoden und Algorithmen, die sich mit der

selbstständigen und automatisierten Lösung von Problemen befassen [2]. KNN und ML bilden ein Teilgebiet der KI, das sich in zahlreichen Problemen verschiedenster Gebiete, auch beim autonomen Fahrbetrieb, als geeignet erwiesen hat. Deshalb fokussiert sich dieser Beitrag auf dieses Teilgebiet. Die zahlreichen positiven Eigenschaften von KNN und ML, wie Anpassungsfähigkeit, Fehlerresistenz, Vielseitigkeit und vor allem Lernfähigkeit sind auf ihre Anlehnung an die Struktur und Funktionsweise des menschlichen Gehirns zurückführen.

Analog zur Biologie sind (künstliche) Neuronen Verarbeitungseinheiten, die Eingangsreize über gewichtete Verbindungen kumulieren und mit Hilfe einer Aktivierungsfunktion eine Ausgabe berechnen. Durch die Zusammenschaltung mehrerer Neuronen in mindestens zwei Schichten entsteht das KNN. Üblich sind Kombinationen von bis zu einigen Hundert Neuronen in bis zu über Einhundert Schichten. Dabei sind nicht nur beliebige vorwärtsgerichtete, sondern auch zeitlich rückgekoppelte Verbindungen im KNN möglich. Die optimale Architektur eines KNN lässt sich bisher nicht analytisch bestimmen [11]. Deshalb sind Erfahrung und Versuchsreihen notwendig, um eine geeignete Architektur im Zielkonflikt zwischen Rechenaufwand und Leistungsfähigkeit zu finden. Die Anzahl, Verschaltung und Gewichtung der Verbindungen charakterisiert die „Intelligenz“ eines KNN. Allgemein gesagt, bedeuten mehr Neuronen und Verbindungen eine höhere Leistungsfähigkeit des KNN, bei gleichzeitig steigendem Rechenaufwand.

Genau wie ein menschliches Gehirn, muss das KNN eine Aufgabe zunächst lernen bzw. trainieren. Diese Begriffe bezeichnen die Anpassung der Verbindungsgewichte. Im Umfeld des autonomen Fahrens sind dafür das Supervised Learning (SL) und das Reinforcement Learning (RL) relevant. Beim SL werden dem KNN Eingangsdaten und die zugehörige Ausgabe präsentiert. Das KNN lernt iterativ den Zusammenhang zwischen den beiden Größen [12]. Dieses Lernverfahren eignet sich z.B. besonders für bildbasierte Objekterkennung [7]. Beim RL lernt das KNN sukzessive aus der Erfahrung vergangener Sequenzen die optimale Strategie im Sinne einer vorgegeben Belohnungsfunktion [12]. Dieses Verfahren wird eingesetzt, wenn keine Trainingsdaten verfügbar sind, z.B. bei der automatisierten Fahrzeugführung [8]. SL und RL sind Oberkategorien von Lernverfahren, mit diversen konkreten Trainingsalgorithmen. Genau wie die KNN Architektur lassen sich die optimalen Trainingsalgorithmen bzw. deren Parameter nicht analytisch bestimmen. So sind auch hier Erfahrung und Versuche erforderlich.

2.3 Entwicklungsumgebungen für intelligente Fahrfunktionen

Für den modellbasierten Entwurf automatisierter Fahrfunktionen existieren einige Entwicklungs- und Simulationsumgebungen, wie MATLAB/Simulink mit Toolboxen, dSPACE Automotive Simulation Models (ASM), und IPG CarMaker, um nur einige zu nennen. Sie alle unterscheiden sich hinsichtlich ihres primären Fokus oder ihrer spezifischen Vor- und Nachteile. Alle bieten umfangreiche Modellbibliotheken für Verkehrs-, Fahrdynamik- Komponenten-, Sensor- oder sogar Fahrermodelle. Sie wurden speziell für die Konfiguration und reproduzierbare Simulation verschiedenster Fahrszenarien und Fahrzeugvariationen zum Zwecke der Fahrzeugentwicklung oder -absicherung entwickelt. Die Tools besitzen meist integrierte Modell-Konfiguratoren, eine Visualisierung, Experimentierumgebungen, teilweise auch ein Szenario- und Testmanagement. Die aufgezählten Tools eigenen sich für die in Kapitel 1 beschriebene Entwurfsmethodik mittels RCP-Prozess. Sie ermöglichen nicht nur MiL-Simulationen, sondern durch automatische Codegenerierung in Verbindung mit MATLAB/Simulink auch echtzeitfähige SiL- und HiL-Simulationen. [13] Allerdings bieten die Tools keine Möglichkeit ohne weiteres KNN in die jeweiligen Simulations- und Konfigurationstools einzubinden, geschweige denn diese zu trainieren.

Die einzige Ausnahme stellt hier die Deep Learning Toolbox (DLT) von MATLAB dar. Mit dieser Toolbox können beliebige KNN Architekturen konfiguriert und mit einer Vielzahl vorimplementierter Algorithmen performant trainiert werden. Die DLT ist grundsätzlich kompatibel mit Simulink sowie der automatischen Codegenerierung. Die KNN Konfiguration ist aber recht kompliziert und erfolgt in der Regel manuell für einzelne KNN. Dadurch eignet sie sich in ihrer Ursprungsform wenig für die Durchführung von Versuchsreihen mit variierenden Architekturen, Trainingsparametern oder auch unterschiedlichen Fahrfunktionen.

Auch für das Training von KNN existieren spezielle Entwicklungsumgebungen. Die meisten von ihnen basieren auf der Programmiersprache Python. Prominente Tools sind unter anderem TensorFlow, PyTorch, Keras und Caffe. Sie alle besitzen komfortable und umfangreiche Funktionalitäten hinsichtlich der Erstellung und des Trainings von KNN. Allerdings gibt es hier nur die Möglichkeit zur skriptbasierten Programmierung. Eine übersichtlichere für den RCP-Prozess geeignete blockschaltbildbasierte Programmierung ist nicht möglich. [4] Daher

gestaltet es sich schwer mit den KNN-spezifischen Tools gleichwertige Simulationen hinsichtlich Genauigkeit, Rechenzeit und Komfort zu entwerfen, wie mit den Tools für automatisierte Fahrfunktionen. Der Aufwand hierfür ist so groß, dass sogar Computerspiele wie Grand Theft Auto [14] oder TORCS [15] für Fahrzeug- und Verkehrssimulationen in Kombination mit den genannten KNN-Tools gekoppelt wurden. Diese Ansätze dienten jedoch eher der Untersuchung der KNN und des ML und werden den Ansprüchen an Genauigkeit, Echtzeitfähigkeit, Reproduzierbarkeit und Variationsmöglichkeit einer realen Fahrfunktionsentwicklung nicht gerecht.

3 Konzeption der Simulationsumgebung

3.1 Ableitung der Problemstellung

Mit den Erkenntnissen aus Kapitel 2 lässt sich eine Problemstellung für die Verfolgung der Projektziele (Kapitel 1) ableiten. Der Entwurf intelligenter automatisierter Fahrfunktionen erfordert den Einsatz von KI-Algorithmen. Diese lassen sich aber, wie beschrieben nicht ohne weiteres in den üblichen Entwicklungsumgebungen einsetzen. KNN und ML Verfahren werden sehr schnell sehr komplex und unübersichtlich. Eine händische Programmierung verschiedener KNN-Architekturen mit den zugehörigen Berechnungsvorschriften wäre nicht nur fehleranfällig, sondern würde auch viel Zeit in Anspruch nehmen. Analog verhält es sich mit den ML-Verfahren, die parametriert und für jede KNN-Architektur angepasst werden müssten. Der Umstand, dass Architekturdesign, Trainings- und Testprozesse stets empirisch und iterativ ablaufen, erfordert viele Simulationsdurchläufe und verschlimmert die Situation weiter.

Der Einsatz von KNN-spezifischen Entwicklungsumgebungen ist ebenfalls ausgeschlossen. Will man die automatisierten Fahrfunktionen unter realistischen Bedingungen auslegen, muss die Modellierungstiefe ausreichend exakt sein. Beim klassischen Weg der analytischen physikalischen Modellbildung müssten die mathematischen Gleichungen und deren numerische Lösungsverfahren selbstständig in einer Skriptsprache programmiert werden. Betrachtet man nun die Anzahl der Subsysteme eines Fahrzeugs, weitere Fahrfunktionen oder andere Verkehrsteilnehmer, die mitsimuliert werden sollen, ergibt sich ein hochkomplexes Simulationssystem. Hinzu kommen verschiedene Varianten, Konfigurationen, Szenarien und gegebenenfalls Echtzeitanforderungen. Die Umsetzung eines solchen Vorhabens wäre zwar grundsätzlich möglich, aber sehr fehleranfällig und wenig zielführend. Schließlich gibt es bereits ressourcenoptimale Simulationsumgebungen für diesen Zweck.

Zusammenfassend eignet sich also keine der derzeitig verfügbaren Simulationsumgebungen für die Auslegung und Absicherung KI-basierter Fahrfunktionen nach der Entwicklungsmethodik aus Kapitel 1. Als Schlussfolgerung ergibt sich demnach, die Entwicklung einer eigenen Simulationsumgebung für diese Entwurfsmethodik.

3.2 Definition von Anforderungen

Zur Bewältigung der genannten Herausforderungen, werden im Folgenden die Anforderungen an die neue Simulationsumgebung für die automatisierte Modellkonfiguration zur Auslegung und Absicherung KI-basierter Fahrfunktionen definiert:

A1. Nutzbar für verschiedene Fahrfunktionen
A2. Betrieb mit Simulink und Kompatibilität mit entsprechenden Blöcken und Modellen
A3. Nutzung verschiedenster vorhandener oder Erstellung eigener Modelle und Funktionen mit beliebiger Modellierungstiefe
A4. Einfache Erstellung und Berechnung von KNN
A5. Automatische Integration von KNN in die Simulinkmodelle
A6. Variation der Modellkonfiguration
A7. Automatische Modellkonfiguration
A8. Automatisierter Durchlauf von Simulationsreihen
A9. Training mittels verschiedener ML Verfahren
A10. Einsatz zur Generierung von Trainingsdaten für ML
A11. Visualisierung in 2D und 3D
A12. Bedienung mit Nutzeroberfläche oder skriptbasiert
A13. Performante Rechenzeiten für große Simulationsreihen und spätere Echtzeitanwendungen
A14. Kompatibilität mit dSPACE ASM für die spätere Weiterentwicklung der Simulationsumgebung
A15. Automatische Codegenerierung möglich

3.3 Konzeptbildung

Der erste Schritt zur Erfüllung der zuvor definierten Anforderungen, ist die Realisierung der Simulationsumgebung in MATLAB und Simulink (A2). Dadurch ist die Kompatibilität zu Standardblöcken und bereits vorhandenen Modellen, die in Simulink verfügbar sind, gegeben (A3). Des Weiteren können beliebige weitere Modelle und automatisierte Fahrfunktionen erstellt werden (A1).

Hierbei kann grundsätzlich auch auf Toolboxen zurückgegriffen werden. Dabei ist lediglich darauf zu achten, dass diese auch die automatische Codegenerierung unterstützen (A15). In diesem Fall ist auch die Kompatibilität zu dSPACE ASM gewährleistet (A14).

Für die Konfiguration und Erstellung von KNN wird auf die DLT zurückgegriffen. Hierfür wird ein zusätzlicher KNN Generator entworfen, der beliebige KNN-Architekturen durch einfache Benutzereingaben über eine grafische Benutzeroberfläche (GUI) oder ein Skript mit einem automatisiert erstellt (A4) und direkt in das Simulationsmodell generiert (A5). Durch die Verwendung der DLT sind automatisch verschiedene ML Algorithmen verfügbar (A9). Für das SL sind bereits mehrere Algorithmen nutzbar, sofern man Trainingsdaten generiert hat (A10). RL Algorithmen sind in der RL Toolbox von Matlab verfügbar. Alternativ ist es sowohl für SL als auch für RL grundsätzlich auch möglich eigene Trainingsalgorithmen zu implementieren, da man mit der DTL recht komfortabel auf die Verbindungsgewichte zugreifen kann.

Mit einem Szenario-Generator können verschiedene Simulationsszenarien hinsichtlich der Fahrstrecke und der Verkehrsteilnehmer konfiguriert werden (A6). Diese Konfiguration wird anschließend automatisch an das Simulationsmodell übergeben. Um das Geschehen im Szenario während und nach der Simulation besser interpretieren zu können, wird zusätzlich eine rudimentäre Visualisierung implementiert. Die 2D- bzw. 3D-Visualisierung basiert auf dem Bird's-Eye Scope und der Plot-Funktion von MATLAB (A11).

Die Konfiguration und Erstellung des Szenarios erfolgt ebenfalls durch eine GUI oder ein Skript (A12). Durch die Möglichkeit der skriptbasierten Szenario- und KNN-Generierung, lassen sich Simulationsabläufe automatisieren (A8). Dabei werden die entsprechenden Modellparameter automatisch konfiguriert und aktualisiert (A7). Genau wie bei den entwickelten Fahrfunktionen und Modellen, muss auch bei der Realisierung von Szenario- und KNN-Generator sowie den Automatisierungsmechanismen auf eine schlanke und rechenzeitoptimierte Programmierung geachtet werden (A13).

4 Realisierung der Simulationsumgebung

Abbildung 1 zeigt Aufbau und Struktur der Simulationsumgebung. Sie besteht im Wesentlichen aus dem Simulationsmodell selbst und aus der Steuerung der Simulationsumgebung. Im Simulationsmodell finden die Modellierung und die Berechnung der Simulation statt. Darin enthalten sind sowohl die Modelle, Parameter und Funktionen des gesamten Ego-Fahrzeugs in der gewünschten Modellierungstiefe als auch die im Szenario definierte Umgebungssimulation. Die zu entwickelnde automatisierte Fahrfunktion, also beispielsweise das KNN ist im Punkt Fahrfunktion zusammengefasst. Sie interagiert mit den anderen simulierten Bestandteilen, wie den Sensor-, Fahrdynamik-, Komponenten- oder Kommunikationsmodellen. Gleichermaßen kommunizieren die Modellbestandteile über definierte Schnittstellen mit den Komponenten des Simulationsszenarios.

Bevor eine Simulation stattfinden kann, muss zunächst eine Konfiguration der Simulationen stattfinden. So teilt sich die Steuerung der Simulationsumgebung in zum einen in die Konfiguration der Simulationsdurchläufe, zum anderen in die Simulationssteuerung und

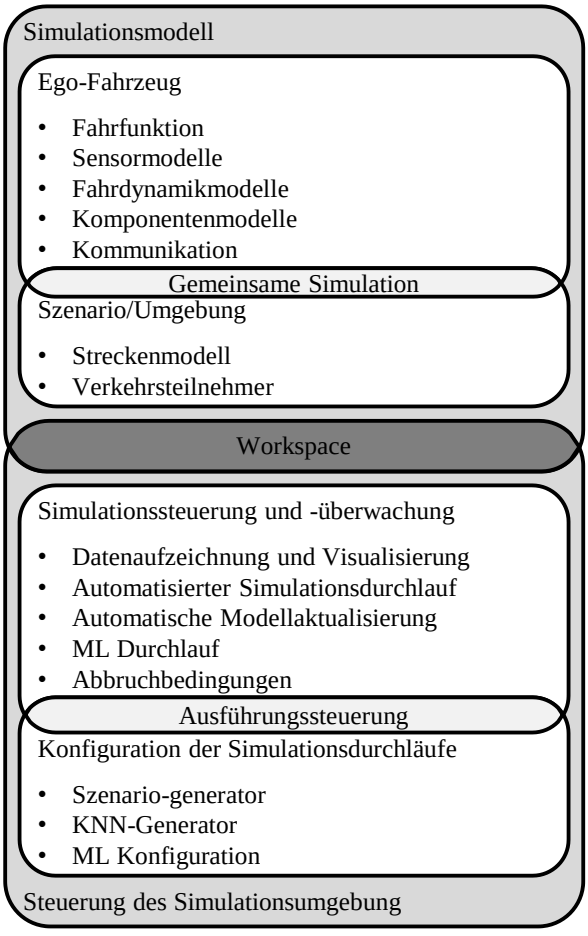

Abbildung 1: Aufbau und Struktur der Simulationsumgebung für die automatisierte Modellkonfiguration zur Auslegung und Absicherung KI-basierter Fahrfunktionen

Überwachung. Die Konfiguration der Simulationsdurchläufe ist im Grunde die Eingabeschnittstelle des Nutzers. Mit dem Szenario-Generator kann entsprechend der Funktionsspezifikation ausgewählt werden wie genau jede einzelne Simulation aussehen und ablaufen soll. Einstellbar sind z.B. der Streckenverlauf, die Art, Anzahl und Bewegung anderer Verkehrsteilnehmer usw. Der KNN-Generator ermöglicht die Auswahl der KNN-Architekturen für die verschiedenen Durchläufe. In der ML Konfiguration werden Lernalgorithmen ausgewählt und parametriert. Die Informationen aus der Nutzereingabe dienen zur Ausführungssteuerung für die Simulationssteuerung und -überwachung.

Es wird also ein Ablaufplan zur Automatisierung der Simulationsdurchläufe erzeugt. Darin werden z. B. mehrere ML Durchläufe zunächst in einem und dann in einem weiteren Szenario gestartet. Vor jedem Durchlauf werden die entsprechenden Parameter der jeweils aktiven Konfiguration geladen und über den MATLAB Workspace an das Simulationsmodell übergeben. So wird das Modell automatisch aktualisiert. Der Workspace dient als Schnittstelle zwischen Steuerung der Simulationsumgebung und dem Simulationsmodell. Er ermöglicht die Aufzeichnung und Speicherung der Simulationsdaten für die Visualisierung und spätere Auswertung. Des Weiteren überträgt er ausgewählte Größen, zur Modellüberwachung. Bei vorher definierten, unzulässigen Zuständen, wird die die Simulation automatisch abgebrochen.

5 Simulation und Auswertung

5.1 Beschreibung des Anwendungsfalls und der Modellbildung

Zur Veranschaulichung soll in diesem Anwendungsfall eine Fahrfunktion zur automatisierten Querführung bei konstanten Geschwindigkeiten auf beliebigen, einspurigen Strecken ohne andere Verkehrsteilnehmer ausgelegt werden. Dabei ist das Beispiel bewusst komprimiert gewählt, um zu zeigen welcher Simulations- und Zeitaufwand entsteht. Dadurch soll der Mehrwert der automatisierten Modellkonfiguration verdeutlicht werden.

Da in diesem Anwendungsfall die Fahrzeuggeschwindigkeit konstant ist, wird ein lineares Einspurmodell zur Abbildung der Fahrdynamik eingesetzt. Automatisch generierte Strecken nach Richtlinien der Bundesanstalt für Straßenwesen mit einer Gesamtbreite von 3,5 m bilden die Umgebung. Das Sensormodell besteht aus elf Linien, die in einem Winkelbereich von ±40 ° und einem Radius von 8 m die Entfernung zu den Fahrbahnbegrenzungen erkennen. Abbildung 2 a) und b) veranschaulichen den Anwendungsfall mit der Visualisierungsfunktion der Simulationsumgebung. Andere Fahrzeugkomponenten und Kommunikationssysteme werden nicht berücksichtigt. Die automatisierte Querführung wird von einem KNN ausgeführt. Dabei sind die elf Sensorwerte die Eingänge, ein zugehöriger Lenkwinkel der Ausgang.

Das KNN soll mit einem selbst implementierten RL-Verfahren, den sogenannten Genetischen Algorithmen (GA) ein natürliches Lenkverhalten erlernen. GA sind eine Gruppe von Algorithmen, die den natürlichen Prozess der Evolution nachahmen, um sich sukzessive einer optimalen Lösung anzunähern. Dabei entwickelt sich eine sogenannte Population, die aus mehreren Lösungskandidaten (Individuen) besteht, generationsweise durch Selektion, Rekombination und Mutation weiter. Eine Populationsgröße von 50 bedeutet hier, dass 50 verschiedene KNN in einer Generation simuliert werden. Der iterative Charakter des GA resultiert in tendenziell relativ vielen Simulationen.

In Abbildung 2 c) ist eine Übersicht über den grundlegenden Aufbau des Simulationsmodells dargestellt. Der GA befindet sich eigentlich außerhalb des Simulationsmodells. Die zugehörige Belohnungsfunktion greift auf Simulationsdaten zu, um das Verhalten der Individuen zu bewerten. Diese Information nutzt der GA, um die Verbindungsgewichte des KNN zu aktualisieren.

5.2 Konfiguration der Simulationsdurchläufe

Wie in Kapitel 2 beschrieben, verläuft die Bestimmung der KNN-Architektur und der Parameter des ML-Verfahrens immer empirisch. Darüber hinaus ist es wichtig beim

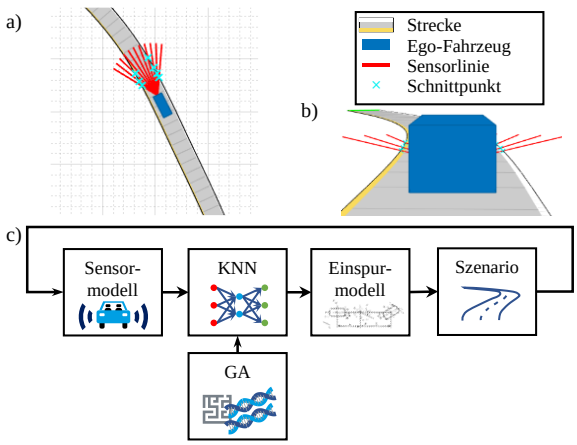

Abbildung 2: Überblick über den Anwendungsfall und die Modellbildung

Training von KNN auf die Generalisierungsfähigkeit zu achten. Das bedeutet in diesem Fall, dass das KNN in der Lage sein muss, auch auf unbekannten Strecken die automatisierte Querführung auszuführen. Um diesen Aspekt direkt während des Trainings zu berücksichtigen, werden mehrere Testdurchläufe direkt nach einzelnen Trainings durchgeführt. Es ist also erforderlich mehre verschiedene Simulationen zu konfigurieren.

Um sukzessive zu einer ideal ausgelegten Fahrfunktion zu gelangen, müssen mehrstufige Versuche in der Konfiguration der Simulationsdurchläufe implementiert werden. Dabei besteht der Simulationsdurchlauf immer aus mehreren ML Durchläufen. Ein ML Durchlauf beinhaltet immer ein Training einer KNN-Architektur mit einem GA-Parametersatz auf einer Strecke mit einer Belohnungsfunktion, gefolgt von vier Testdurchläufen auf weiteren Strecken. Während die Trainingsphase des GA aus vielen einzelnen Simulationen für die Individuen und Generationen besteht, wird bei den Testdurchläufen jeweils nur das beste Individuum aus dem Training geprüft. Die Anzahl der durchzuführenden Simulationen S_i pro ML Durchlauf i ergibt sich also in Abhängigkeit der Populationsgröße P_i, der Generationenzahl G_i und Anzahl der Testdurchläufe T_i . Die Anzahl der Simulationen im Gesamtdurchlauf N_G ist die Summe aller S_i für die verschiedenen ML Durchläufe bzw. -Konfigurationen K_{ML}:

$$N_G = \sum_{i \in K_{ML}} S_i = \sum_{i \in K_{ML}} P_i \cdot G_i + T \qquad (1)$$

Bei einer angenommenen Populationsgröße von 50 und einer Generationenzahl von 25, werden in einem ML Durchlauf folglich 1254 Simulationen durchgeführt.

Die erste Konfiguration sieht die Untersuchung von sechs Varianten der GA-Parametersätze für das Training eines Basis-KNN vor (Abbildung 3 a)). Das Basis-KNN ist ein nicht optimiertes KNN von dem man ausgeht, dass es die Funktion grundlegend erfüllen kann. Analog verhält es sich mit der Basis-Belohnungsfunktion. Der so ermittelte optimale GA-Parametersatz wird, wie in Abbildung 3 b) gezeigt, genutzt, um die beste aus zwölf vorkonfigurierten KNN-Architekturen zu bestimmen. Im letzten Schritt (Abbildung 3 c)) erfolgt die eigentliche Optimierung des KNN-Verhaltens durch weitere ML Durchläufe mit neun verschiedenen Belohnungsfunktionen. Die Anzahl der insgesamt durchgeführten Simulationen N_G liegt nach Gleichung (1) im hohen sechsstelligen Bereich. Ohne die automatische Modellkonfiguration der automatisierten Simulationsumgebung, hätte der benötigte Aufwand das vertretbare Maß überschritten.

5.3 Simulationsergebnisse und Auswertung

Nach dem automatischen Durchlauf aller Konfigurationen aus Abbildung 3, ist eine Funktion zur automatisierten Querführung entstanden. Das KNN kann auf beliebigen Strecken in einem Geschwindigkeitsbereich von 30 bis 70 km/h sicher und komfortabel die Querführung übernehmen. Für detailliertere Beschreibungen der Auslegung und Absicherung der Fahrfunktion aus diesem Anwendungsfall, sei auf die Vorarbeiten [5] und [16] verwiesen. Der Schwerpunkt dieses Beitrags liegt auf der Simulationsumgebung. Zu deren Auswertung wurde ein einzelner automatisierter ML-Durchlauf händisch in MATLAB programmiert und hinsichtlich der Laufzeit mit der Simulationsumgebung verglichen. Bei der händisch programmierten Version wurden auch alle Modellbestandteile und die Visualisierung selbst implementiert.

Der Vergleich zeigte, dass ein ML Durchlauf in der händisch programmierten Umgebung etwa 22 h und 36 min dauert. In der Simulationsumgebung für die automatisierte Modellkonfiguration hingegen, dauert ein ML Durchlauf mit identischer Konfiguration mit ca. 28 min nur etwa 2 % dieser Zeit. Weiterhin ist es möglich, die Visualisierung in der Simulationsumgebung abzuschalten. So lässt sich die benötigte Dauer je ML Durchlauf noch einmal vierteln.

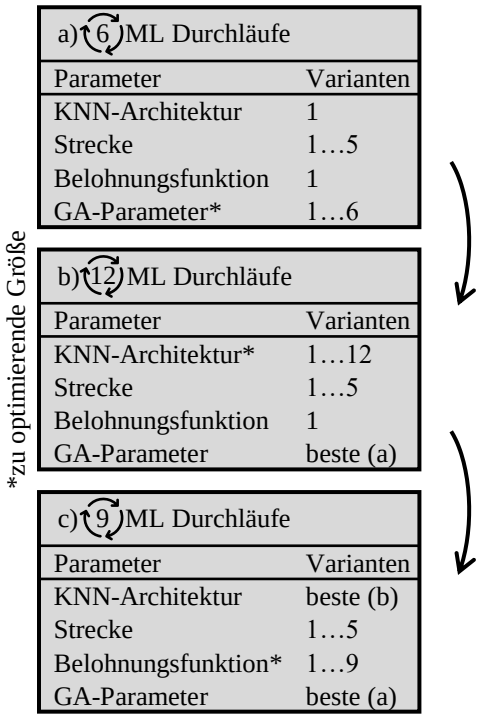

Abbildung 3: Ablauf des automatischen Simulationsdurchlaufs

6 Zusammenfassung und Ausblick

In diesem Beitrag wurde eine Simulationsumgebung für die automatisierte Modellkonfiguration zur Auslegung und Absicherung KI-basierter Fahrfunktionen vorgestellt. Ausgehend von einer Einleitung und Motivation wurden die Entwurfsmethodik und der Stand der Technik vorgestellt. Hieraus wurde die Notwendigkeit zur Entwicklung der vorgestellten Simulationsumgebung abgeleitet und Anforderungen für diese definiert. Die Umsetzung der Anforderungen wurde im Konzept und der Realisierung beschrieben. Das Ergebnis ist eine Simulationsumgebung auf Basis von MATLAB und Simulink, die einen hohen Kompatibilitätsgrad zu vorhandenen Entwicklungsumgebungen aufweist. Außerdem kann sie große Simulationsreihen nach einer benutzerfreundlichen Konfiguration in kurzer Zeit automatisch durchführen und visualisieren. Damit eignet sie sich hervorragend für den Einsatz in der vorgestellten Entwurfsmethodik. Schließlich wurde die Simulationsumgebung in einem Anwendungsbeispiel eingesetzt und so deren Mehrwert und Funktionsfähigkeit nachgewiesen. Zukünftige Arbeitsschritte umfassen unter anderem die Erweiterung der Modell- und Funktionsbibliothek sowie die Integration mit dSPACE ASM.

Danksagung

Diese Veröffentlichung entstand aus dem Teilprojekt "autoEVM" (*Ganzheitliches elektronisches Fahrzeugmanagement für autonome Elektrofahrzeuge*) (ZW 6-85030889), im Rahmen des Innovationsverbundes "autoMoVe" (*Dynamisch konfigurierbare Fahrzeugkonzepte für den nutzungsspezifischen autonomen Fahrbetrieb*), welcher vom Europäischen Fonds für regionale Entwicklung (EFRE) gefördert und vom Projektträger NBank verwaltet wird .

EUROPÄISCHE UNION

EUROPA FÜR NIEDERSACHSEN
Europäische Förderung für die niedersächsischen Regionen

Literatur

[1] Milz S., Schrepfer J. Is artificial intelligence the solution to all our problems? Exploring the applications of AI for automated driving. In: Bertram T. (eds) *Automatisiertes Fahren 2019*. Springer Vieweg, Wiesbaden, 2020.

[2] Schiekofer, P. et al. Maschinelles Lernen für das automatisierte Fahren. *ATZ Automobiltech Z*, vol. 121, 2019.

[3] Liu-Henke X. et. al. Holistic development of a full-active electric vehicle by means of a model-based systems engineering. *2016 IEEE International Symposium on Systems Engineering (ISSE)*, Edinburgh, UK, 2016.

[4] Stančin I., Jović A. An overview and comparison of free Python libraries for data mining and big data analysis. *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia, 2019.

[5] Yarom O. A. et. al. Artificial Neural Networks and Reinforcement Learning for model-based design of an automated vehicle guidance system. *12th International Conference on Agents and Artificial Intelligence (ICAART)*, Valletta, Malta, 2020.

[6] Reif K. Fahrerassistenzsysteme. In: *Automobilelektronik*. Vieweg+Teubner Verlag, 2012.

[7] Lyu H. et. al. Esnet: Edge-Based Segmentation Network for Real-Time Semantic Segmentation in Traffic Scenes. *2019 IEEE International Conference on Image Processing (ICIP)*, Taipei, Taiwan, 2019.

[8] Huang Z. et. al. Parameterized batch reinforcement learning for longitudinal control of autonomous land vehicles. *IEEE Trans. Syst., Man, Cybern, Syst.*, vol. 49, no. 4, 2019

[9] Kuutti S. et. al. A Survey of Deep Learning Applications to Autonomous Vehicle Control, *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, 2021.

[10] Alaoui C. Hybrid Vehicle Energy Management Using Deep Learning. *2019 International Conference on Intelligent Systems and Advanced Computing Sciences (ISACS)*, Taza, Marokko, 2019.

[11] Tirumala S.S. Evolving deep neural networks using coevolutionary algorithms with multi-population strategy. In: *Neural Comput & Applic*, vol. 32, 2020.

[12] Duriez T., Brunton S., Noack B. R. *Machine Learning Control*. Springer International Publishing, Cham, Switzerland, 2017.

[13] Deter D. et. al. Simulating the Autonomous Future: A Look at Virtual Vehicle Environments and How to Validate Simulation Using Public Data Sets. in *IEEE Signal Processing Magazine*, vol. 38, no. 1, 2021.

[14] Wang D. et. al. Deep object centric policies for autonomous driving. arXiv preprint arXiv:1811.05432, 2018.

[15] Zhang J., Cho K. Query-efficient imitation learning for end-to-end autonomous driving," arXiv preprint arXiv:1605.06450, 2016.

[16] Yarom O. A., Jacobitz S., Liu-Henke X. Design of Genetic Algorithms for the Simulation-Based Training of Artificial Neural Networks in the Context of Automated Vehicle Guidance. *2020 19th International Conference on Mechatronics - Mechatronika (ME)*, Prague, Czech Republic, 2020.