Karlsruhe Institute of Technology
Fakultät für Elektrotechnik

Institut für Technik der
Informationsverarbeitung (ITIV)

Erweiterung eines Datenmodells zur Übertragung von Fahrdaten eines automatisierten Fahrzeugs in eine virtuelle Testumgebung um Informationen der statischen Umgebung mittels Machine Learning-Methoden

Bachelorarbeit von

Philipp Metzger

11. Mai 2020

Institutsleitung:	Prof. Dr.-Ing. Dr. h. c. J. Becker Prof. Dr.-Ing. E. Sax Prof. Dr. rer. nat W. Stork
Prüfer:	Prof. Dr.-Ing. E. Sax
Korreferent:	Prof. Dr. S. Nickel (Institut für Operations Research (IOR))
Betreuer:	Dipl.-Wi.-Ing. M. Sc. Raphael Pfeffer

Danksagung

Diese Bachelorarbeit entstand am Institut für Technik der Informationsverarbeitung (ITIV) des Karlsruher Instituts für Technologie (KIT).

Ich danke meinem Betreuer Raphael Pfeffer, der mir durch das Stellen der richtigen Fragen und mit sehr hilfreichem Feedback dabei geholfen hat, eine Arbeit zu verfassen, mit der ich zufrieden bin, und, der außerdem stets sicherstellte, dass die Bearbeitung der Aufgabenstellung für mich nicht nur mit Interesse, sondern auch mit Spaß verbunden war.

Des Weiteren danke ich Jan Metzger und Julius Floßmann, die sich die Zeit genommen haben, diese Arbeit gegenzulesen und mich auf Unstimmigkeiten aller Art aufmerksam zu machen.

Zusammenfassung

Die vorliegende Arbeit beschreibt die Erweiterung eines Datenmodells, dessen Zweck die Übertragung realer Fahrzeugdaten in eine Simulationsumgebung ist. Die Übertragung der Fahrzeugdaten soll ermöglichen, reale Fahrszenarien in der Simulationsumgebung abzubilden und mit ihnen automatisierte Fahrfunktionen zu testen. Die Erweiterung des Datenmodells hat das Ziel, in das Modell neben dynamischen Informationen auch Informationen von statischer Natur, wie zum Beispiel die Positionen von Verkehrsschildern, einzubetten. Des Weiteren beschreibt die vorliegende Arbeit die konkrete Entwicklung und Implementierung von Funktionen mittels derer aus Sensordaten Informationen über Verkehrsschilder und die Höhe und die Breite von beteiligten Fahrzeugen extrahiert und in das Modell übertragen werden können. Außerdem enthält diese Arbeit Ausführungen über den Zweck der Modellerweiterung und der Entwicklung und Implementierung der Funktionen sowie Beschreibungen der zum Verständnis der Arbeit notwendigen Grundlagen und eine Auswertung der Ergebnisse.

Urheberrecht

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit selbständig und unter Beachtung der Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) in der aktuellen Fassung angefertigt habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich übernommene Stellen als solche kenntlich gemacht.

Karlsruhe, den 11. Mai 2020

_____

Philipp Metzger

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Bevor ein technisches System im Straßenverkehr Verwendung findet, ist eine Freigabe erforderlich. Die Freigabe wird erteilt, nachdem durch einen Testprozess von ausreichendem Umfang gesichert wurde, dass die Fahrfunktion in ihrem Anwendungsbereich ausreichend wenig Fehler macht. Essenziell ist hierbei, dass die Testabdeckung ausreichend ist, dass also die Anzahl an Testfällen beziehungsweise gefahrenen Testkilometern ausreichend hoch ist.

Mit zunehmender Automatisierung von Fahrfunktionen geht außerdem eine Vergrößerung des Anwendungsbereiches derselben einher. Auf die verschiedenen Automatisierungsgrade wird in 2.1.2 eingegangen. Ein Parkassistent hat beispielsweise im Vergleich zu einem Autobahn-Piloten einen kleinen Anwendungsbereich.

Aus der beschriebenen Situation resultiert folgende Problematik: Einen Prototyp mit einem höheren Automatisierungsgrad auf konventionelle Weise zu testen kostet sehr viel Geld. Dies hat drei Gründe: Der erste Grund ist, dass die für einen aussagekräftigen Test benötigte Teststrecke sehr lang ist. Auf Ansätze zur Bestimmung der Länge der benötigten Teststrecke wird in 2.2 näher eingegangen. Der zweite, vom Automatisierungsgrad unabhängige Grund ist, dass für die Testfahrt ein Testfahrer benötigt wird. Der dritte, ebenfalls vom Automatisierungsgrad unabhängige Grund ist, dass die Testfahrt nach jeder Veränderung am Prototyp wiederholt werden muss.

Ein naiver Lösungsansatz hierfür ist der folgende: Man könnte Testfahrten anstatt auf realen Straßen in einer Simulationsumgebung durchführen. Das wäre kostengünstig und ginge schnell.

Dieser Ansatz ist im Prinzip vielversprechend, birgt jedoch Probleme, die unbedingt bedacht und umgangen werden müssen. Um diese zu verstehen, muss zuerst kurz der klassische Fahrsimulationsansatz erklärt werden: Dieser besteht darin, Fahrsituationen zu erzeugen, indem die bekannten Einflussfaktoren mit ihren Wertebereichen kombiniert werden. Mit diesem Ansatz kann eine Vielzahl von Fahrsituationen erzeugt werden. Allerdings

gibt es eventuell Fahrsituationen, die durch diesen Ansatz nicht abgedeckt sind (beispielsweise ein Blitzeinschlag oder eine Person, die einen Stein von einer Brücke fallen lässt). Bezüglich jeder dieser nicht abgedeckten Fahrsituationen gibt es zwei Möglichkeiten:

1. Die Person, die den Test entwirft, denkt daran, sie manuell zur Testumgebung hinzuzufügen.
2. Die Person, die den Test entwirft, denkt nicht daran, sie manuell zur Testumgebung hinzuzufügen.

In diesem zweiten Fall liegt natürlich das Problem: Es kann nicht garantiert werden, dass alle Fahrsituationen, die im realen Straßenverkehr auftreten könnten, Teil der zum Testen verwendeten Simulation sind. Das bedeutet, dass eventuell nicht herausgefunden wird, dass der Prototyp in einer solchen Situation unerwünschtes Verhalten aufweist. Was deshalb benötigt wird, ist eine wirkungsvolle Möglichkeit, große Mengen an realen Fahrsituationen in eine Simulation zu überführen. Wenn die Menge solcher realer Fahrkilometer groß genug ist, also eine gewisse Schranke überschreitet, kann es als statistisch gesichert angesehen werden, dass der Prototyp, der mithilfe dieser Menge an Fahrkilometern getestet wird, bestimmten Qualitätsanforderungen genügt.

Konkret bedeutet dies, dass Mittel und Wege zu finden sind, um eine ausreichend große Menge an gefahrenen Kilometern und somit an realistischen Fahrsituationen unter möglichst geringem Aufwand und mit einer möglichst genauen Abbildung der realen Größen in eine Simulationsumgebung zu überführen. Sobald dies erreicht ist und die realen Fahrdaten auf geeignete Weise in die Simulationsumgebung überführt wurden, können die so erhaltenen Fahrszenarien zum Testen von beliebigen Prototypen verwendet werden. Wichtig ist hierbei, dass sowohl durch die Überführung der Daten, als auch durch die Simulationsumgebung eine realitätsgetreue Abbildung der zugrundeliegenden wahren Fahrsituationen erfolgt. Die hierfür verwendeten Funktionen und Programme müssen also auf ihre Validität überprüft werden, bevor sie zu diesem Zwecke eingesetzt werden können.

Das Fahren der Teststrecke beziehungsweise das Sammeln von Fahrdaten muss nun nicht mehr unbedingt durch einen zu diesem Zwecke beschäftigten Testfahrer durchgeführt werden. Vielmehr kann es zum Beispiel geschehen, indem in eine bestimmte Klasse von Fahrzeugen, welche an Kunden verkauft wird, diejenigen Sensoren eingebaut werden, die von Interesse sind. Mit ihnen werden dann über eine bestimmte Zeitspanne hinweg Daten aufgezeichnet und am Ende ausgewertet.

Die für das Testen einer beliebigen Anzahl von Prototypen erforderliche Teststrecke muss durch den beschriebenen Ansatz also insgesamt nur ein mal gefahren werden, wobei die Strecke sogar auf eine Flotte von Fahrzeugen aufgeteilt werden kann. Daraus resultiert eine erhebliche Aufwands- und somit Kostenreduktion beim Testen von Fahrfunktionen.

1.2 Ziele und Grenzen der Arbeit

Diese Arbeit entsteht aufbauend auf dem Konzept und der Implementierung einer Masterarbeit, die ebenfalls am ITIV entstand [Kha19]. In dieser wurde ein Modell entwickelt in

dem Metainformationen über reale Fahrszenarien gespeichert werden können. Diese Metainformationen können im Anschluss in eine Simulationsumgebung verschoben werden. Der Fokus lag hierbei auf der Speicherung von dynamischen Informationen wie beispielsweise der Geschwindigkeit und der Position des Ego-Fahrzeugs und anderer Verkehrsobjekte. Daran anknüpfend verfolgt meine Arbeit zwei Ziele:

1. Das erste Ziel ist das oben genannte Modell so zu erweitern, dass in diesem neben dynamische Informationen auch statische gespeichert und im Anschluss in die Simulationsumgebung verschoben werden können. Mit statischen Informationen sind Informationen gemeint, die sich nicht im Zeitverlauf, jedoch eventuell im Streckenverlauf ändern. Beispiele für solche statischen Informationen sind die Positionen und Identifikationen der entlang der Fahrstrecke sichtbaren Verkehrsschilder, die Abmessungen der Verkehrsobjekte, die sich während des betrachteten Szenarios[1] im Sichtfeld des Ego-Fahrzeugs befinden, die Anzahl der Fahrbahnspuren und die Krümmung der Fahrbahn. An die Modellerweiterungen werden hierbei zwei Anforderungen formuliert:

   (a) Die im Modell gespeicherten Informationen sollen möglichst exakt die realen Größen wiedergeben, die sie repräsentieren, und

   (b) in abstrahierter Form gespeichert werden; es soll also eine gewisse Reduktion der Rohdaten auf eine möglichst minimale Menge an gespeicherten Daten stattfinden.

2. Das zweite Ziel ist die konzeptionelle Entwicklung, die Implementierung und die Validierung von Methoden, um aus Daten, die aus den Sensoren eines realen Fahrzeugs stammen, die oben genannten Informationen zu extrahieren und in dem erweiterten Modell zu speichern. Der Fokus liegt hierbei auf der Bestimmung der Breiten und Höhen der involvierten Verkehrsobjekte und der Positionen und Identifikationen der entlang der Fahrstrecke sichtbaren Verkehrsschilder. Dabei wird während der konzeptionellen Entwicklung auch darauf eingegangen, wie allgemeine statische Informationen im Modell integriert werden können. Die Implementierung hingegen beschränkt sich gänzlich auf die genannten zwei Beispielaspekte.

Die Extraktion von dynamischen Informationen und die Überführung dieser in das Modell stellt einen weiteren Aspekt der übergeordneten Problemstellung dar, auf den in der vorliegenden Arbeit jedoch nicht eingegangen wird. Des Weiteren wird nur die Fahrt auf autobahnähnlichen Streckenabschnitten betrachtet.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit umfasst sechs Kapitel:

In der Einleitung werden Motivation, Ziele und Grenzen dieser Arbeit, sowie ihr Aufbau beschrieben.

[1] Der Begriff „Szenario“ wird in 2.3 definiert.

Im Kapitel „Grundlagen“ werden einige Begriffe des automatisierten Fahrens eingeführt, die Grenzen des konventionellen Testansatzes beleuchtet und die Begriffe „Szene“ und „Szenario definiert. Des Weiteren wird das in der vorangegangenen Masterarbeit entwickelte Datenmodell vorgestellt und es werden einige für die vorliegende Arbeit relevante geometrische Zusammenhänge beschrieben. Schließlich werden einige Begriffe des Themenfelds des maschinellen Lernens eingeführt und die technischen Grundlagen für zwei in der vorliegenden Arbeit verwendeten Technologien erläutert.

Das Kapitel „Konzept“ beschäftigt sich mit den an diese Arbeit gestellten Anforderungen und den getroffenen Annahmen und enthält eine Beschreibung des der Implementierung zugrundeliegenden Konzeptes.

Im Kapitel „Implementierung“ wird zunächst auf die verwendete Simulationssoftware CarMaker eingegangen. Dann werden die Daten beschrieben, die Verwendung fanden. Anschließend werden die in Matlab implementierten Funktionen beschrieben.

Im Kapitel „Auswertung“ werden zunächst die Validierungs- und Testkriterien beschrieben. Dann wird auf das Validierungs- und Testvorgehen eingegangen und die Ergebnisse der Validierungen und Tests genannt. In Anschluss wird diskutiert inwieweit die Ergebnisse den Anforderungen entsprechen und mögliche Verbesserungspotenziale beleuchtet.

Das letzte Kapitel „Zusammenfassung und Ausblick“ enthält eine Zusammenfassung der vorliegenden Arbeit sowie eine Aufstellung relevanter Aspekte, die im Rahmen dieser Arbeit noch nicht zur zur Vollständigkeit adressiert wurden.

Der Anhang dieser Arbeit enthält weitere Ausführungen, die als Ergänzung von Abschnitt 3.3.3 zu verstehen sind.

Kapitel 2

Grundlagen

2.1 Automatisiertes Fahren

2.1.1 Begriffsdefinition

Der Verband der Automobilindustrie definiert den Begriff des automatisierten Fahrens in [Ver15] als „das selbständige, zielgerichtete Fahren eines Fahrzeugs im realen Verkehr mit bordeigenen Sensoren, nachgeschalteter Software und im Fahrzeug gespeichertem Kartenmaterial für die Erfassung der Fahrzeugumgebung“.

2.1.2 Automatisierungsgrade automatisierter Fahrfunktionen

Es werden sechs verschiedene Automatisierungsgrade von automatisierten Fahrfunktionen unterschieden, von denen die Bundesanstalt für Straßenwesen in [GAA+12] fünf beschreibt. In [Ver15] werden diese fünf Automatisierungsgrade durch einen sechsten Grad „Fahrerlos“ erweitert. Es folgt eine Beschreibung der sechs Automatisierungsgrade nach [GAA+12, Ver15], teilweise ergänzt um beispielhafte Systemausprägungen:

- 0 - Driver Only: Der Fahrer führt während der gesamten Fahrt die Längsführung (Beschleunigen und Verzögern) und die Querführung (Lenken) aus. Es ist kein eingreifendes (Fahrerassistenz-)System aktiv.
- 1 - Assistiert: Der Fahrer führt dauerhaft entweder die Quer- oder die Längsführung aus. Die jeweils andere Fahraufgabe wird in gewissen Grenzen vom System durchgeführt, wobei eine dauerhafte Überwachung des Systems durch den Fahrer notwendig ist. Der Fahrer muss jederzeit zur vollständigen Übernahme der Fahrzeugführung bereit sein. Beispiele hierfür sind:
  - Adaptive Cruise Control: Das System übernimmt die Längsführung mit adaptiver Abstands- und Geschwindigkeitsregelung.

  - Parkassistent: Der Fahrer steuert die Längsführung, während die Querführung durch den Parkassistent durchgeführt wird.

- 2 - Teilautomatisiert: Das System übernimmt für eine gewisse Zeit und/oder in spezifischen Situationen die Längs- und Querführung, wobei der Fahrer das System dauerhaft überwachen muss. Der Fahrer muss jederzeit zur vollständigen Übernahme der Fahrzeugführung bereit sein. Ein Beispiel hierfür ist der Autobahnassistent: Automatische Längs- und Querführung bis zu einer oberen Geschwindigkeitsbegrenzung, wobei der Fahrer dauerhaft überwachen und bei Übernahmeaufforderung sofort reagieren muss.

- 3 - Hochautomatisiert: Das System übernimmt die Längs- und Querführung für einen gewissen Zeitraum in spezifischen Situationen, wobei der Fahrer das System nicht dauerhaft überwachen muss. Bei Bedarf wird der Fahrer zur Übernahme der Fahraufgabe mit ausreichender Zeitreserve aufgefordert. Beim Erreichen von Systemgrenzen, welche alle vom System erkannt werden, muss manchmal der Fahrer die Führung übernehmen, manchmal überführt sich das System jedoch selbstständig in einen Zustand, der sich wieder innerhalb der Systemgrenzen befindet. Das System ist nicht in der Lage, aus jeder Ausgangssituationen den risikominimalen Zustand herbeiführen. Ein Beispiel hierfür ist der Autobahn-Chauffeur: Automatische Längs- und Querführung bis zu einer oberen Geschwindigkeitsgrenze, wobei der Fahrer nicht dauerhaft überwachen, nach Übernahmeaufforderung aber mit gewisser Zeitreserve reagieren muss.

- 4 - Vollautomatisiert: Das System übernimmt Längs- und Querführung vollständig in einem definierten Anwendungsfall, wobei der Fahrer das System nicht überwachen muss. Vor dem Verlassen des Anwendungsfalles fordert das System den Fahrer mit ausreichender Zeitreserve zur Übernahme der Fahraufgabe auf. Erfolgt dies nicht, überführt sich das System in einen risikominimalen Zustand. Systemgrenzen werden alle vom System erkannt und das System ist in allen Situationen in der Lage, in den risikominimalen Systemzustand zurückzuführen. Ein Beispiel hierfür ist der Autobahn-Pilot: Automatische Längs- und Querführung bis zu einer oberen Geschwindigkeitsgrenze, wobei der Fahrer nicht überwachen muss. Reagiert der Fahrer nicht auf eine Übernahmeaufforderung, so bremst das Fahrzeug in den Stillstand herunter.

- 5 - Fahrerlos: Das System übernimmt die Fahraufgabe vollumfänglich bei allen Straßentypen, Geschwindigkeitsbereichen und Umweltbedingungen. Ein Fahrer ist nicht erforderlich.

2.1.3 Sensorik eines autonom fahrenden Fahrzeugs

In Fahrzeugen kommt vielfältige Sensorik zum Einsatz. Im Jahr 2018 lag die Anzahl an (diskreten) Sensoren in einem Fahrzeug je nach Markt und Ausstattung bei etwa 30 bis über 150 [Dix18]. Durch die fortschreitende Automatisierung entstehen neue Anforderungen an die Sensoren von Fahrzeugen. In [Eym19] werden im Kontext des automatisierten Fahrens fünf Sensortypen unterschieden: Lidar, Mono- oder Stereokamera, Radar,

Ultraschall und Infrarot-Kamera. Abbildung 2.1 zeigt diese Sensoren und eine kurze Be-

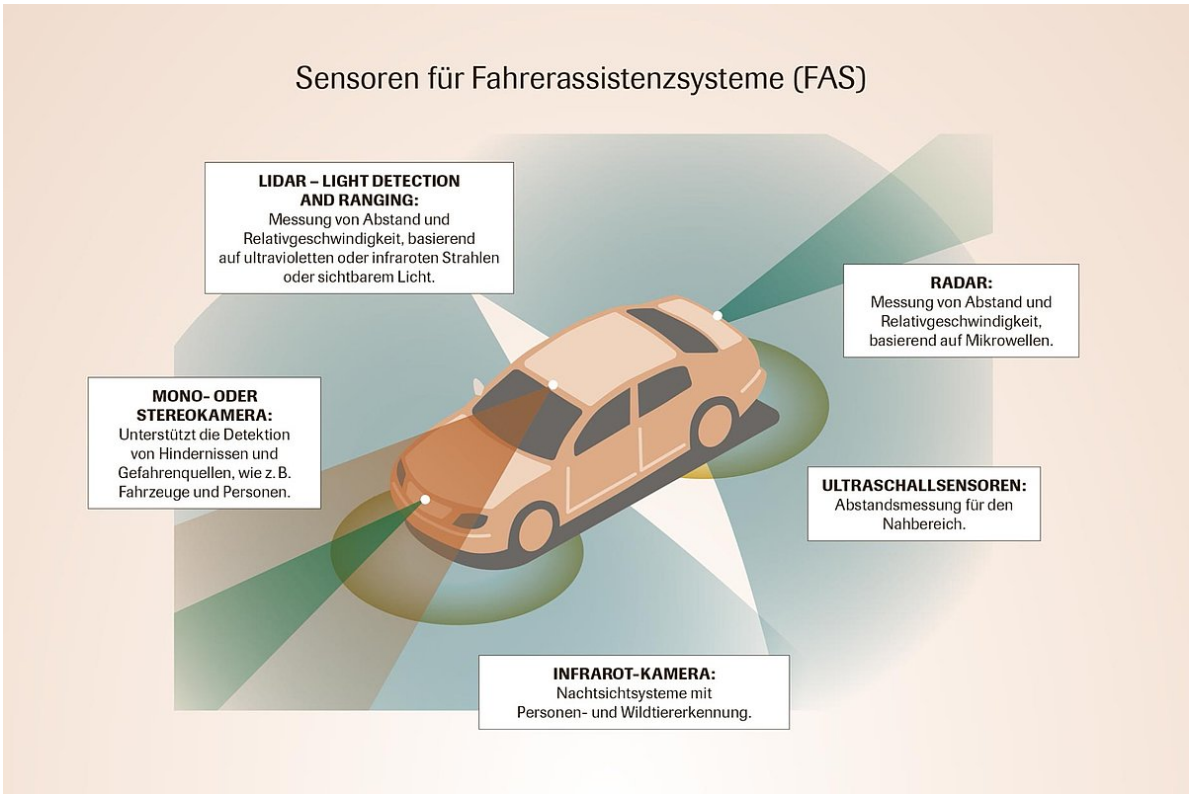

Abbildung 2.1: Sensoren für Fahrerassistenzsysteme (FAS). Bild aus [Eym19].

schreibung ihres Zwecks. In [Dix18] werden zusätzlich noch Dead Reckoning Sensoren (Koppelnavigation) und Sensoren für adaptive Lichtsteuerung genannt. Der folgende Abschnitt gibt angelehnt an [Dix18, Eym19] einen kurzen Überblick über die relevanteren Sensortypen:

- Kameras: Liefern Informationen, die für viele Zwecke wie zum Beispiel Fahrspurmarkierung und Verkehrszeichenerkennung verwendet werden. Sie haben eine hohe Winkelauflösung, liefern bei ungünstigen Wetterbedingungen wie Nebel jedoch schlechtere Resultate. Kameras haben eine Reichweite von 100 bis 240 Metern. [Eym19]

- Radarsysteme: Werden für genaue Geschwindigkeits- und Entfernungsmessung verwendet und weisen eine große Wetterunabhängigkeit auf. Ihre Winkelauflösung ist geringer als die von Kameras. Es gibt Mid-Range-Radarsysteme mit bis zu 30 Metern und Long-Range-Radarsysteme mit bis zu 250 Metern Reichweite. [Eym19]

- Lidar: Die Bezeichnung steht für „Light Detection and Ranging“. Es handelt sich hierbei und eine auf Lasern basierte Technologie, die Entfernungen zu ruhenden und bewegten Objekten misst. Lidar hat eine hohe Winkelauflösung und kann 360° abdecken. Bei guten Wetterbedingungen beträgt die Reichweite bis zu 200 Meter. [Eym19]

- Dead Reckoning Sensoren: Basieren „auf der Berechnung der aktuellen Position durch die Verwendung zuvor festgestellter Lokalisationsdaten und durch Verrücken dieser Position auf Grundlage bekannter Geschwindigkeitsmaße über einen bestimmten Zeitraum“ [Dix18]. Diese Technologie gewinnt immer mehr an Bedeutung, wobei sie vor allem für die Automatisierungsgrade 3 und höher (siehe 2.1.2) erforderlich ist. [Dix18]

2.2 Warum der konventionelle Testansatz an seine Grenzen stößt

Wachenfeld und Winner zeigen in [WW15] beispielhaft das Folgende: Um auf einem Signifikanzniveau von 5% abzusichern, dass eine vollautomatisierte Fahrfunktion „hinsichtlich des Kriteriums der erwarteten Anzahl gefahrener Kilometer zwischen zwei Unfällen mit Getöteten“ [Sch17, S. 5] doppelt so gut wie ein menschlicher Fahrer ist, was im Rahmen dieses Beispiels als Minimalanforderung für die Akzeptanz der Fahrfunktion angenommen wurde, ist eine Teststrecke zu absolvieren, die etwa zehnmal so lang ist wie die aus Unfallstatistiken ermittelte durchschnittliche Strecke zwischen zwei tödlichen Unfällen. Schuldt, der in [Sch17] diese Überlegung mit Zahlen des Statistischen Bundesamtes aus dem Jahr 2014 nachvollzieht, berechnet, dass folglich bei einem Durchschnittswert von 217 Millionen Kilometern zwischen zwei Unfällen mit Todesfolge [Lam16] die Länge der benötigte Teststrecke 2,17 Milliarden Kilometer betrüge. Falls sich die Einsatzumgebung des automatisierten Fahrsystems auf die Autobahn beschränken würde, ergebe sich, so Schuldt, sogar eine noch größere durchschnittliche Distanz von 614 Millionen Kilometern zwischen zwei tödlichen Unfällen und somit eine benötigte Anzahl von Testfahrtkilometern von 6,14 Milliarden. Da darüber hinaus nach einer Änderung am automatisierten Fahrsystem oder einer Veränderung der Testgegebenheiten (wie zum Beispiel der Veränderung der Anzahl der Fahrzeuginsassen) das Fahren der genannten Streckenkilometer wiederholt werden müsste [WW15], liegt auf der Hand, dass ein Testansatz, bei dem jeder Prototyp die geforderten Testkilometer selbst absolvieren muss, bei vollautomatisierten Fahrfunktionen ökonomisch nicht sinnvoll ist.

Ein weiterer Negativaspekt eines solchen streckenbasierten Ansatzes ist, so Schuldt [Sch17], dass dadurch, dass nicht vorausgesagt werden kann welche Ereignisse während der Testfahrt eintreten werden, die Testtiefe beziehungsweise die Testabdeckung[1] nur ansatzweise geschätzt werden kann. Aus dem selben Grund werde durch eine Erhöhung der Anzahl der Testkilometer nur bedingt eine Erhöhung der Testabdeckung erreicht, so Schuldt.

Eine Lösung dieses Problems könnte, wie in 1.1 bereits erklärt wurde, sein, zuerst alle relevanten Daten über eine (eventuell sogar durch ein ganze Flotte von Fahrzeugen) real gefahrene Strecke zu sammeln und in einem geeigneten Modell zu speichern. Anschließend können diese Informationen dann verwendet werden um in einer geeigneten Simulationsumgebung beliebige Prototypen zu testen.

[1] Der Begriff der Testabdeckung wird in [Sch17, Kapitel 2.2.1] vorgestellt.

2.3 Szene und Szenario

In der vorliegenden Arbeit werden die Begriffe „Szene“ und „Szenario“ wiederholt verwendet. Sie werden deshalb im Folgenden definiert. Die Definitionen werden so wiedergeben, wie sie durch Schuldt in [Sch17] eingeführt wurden. Beide Definitionen gehen auf Arbeiten von Ulbrich et al. [UMR+15] und Reschka [Res17] zurück:

> Definition „Szene“:
>
> Eine Szene beschreibt eine Momentaufnahme des Umfelds, welche die beweglichen und unbeweglichen Elemente des Umfelds, die Selbstrepräsentation aller Akteure und Beobachter und die Verknüpfung dieser Entitäten umfasst. Einzig eine Szenenrepräsentation in einer simulierten Welt kann allumfassend sein (objektive Szene). In der realen Welt ist sie immer unvollständig, fehlerbehaftet, unsicherheitsbehaftet und aus der Perspektive eines oder mehrerer Beobachter (subjektive Szene).

Bei beweglichen Elementen handelt es sich, so Reschka, um Elemente, die die Fähigkeit haben sich zu bewegen. Ein Akteur ist ein Element, das selbst handelt. Ein Beobachter hingegen ist ein wahrnehmendes Element innerhalb der Szene oder betrachtet sie als Ganzes [Res17, Seite 59].

> Definition „Szenario“:
>
> Ein Szenario beschreibt die zeitliche Entwicklung von Szenenelementen innerhalb einer Folge von Szenen, welche mit einer Startszene beginnt. Aktionen und Ereignisse ebenso wie Ziele und Werte können spezifiziert werden, um diese zeitliche Entwicklung in einem Szenario festzulegen. Im Gegensatz zu Szenen decken Szenarien eine gewisse Zeitspanne ab.

Abbildung 2.2 aus der Arbeit von Ulbrich et al. [UMR+15] verdeutlicht den Aufbau eines Szenarios als zeitliche Abfolge von Aktionen oder Ereignissen, welche hier als die Kanten eines Graphen und Szenen, welche als die Knoten des selben Graphen dargestellt sind.

2.4 Das in einer vorangegangenen Masterarbeit entwickelte Datenmodell

Die vorliegende Arbeit kann als Fortführung der Masterarbeit [Kha19] betrachtet werden, in der das Datenmodell entwickelt wurde, das in der vorliegenden Arbeit verwendet, erweitert und befüllt wird. Dieses Datenmodell stellt eine erste Version der Struktur dar, die in Zukunft zur Speicherung der Informationen dienen könnte, die durch das Fahren von realen Fahrstrecken entstehen und - sobald in geeigneter Weise im Modell gespeichert - die daraus resultierenden simulierten Fahrszenarien definieren.

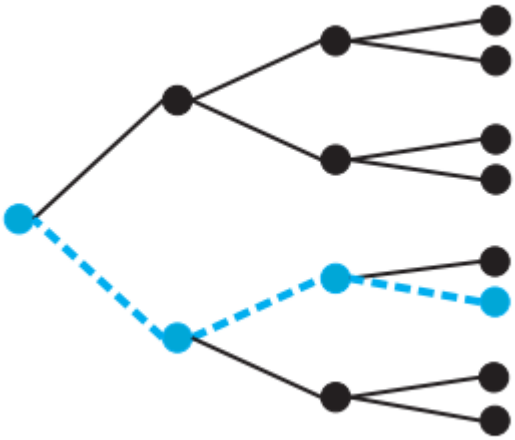

Abbildung 2.2: Ein Szenario (blau gestrichelt) als zeitliche Abfolge von Aktionen/Ereignissen (Kanten) und Szenen (Knoten). Bild und Bildbeschreibung stammen aus [UMR+15, Seite 28].

Bei dem Datenmodell handelt es sich um mehrere Matrizen, die jeweils ein Verkehrsobjekt repräsentieren und eine Matrix, die das Ego-Fahrzeug repräsentiert. Die Spalten dieser Matrizen korrespondieren mit den Zeitpunkten, zu denen von mindestens einem der Verkehrsobjekte oder dem Ego-Fahrzeug ein Fahrmanöver eingeleitet wurde. Diese sind von links nach rechts zeitlich aufsteigend sortiert. Die Zeilen der Matrizen enthalten die Belegungen von je einer Variable pro Zeile zu diesen Zeitpunkten. Die Zeilen der Matrizen können auch als mit jeweils einer Variablen korrespondierende Zeilenvektoren aufgefasst werden, die jeweils die Variablenbelegungen der entsprechenden Variable über den betrachteten Zeitraum beinhalten; die Bezeichnungen dieser Vektoren und ihre Bedeutungen, so wie sie in [Kha19] definiert werden, sind in Tabelle 2.1 dargestellt.

Für das Ego-Fahrzeug und für jedes im betrachteten Zeitraum detektierte Verkehrsobjekt wird eine solche Matrix erstellt. Die Gesamtheit dieser Matrizen beschreibt das ganze betrachtete Szenario. Die Struktur, die entsteht, wenn diese Matrizen hintereinander platziert werden kann auch als ein Tensor aufgefasst werden; zur Verdeutlichung des Aufbaus dieses Tensors ist dieser in Abbildung 2.3 dargestellt.

Um Übereinstimmung aller im Tensor enthaltenen Matrizen hinsichtlich der Spaltendimension zu erzielen, enthalten alle Matrizen alle Zeitpunkte, zu denen von mindestens einem der Verkehrsobjekte oder dem Ego-Fahrzeug ein Fahrmanöver eingeleitet wurde. Im Falle, dass ein solcher Zeitpunkt für ein Verkehrsobjekt keinen Manöverbeginn darstellt, werden in der Matrix dieses Verkehrsobjektes in der entsprechenden Spalte alle Variablen mit -9999 belegt. Das Ergebnis dieses Vorgehens ist in Abbildung 2.4 beispiel-

[2] In der dem Autor vorliegenden Implementierung handelt es sich hierbei nicht um den relativen Längsabstand zum Ego-Fahrzeug, sondern um eine im Zeitverlauf monoton steigende Größe. Dies ist jedoch offensichtlich nicht beabsichtigt.

Variable	Bedeutung
$\vec{t}$	Zeitpunkt von Manöverbeginn
$\vec{v_s}$	Anfangsgeschwindigkeit zu Manöverbeginn
$\vec{v_e}$	Geschwindigkeit bei Manöverende
$\vec{\triangle t}$	Dauer des Manövers
$\vec{\triangle x}$	während des Manövers zurückgelegte Distanz
$\vec{y_s}$	laterale Position zu Manöverbeginn
$\vec{y_e}$	laterale bei Manöverende
$\vec{d_{s_x}}$	relativer Abstand zum Ego-Fahrzeug in Längsrichtung[2] bei Manöverbeginn
$\vec{d_{s_y}}$	relativer Abstand zum Ego-Fahrzeug in Querrichtung bei Manöverbeginn
$\vec{L_{long}}$	Kodierung der Art des Längsmanövers (1: Beschleunigen, 0: Cruise, -1: Bremsen)
$\vec{L_{lat}}$	Kodierung der Art des Quermanövers 1: Spurwechsel rechts, 0: Spur halten, -1: Spurwechsel links
$\vec{L_{state,long}}$	Kodierung der relativen räumlichen Relation zum Ego-Fahrzeug in Längsrichtung -1: Hinter, 0: Neben, 1: Vor
$\vec{L_{state,lat}}$	Kodierung der relativen räumlichen Relation zum Ego-Fahrzeug in Querrichtung -1: Links, 0: Gleiche Spur, 1: Rechts

Tabelle 2.1: Darstellung der Modellvariablen und ihre Bedeutung. Variablenbezeichnungen übernommen aus [Kha19].

haft dargestellt.

Vier der Modellvariablen - $\vec{d_{s_x}}$, $\vec{d_{s_y}}$, $\vec{L_{state,long}}$ und $\vec{L_{state,lat}}$ - stellen Größen dar, die den relativen Abstand beziehungsweise die relative Position von Verkehrsobjekten zum Ego-Fahrzeug beschreiben. Folglich müssten die mit diesen Variablen korrespondierenden Zeilen in der Matrix des Ego-Fahrzeuges Nullwerte enthalten. Für das vorliegende Modell wurde entschieden diese Zeilen ebenfalls vollständig mit dem Wert -9999 belegt. Das Ergebnis dieses Vorgehens ist in Abbildung 2.5 dargestellt.

Zusätzlich zum oben beschriebenen Tensormodell wird durch die in [Kha19] vorgestellte Implementierung noch eine Datenstruktur erstellt, welche die mit -9999 befüllten Zeilen und Spalten nicht enthält. Der Zweck dieser zweiten Struktur ist die Überführung der Daten in die Simulationsumgebung CarMaker.

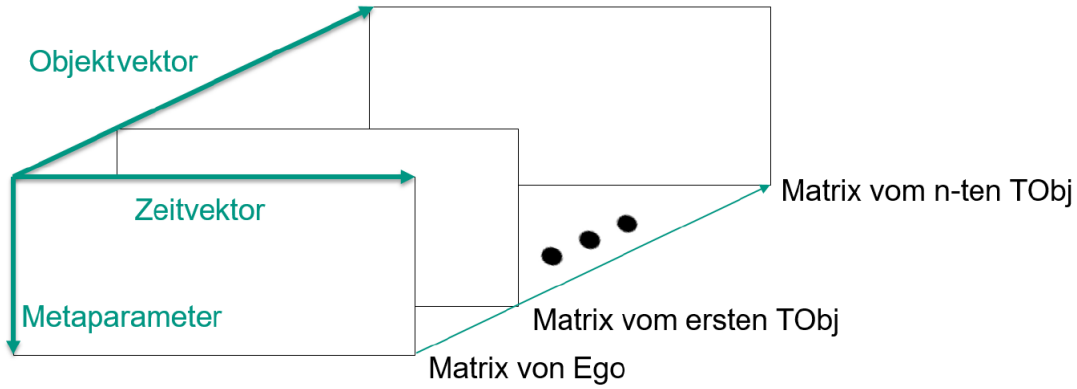

Abbildung 2.3: Geometrische Darstellung des in der vorangegangenen Arbeit entwickelten Tensor-Modells. Bild aus [Kha19].

2.5 Optik und geometrische Zusammenhänge

2.5.1 Verschiedene Typen von Kameralinsen

Jede Linse besitzt eine Projektionsfunktion, die den Zusammenhang zwischen dem Einfallswinkel eines Lichtimpulses und der Position der Abbildung desselben im resultierenden Bild beschreibt[Bet05, S. 11]. Diese Projektionsfunktion und somit der Linsentyp sind für diese Arbeit von Bedeutung, da von ihr abhängt, wo im Bild ein Objekt, das sich im Blickfeld einer Kamera aufhält, abgebildet wird. Folglich hängt von der Gestalt der Linse und ihrer Projektionsfunktion ebenso ab, wo im Raum sich ein Objekt, das im Kamerabild zu einem bestimmten Zeitpunkt abgebildet wurde, zu diesem Zeitpunkt befand. In Tabelle 2.2 sind einige Linsentypen und ihre Projektionsfunktionen, sowie ihre Eigenschaften dargestellt. Bei dieser Auswahl von Linsentypen handelt es sich um die Typen, die in IPG CarMakers IPGMovie als vordefinierte Linsen auswählbar sind. Dabei handelt es sich um die konventionelle rechtlinige Linse ohne Verzerrung und um einige Fisheye-Linsen. Fisheye-Linsen besitzen immer eine Verzerrung [Bet05, S.11]. Mit f sei die Brennweite (engl.: focal length) der jeweiligen Linse, mit Θ der Einfallswinkel des Lichtimpulses in die Linse (also der Winkel, der von den Richtungsvektoren des Lichtimpulses und der Blickachse der Kamera aufgespannt wird) und mit r der vom Bildmittelpunkt aus gemessene euklidische Abstand der Abbildung des Lichtimpulses in der Bildebene bezeichnet. Die Zusammenhänge, Beschreibungen und Variablendefinitionen stammen aus [Bet05, S. 11f]. Neben den dargestellten Linsentypen existiert noch eine Vielzahl rechtliniger Linsen mit Verzerrung, auf die hier nicht weiter eingegangen wird.

$$\mathbb{M}_{TObj,Tensor} = \begin{pmatrix} t_1 & t_2 & t_3 & \dots & t_N \\ v_{s,1} & -9999 & v_{s,3} & \dots & v_{s,N} \\ v_{e,1} & -9999 & v_{e,3} & \dots & v_{e,N} \\ \Delta t_1 & -9999 & \Delta t_3 & \dots & \Delta t_N \\ \Delta x_1 & -9999 & \Delta x_3 & \dots & \Delta x_N \\ y_{s,1} & -9999 & y_{s,3} & \dots & y_{s,N} \\ y_{e,1} & -9999 & y_{e,3} & \dots & y_{s,N} \\ ds_{x,1} & -9999 & ds_{x,3} & \dots & ds_{x,N} \\ ds_{y,1} & -9999 & ds_{y,3} & \dots & ds_{y,N} \\ l_{long,1} & -9999 & l_{long,3} & \dots & l_{long,N} \\ l_{lat,1} & -9999 & l_{lat,3} & \dots & l_{lat,N} \\ l_{state,long,1} & -9999 & l_{state,long,3} & \dots & l_{state,long,N} \\ l_{state,lat,1} & -9999 & l_{state,lat,3} & \dots & l_{state,lat,N} \end{pmatrix}$$

Abbildung 2.4: Beispielhafte Darstellung der Matrix eines Verkehrsobjektes. Zeitpunkt $t_2$stellt für dieses Verkehrsobjekte keinen Manöverbeginn-Zeitpunkt dar. Die entsprechende Zeile ist somit mit dem Wert -9999 befüllt. Darstellung aus [Kha19].

2.5.2 Geometrische Zusammenhänge

Seien x_{Real}, y_{Real} und z_{Real} die Koordinaten eines Punktes im Raum, welcher sich im Blickfeld einer Kamera befindet. Die Einheit der drei Größen x_{Real}, y_{Real} und z_{Real} sei Meter. Sei weiterhin der Koordinatenursprung im Zentrum der Linse dieser Kamera. Die y-Achse zeige hierbei aus der Sicht der Kamera nach links und die z-Achse zeige aus der Sicht der Kamera nach oben. Die x-Achse zeige in Blickrichtung der Kamera. Des Weiteren seien y_{Bild} und z_{Bild} die Koordinaten des selben Punktes im von der Kamera aufgenommenen Bild, wobei sich der Koordinatenursprung im Zentrum des Bildes befindet und die Richtungen der y- und z-Achse denen im Raum entsprechen. Da sich diese Betrachtung auf digitale Bilder bezieht, sei die Einheit der zwei Größen y_{Bild} und z_{Bild} Pixel.[3]

Des Weiteren sei R der euklidische Abstand des Punktes (x_{Real},y_{Real},z_{Real}) im Raum zur Blickachse der Kamera (die Einheit von R ist somit Meter) und φ der Winkel, der sowohl

[3]Bei dieser Betrachtungsweise ist insofern Vorsicht geboten, als die Auflösung einer Kamera meist eine einstellbare Größe darstellt: Wenn das Bild in doppelter Auflösung aufgenommen wird, dann liegen auf der gleichen „Strecke“ im Bild doppelt so viele Pixel. Dies ist der Grund, warum in der später in dieser Arbeit beschriebenen Implementierung die Auflösung der Eingangsbilder fest als $1920 * 1080$ Pixel definiert wird. Würden andere Auflösungen verwendet, müssten einige Aspekte der Implementierung entsprechend angepasst werden.

$$\mathbb{M}_{Ego,Tensor} = \begin{pmatrix} t_1 & t_2 & \dots & t_m \\ v_{s,1} & v_{s,2} & \dots & v_{s,m} \\ v_{e,1} & v_{e,2} & \dots & v_{e,m} \\ \Delta t_1 & \Delta t_2 & \dots & \Delta t_m \\ \Delta x_1 & \Delta x_2 & \dots & \Delta x_m \\ y_{s,1} & y_{s,2} & \dots & y_{s,m} \\ y_{e,1} & y_{e,2} & \dots & y_{e,m} \\ -9999 & -9999 & \dots & -9999 \\ -9999 & -9999 & \dots & -9999 \\ l_{long,1} & l_{long,2} & \dots & l_{long,m} \\ l_{lat,1} & l_{lat,2} & \dots & l_{lat,m} \\ -9999 & -9999 & \dots & -9999 \\ -9999 & -9999 & \dots & -9999 \end{pmatrix}$$

Abbildung 2.5: Darstellung der Matrix des Ego-Fahrzeugs. Die Zeilen, die mit den Variablen $\vec{d_{s_x}}$, $\vec{d_{s_y}}$, $\vec{L_{state,long}}$ und $\vec{L_{state,lat}}$ korrespondieren, enthalten den Wert -9999. Darstellung aus [Kha19].

durch die y-Achse und den Vektor $\overrightarrow{(y_{Real}, z_{Real})}$ als auch durch die y-Achse und den Vektor $\overrightarrow{(y_{Bild}, z_{Bild})}$ aufgespannt wird. Die Definitionen von f, θ und r aus 2.5.1 gelten weiterhin (die Einheit von r ist somit Pixel). Die beschriebene Geometrie ist in Abbildung 2.6 anhand eines Beispielpunktes dargestellt.

Mit den obigem Definitionen gilt $\cos(\varphi) = \frac{y_{Real}}{R} \iff \cos(\varphi) * R = y_{Real}$ und $\sin(\varphi) = \frac{z_{Real}}{R} \iff \sin(\varphi) * R = z_{Real}$ sowie $\tan(\theta) = \frac{R}{x_{Real}} \iff R = x_{Real} * \tan(\theta)$. Daraus ergeben sich die folgenden zwei Gleichungen, die im folgenden Abschnitt verwendet werden:

$$y_{Real} = \cos(\varphi) * R = \cos(\varphi) * x_{Real} * \tan(\theta) \tag{2.1}$$

$$z_{Real} = \sin(\varphi) * R = \sin(\varphi) * x_{Real} * \tan(\theta) \tag{2.2}$$

2.5.3 Implikation der Wahl einer konventionellen rechtlinigen Linse ohne Verzerrung

Angenommen, bei der betrachteten Linse handelt es sich um eine konventionelle rechtlinige Linse ohne Verzerrung mit Projektionsfunktion $r = f * \tan(\theta)$. Die inverse Funktion der

Linsentyp	Projektionsfunktion	Eigenschaft
Konventionelle rechtlinige Linse (engl.: rectilinear lens) ohne Verzerrung	$r = f * \tan(\Theta)$	Linearer Zusammenhang zwischen dem Abstand zur Blickachse im Raum und dem Abstand zur Blickachse im Bild (wird in 2.5.3 gezeigt).
Äquidistante Projektion (engl.: equidistant projection)	$r = f * \Theta$	Die Skalierung von Winkeldistanzen ist im ganzen Bild konstant.
Äqui-Raumwinkel-Projektion (engl.: equi-solid angle projection)	$r = 2 * f * \sin(\frac{\Theta}{2})$	Abgebildete Flächen sind lineare Skalierungen der dargestellten Flächen.
Orthographische Projektion (engl.: orthographic projection)	$f * \sin(\Theta)$	Die Helligkeit eines abgebildeten Objektes ist überall im Bild gleich.
Stereographische Projektion (engl.: stereographic projection)	$r = 2 * f * \tan(\frac{\Theta}{2})$	Winkel zwischen abgebildeten Kurven bleiben erhalten.

Tabelle 2.2: Einige Linsentypen, ihre Projektionsfunktionen und Eigenschaften. Tabelleninhalt aus [Bet05, S. 11f].

Projektionsfunktion ist für $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$ gegeben mit $\theta = \arctan(\frac{r}{f})$[4]. Des Weiteren gilt mit den Definitionen aus 2.5.2 $\cos(\varphi) = \frac{y_{Bild}}{r} \iff \cos(\varphi) * r = y_{Bild}$ und $\sin(\varphi) = \frac{z_{Bild}}{r} \iff \sin(\varphi) * r = z_{Bild}$. Schlussendlich gilt unter Verwendung dieser Gleichungen, der obigen inversen Funktion, sowie der Gleichungen 2.1 und 2.2:

$$y_{Real} = \cos(\varphi) * x_{Real} * \tan(\arctan(\frac{r}{f})) = \cos(\varphi) * r * x_{Real} * \frac{1}{f} = y_{Bild} * x_{Real} * \frac{1}{f}$$

$$\iff y_{Bild} = \frac{y_{Real}}{x_{Real}} * f \tag{2.3}$$

$$z_{Real} = \sin(\varphi) * x_{Real} * \tan(\arctan(\frac{r}{f})) = \sin(\varphi) * r * x_{Real} * \frac{1}{f} = z_{Bild} * x_{Real} * \frac{1}{f}$$

$$\iff z_{Bild} = \frac{z_{Real}}{x_{Real}} * f \tag{2.4}$$

Somit sind Formeln gegeben, um für beliebige, sich im Sichtfeld der Kamera befindliche Punkte, bei gegebenem Abstand in x-Richtung und gegebener Brennweite f, die y- beziehungsweise die z-Koordinate von ihrem Wert in der Bildebene in ihren realen Wert im Raum umzurechnen und umgekehrt. Diese Erkenntnis wird in 3.3.1.1 verwendet. Es lohnt sich außerdem festzustellen, dass bei der Wahl einer konventionellen rechtlinigen Linse ohne Verzerrung der mathematische Zusammenhang zwischen y_{Bild} und $\frac{y_{Real}}{x_{Real}}$ beziehungsweise zwischen z_{Bild} und $\frac{z_{Real}}{x_{Real}}$ linear ist.

[4] $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$ ist gegeben, da für die Position von betrachteten Objekten stets $x_{Real} > 0$ gilt. Punkte, die sich im Blickfeld der Kamera befinden, befinden sich also stets vor der Kamera. Anmerkung: Bei Fisheye-Linsen wäre dies nicht notwendigerweise der Fall.

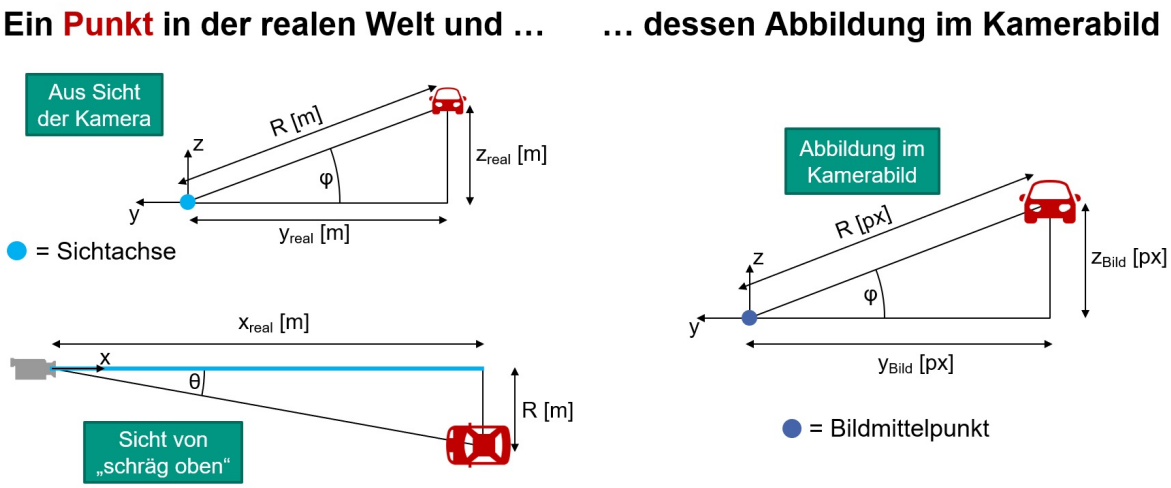

Abbildung 2.6: Die Position eines Beispielpunktes (zwecks Unterscheidbarkeit zwischen der Vorderansicht und der Ansicht von oben dargestellt als ein Automobil) in der realen Welt (links) und im Kamerabild (rechts), versehen mit den in den Abschnitten 2.5.1 und 2.5.2 eingeführten Größen. Eigene Darstellung.

2.6 Einführung der Begriffe Training, Objekterkennung, Detektor und Klassifikator

Es werden zunächst einige Begriffe für die Verwendung in dieser Arbeit festgelegt. Hierbei handelt es sich nicht um allgemeingültige Definitionen, sondern um vom Autor gewählte Begriffsfestlegungen für den Kontext dieser Arbeit.

Der Begriff „Training“ bezeichnet im Kontext des maschinellen Lernens meist das Suchen und im Idealfall auch das Finden von optimalen oder approximativ optimalen Programm- bzw. Modellparametern, mit dem Ziel, dass das resultierende Programm beziehungsweise das resultierende Modell einen Sachverhalt möglichst exakt abbildet oder ein gegebenes Problem möglichst zufriedenstellend löst. Dies ist gemeint, wenn in dieser Arbeit der Begriff Training verwendet wird.

Der Begriff „Objekterkennung“ (engl.: object recognition) beschreibt in unserem Kontext Verfahren zum Identifizieren bekannter Objekte in einem digitalen Bild. Objekterkennung findet im Rahmen dieser Arbeit in zwei Schritten statt:

1. Detektion: Eine Objekt-Detektion wird durch einen Programm durchgeführt, das mittels annotierter Trainingsdaten so trainiert wurde, dass es bei gegebenem Eingabebild mit einer gewissen Genauigkeit ausgibt, ob und falls ja wo sich in diesem Bild das Objekt, das von Interesse ist, befindet. Die Ausgaben sind hier die Positionen im Bild und die Abmessungen der Bildausschnitte, in denen sich das Objekt befindet, welches von Interesse ist. Die diese Bildausschnitte umschließenden Begrenzungsrahmen werden auch als Bounding-Boxen bezeichnet. Ein Programm, welches den hier

beschriebenen Zweck erfüllt, wird im Folgenden als Detektor oder Objekt-Detektor bezeichnet.

2. Klassifikation: Die Objekt-Klassifikation wird durch ein Programm durchgeführt, das mittels annotierter Trainingsdaten so trainiert wurde, dass es einen gegebenen Eingangsdatenpunkt einer Kategorie aus einer vorher definierten, endlichen Menge an Kategorien zuordnet. In unserem Kontext stellen die Ausgaben eines Verkehrsschild-Detektors - also die Bildausschnitte, in denen sich detektierte Verkehrsschilder befinden - die zu klassifizierenden Datenpunkte dar. Die Ausgabe ist eine natürliche Zahl, die mit der dem Eingangsdatenpunkt zugeordneten Klasse korrespondiert. Ein auf diese Weise klassifizierendes Programm wird im Folgenden als Klassifikator bezeichnet.

2.7 Aggregate Channel Features

In dieser Arbeit wird sowohl für die Detektion von Fahrzeugen als auch für die Detektion von Verkehrsschildern ein Detektor verwendet, der auf einem Konzept basiert, das in [DABP14] durch Dollár et al. zuerst vorgestellt wurde: Den Aggregate Channel Features (ACF - auf Deutsch in etwa: Aggregierte Kanal-Merkmale). In den folgenden Abschnitten werden zunächst einige Konzepte vorgestellt, die für das Verständnis des Aggregate Channel Features-Detektors relevant sind. Daraufhin wird die grobe Funktionsweise eines Aggregate Channel Features-Detektors erklärt.

2.7.1 Relevante Begriffe für Aggregate Channel Features

Channel (engl., „Kanal“):

Für ein Eingabebild I ist ein korrespondierender Channel $C = \Omega(I)$ eine Abbildung, welche definierte Anordnungen (man könnte auch sagen „Flecken“) von Pixeln in I auf je einen korrespondierenden Ausgabepixel abbildet, wobei die allgemeine Anordnung von I erhalten bleibt [DTPB09, S. 4]. Einfache Beispiele für Channels sind die drei Farbkanäle eines digitalen Kamerabildes: Rot, Grün und Blau. Um diese zu erhalten, wird je Pixel des Originalbildes nur der entsprechende der drei Farbwerte übernommen und im Ausgabepixel gespeichert. Allgemein stellen Channels lineare oder nichtlineare Transformationen des Input-Bildes dar.

Feature:

Für praktische Anwendungen im Kontext des maschinellen Lernens und der Mustererkennung werden die Original-Eingangsvariablen eines Problems (wie zum Beispiel die Pixel eines digitalen Bildes) typischerweise vorverarbeitet und somit in eine neue, meist kleinere Menge an Variablen umgewandelt. Dies geschieht in der Hoffnung, dass das Problem so einfacher zu lösen ist. [Bis06, Kapitel 1; Seite 2f] Sowohl diese neuen Variablen als auch die Original-Eingangvariablen werden als Features (engl., „Merkmale“) bezeichnet. Andere Begriffe hierfür sind Attribut, Eigenschaft und Charakteristik[LM00, Unterkapitel 1.1.1;

Seite 3]. Beispielsweise kann die gemessene Körpergröße von betrachteten Personen ein Feature darstellen, aber auch Linearkombinationen mehrerer Features können wiederum als solche aufgefasst werden. Ein weiteres Beispiel für ein Feature, welches für ein digitales Bild berechnet werden kann, ist ein Histogramm, welches die Häufigkeitsverteilung der Lichtintensitäten im Bild beschreibt.

Feature Pyramid (engl., auf Deutsch etwa „Merkmalspyramide“):

Laut Dollár et al. [DABP14] zeigen empirische Studien im Feld der Computer Vision (engl., „Maschinelles Sehen“), dass für die Extrahierung von visueller Information aus Bilddaten übervollständige Repräsentationen (engl.: overcomplete representations) dieser Daten vorteilhaft sind. Übervollständigkeit bedeutet in diesem Fall, dass Daten nicht nur in ihrer ursprünglichen Form, sondern zusätzlich auch mit Transformationen wie zum Beispiel verschiedenen Skalierungen und Orientierungen versehen, verwendet werden. Eine Vereinigung von Mengen von Features, die aus verschiedenen Skalierungen eines Bildes berechnet wurden, nennt man Feature Pyramid[LDG+16].

AdaBoost:

AdaBoost ist eine spezielle Form, einen Boosted Classifier (engl., auf Deutsch etwa „verstärkter Klassifikator“) zu trainieren. Ein Boosted Classifier ist ein Klassifikator der Form: $F_T(x) = \sum_{t=1}^{T} f_t(x)$, wobei f_t jeweils schwache Klassifikatoren (engl.: weak learner) bezeichnet, die einen Datenpunkt x als Input akzeptieren und einen Wert zurückgeben, der die Prädiktion der Klasse von x durch f_t darstellt.

Im Folgenden wird die grobe Funktionsweise des AdaBoost-Trainingsprozesses beschrieben (angelehnt an [FS97, Unterkapitel 4.1.; S. 125f] und [Bis06, Unterkapitel 14.3.; S. 657ff]):

- Gegeben: N annotierte Datenpunkte ($i = 1, ..., N$), Anzahl an Iterationen: T ($t = 1, ..., T$)
- Gewichtskoeffizienten initialisieren: Jedem Datenpunkt wird ein Gewichtskoeffizient $w_i^{(t)}$ zugeordnet. Alle Gewichtskoeffizienten werden mit $w_i^{(1)} = \frac{1}{N}$ initialisiert.
- In jeder Iteration t:
  - Ein schwacher Klassifikator wird aufgerufen. Dieser wird unter Berücksichtigung der Gewichtung der Datenpunkte auf das Trainingsset angepasst und trifft für jeden Punkt im Trainingsset eine Hypothese $h_t(x_i)$.
  - Ein Koeffizient β_t wird so berechnet, dass folgende Summe minimal ist, wobei Err irgendeine Fehlerfunktion und F_{t-1} den Boosted Classifier, der bis zur vorherigen Stufen aufgebaut wurde, darstellt:

$$Err_t = \sum_i Err[F_{t-1}(x_i) + \beta_t * h_t(x_i)] \tag{2.5}$$

  - Die Gewichtskoeffizienten werden so aktualisiert, dass das Gewicht von Datenpunkt i in der nächsten Iteration hoch ist, wenn der Fehler $Err(F_{t-1}(x_i))$ hoch war und niedrig, wenn $Err(F_{t-1}(x_i))$ niedrig war.

- Nach T Iterationen ist der resultierende durch AdaBoost trainierte Klassifikator gegeben durch $F_T(x) = \sum_{t=1}^{T} \beta_t * h_t(x)$.

AdaBoost ist der Trainingsprozess, der verwendet wurde, um die zwei im Rahmen dieser Arbeit verwendeten Aggregate Channel Features-Detektoren zu trainieren.

2.7.2 Funktionsweise des Aggregate Channel Features-Detektors

Es folgt eine Beschreibung der Funktionsweise von Aggregate Channel Features, die an die Beschreibung in [DABP14] angelehnt ist:

Gegeben sei ein Input-Bild I. Von diesem wird eine bestimmte Anzahl von Skalierungen erstellt[5]. Für jede dieser skalierten Bilder I_i werden dann die folgenden Schritte ausgeführt[6]:

- Glätten von I_i mithilfe eines [1 2 1]/4-Filters. Das bedeutet, dass die Werte aller Pixel in I_i auf folgende Weise transformiert werden, wobei p den Wert des betrachteten Pixels nach Anwendung des Filters und p_{orig} den Wert des betrachteten Pixels vor Anwendung des Filters bezeichne und die zwei benachbarten Pixel des betrachteten Pixels mit p_{links}und p_{rechts} bezeichnet seien:

$$p = \frac{1 * p_{links} + 2 * p_{orig} + 1 * p_{rechts}}{4} \tag{2.6}$$

- Aus dem geglätteten Bild werden 10 Channels $C_{ij} = \Omega_j(I_i)$ berechnet[7], es gilt also $j = 1, ..., 10$.

- Nun werden alle Pixel aus den zehn Channels C_{ij} blockweise (jeweils 4∗4 Pixel) summiert, sodass sich als Resultat 10 kleinere Channels c_{ij} - die Aggregated Channels (engl., „aggregierte Channels“) - ergeben. Folgen wir dem Beispiel aus [DABP14, S. 9] und nehmen an, dass eines der skalierten Bilder I_i und somit auch die aus ihm resultierenden zehn Channels C_{ij} jeweils die Größe $128 * 64$ Pixel haben: Dann besitzen die zehn resultierenden aggregierten Channels c_{ij} jeweils eine Größe von $32 * 16$ Pixeln.

[5] In der vorliegenden Implementierung werden acht Skalierungsschritte pro Oktave erstellt. Eine Oktave bezeichnet das Intervall zwischen zwei Skalierungen, wobei die zweite genau halb oder doppelt so groß ist wie die erste [DABP14, S.8].

[6] Eigentlich werden pro Oktave nur einige Skalierungen wirklich erstellt. Mit den Ergebnissen der tatsächlich durchgerechneten Skalierungsstufen werden die Ergebnisse der fehlenden Stufen approximiert um Zeit zu sparen (deshalb auch „Fast Feature Pyramids“ [DABP14]). Dies wird im Rahmen dieser groben Erklärung jedoch nicht weiter beleuchtet.

[7] In der vorliegenden Implementierung werden die folgenden Channels berechnet: Normalisierte Gradientengröße (ein Channel), Histogramm über die Gradientenausrichtung (sechs Channels) und „LUV color channels“ (drei Channels) [DABP14, S.9].

- Die aggregierten Channels c_{ij} werden auch mit einem [1 2 1]/4-Filter geglättet.
- Dann werden die Pixel der zehn aggregierten Channels zu einem einzigen Vektor umstrukturiert. Dieser besitzt mit dem obigen Beispiel $32 * 16 * 10 = 5120$ Einträge, wobei jedes der 5120 Vektorelemente nun ein Feature (Merkmal) darstellt.

Mithilfe der so berechneten Merkmale werden nun durch AdaBoost 2048 Entscheidungsbäume (engl: decision trees) der Tiefe zwei trainiert und kombiniert. Tiefe zwei bedeutet, dass die längste vorhandene Strecke im Entscheidungsbaum von der Wurzel zu einem Blatt aus genau zwei Kanten besteht.

2.8 Künstliche neuronale Netze (KNN) und Convolutional Neural Networks (CNN)

Der Begriff „Neuronales Netz“ hat seinen Ursprung in dem Versuch, mathematische Repräsentationen für Informationsverarbeitung in biologischen Systemen zu finden. Meist liegt der Fokus bei künstlichen neuronalen Netzen (KNN) jedoch nicht auf biologischer Wirklichkeitstreue, welche unnötige Einschränkungen mit sich bringen würde. Stattdessen handelt es sich in erster Linie um effiziente Modelle zur statistischen Mustererkennung [Bis06, S. 226]. Im folgenden Abschnitt werden einige Kernkonzepte, die zum grundlegenden Verständnis von künstlichen neuronalen Netzen hilfreich sind, vorgestellt. Schließlich wird eine spezielle Form der KNN vorgestellt: Das Convolutional Neural Network (CNN), welches in dieser Arbeit für die Klassifizierung von Verkehrsschildern Verwendung findet.

2.8.1 Grundlegende Funktionsweise eines KNN

Die folgende Beschreibung ist angelehnt an [Bis06, Kapitel 5.1; S. 227 ff]: Künstliche neuronale Netze können als gerichtete Graphen aufgefasst werden, deren Knoten Eingabevariablen, Ausgabevariablen und die sogenannten verborgenen Variablen darstellen. Die Kanten stellen Gewichte dar, die definieren, inwieweit Information von vor- in nachgeschaltete Knoten übertragen wird. Die Knoten, die mit den Eingabevariablen korrespondieren, haben keine vorgeschalteten Knoten. Sie stellen vertikal angeordnet die erste Schicht eines künstlichen neuronalen Netzes dar. Jeder der Knoten der ersten Schicht gibt seine Information an jeden Knoten der nächsten Schicht weiter. Diese nächste Schicht ist die erste verborgene Schicht (engl.: hidden layer). Jeder Knoten der ersten verborgenen Schicht erhält als Eingaben Informationen von allen Knoten der vorangegangenen Schicht und kombiniert diese Werte auf lineare Weise gemäß der Gewichte, die durch die ankommenden Kanten dargestellt sind. Das Ergebnis dieser Linearkombination wird anschließend mit einem Fehlerterm (engl.: bias) addiert. Das Ergebnis dieser Berechnung wird als Aktivierung (engl.: activation) bezeichnet. Die Aktivierung eines Knotens mit Index j der ersten verborgenen Schicht ist somit durch die folgende Gleichung gegeben, wobei D die

Anzahl der Knoten in der mit den Eingabevariablen korrespondierenden Schicht ist:

$$a_j^{(1)}(\overrightarrow{x}) = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j_0}^{(1)} \tag{2.7}$$

Die Aktivierung jedes Knotens wird im Anschluss noch durch eine nichtlineare, differenzierbare Aktivierungsfunktion $h(\cdot)$ transformiert, sodass das Endergebnis für die Knoten dieser Schicht mit

$$z_j^{(1)} = h(a_j^{(1)}) \tag{2.8}$$

gegeben ist. Es wird nicht notwendigerweise dieselbe Aktivierungsfunktion für alle Schichten des KNN verwendet.

Nach der ersten verborgenen Schicht folgen entweder weitere verborgene Schichten, deren Knoten als Eingangsinformationen $\overrightarrow{x}$ jeweils den Vektor der Endergebnisse $\overrightarrow{z}_j^{(l-1)}$ der vorangehenden Schicht verwenden und ihre Endergebnisse auf die gleiche oben beschriebene Weise berechnen, oder die Ausgabeschicht, deren Knoten mit den Ausgabevariablen korrespondieren. Die Knoten der Ausgabeschicht ähneln den Knoten von verborgenen Schichten mit dem Unterschied, dass sie ihre Endergebnisse nicht an eine weitere Schicht weitergeben. Falls es sich um ein Mehrklassen-Klassifizierungsproblem (engl.: multiclass classification problem) handelt, werden die Aktivierungen der Ausgabeschicht mit einer Softmax-Aktivierungsfunktion ([Bis06, S. 198]) transformiert, was zur Folge hat, dass die Summe der Aktivierungen der Ausgabeschicht gleich eins ist. Die resultierenden Aktivierungen der Knoten der Ausgabeschicht repräsentieren die durch das KNN berechnete Vorhersage für einen gegebenen Eingabevektor.

Ein KNN mit einer verborgenen Schicht ist in Abbildung 2.7 schematisch dargestellt. Zu beachten ist, dass die Fehlerterme, die mit dem Ergebnis der Linearkombination der Werte der vorangegangenen Schicht addiert werden (vgl. 2.7), hier als die Gewichte von zusätzlichen Knoten $x_0$und z_0 dargestellt werden. Diese zusätzlichen Knoten werden dauerhaft mit dem Wert 1 belegt, wodurch ihr Einfluss genau den von ihnen ausgehenden Kantengewichtungen entspricht.

Auf das Training eines KNN wird in dieser Arbeit nicht im Detail eingegangen. Zum grundlegenden Verständnis ist es jedoch hilfreich, zu erwähnen, dass der Trainingsprozess meist aus dem Finden von einer für einen gegebenen Trainingsdatensatz optimalen oder approximativ optimalen Belegungen für die Gewichte $w_{ji}^{(l)}$ besteht. Dies geschieht durch das Lösen beziehungsweise durch das Approximieren der Lösung eines globalen Optimierungsproblems. Da es sich hierbei im Allgemeinen nicht um ein konvexes Optimierungsproblem handelt, kann nicht einfach angenommen werden, dass es sich bei einem gefundenen oder approximierten lokalen Minimum auch um das globale Minimum handelt. Eine Methode, die zur Lösung der beschriebenen Problemstellung verwendet wird, trägt den Namen Stochastic Gradient Descent (auf Deutsch etwa „stochastischer Gradientenabtieg“) [Bis06, Kapitel 5.2.4]. Für nähere Informationen zum Training von künstlichen neuronalen Netzen sei auf [Bis06, Unterkapitel 5.2] verwiesen.

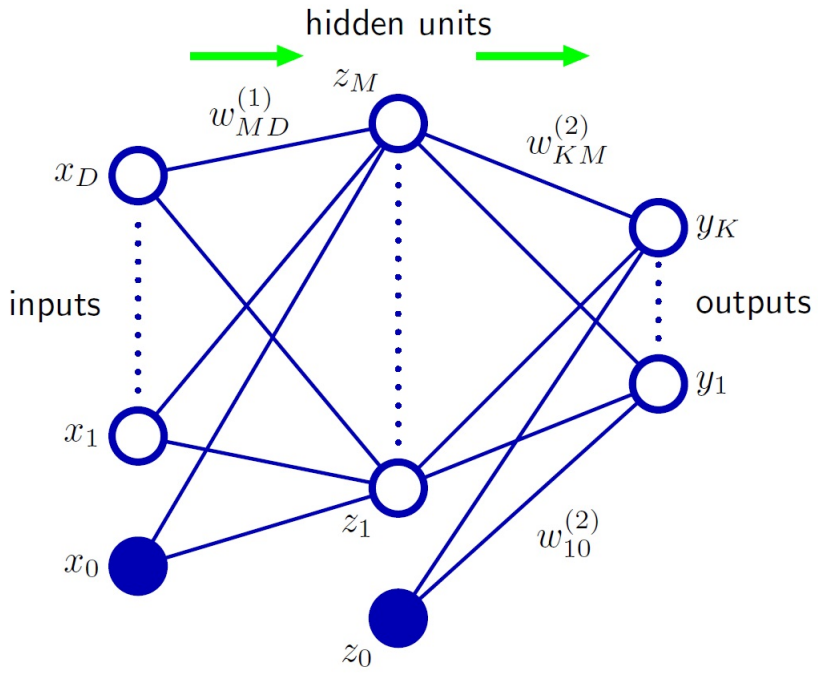

Abbildung 2.7: Schematische Darstellung eines künstlichen neuronalen Netzes mit einer verborgenen Schicht. Abbildung aus [Bis06, S. 228].

2.8.2 Convolutional Neural Networks

Ein Convolutional Neural Network (CNN; auf Deutsch etwa „faltendes neuronales Netz“) ist ein künstliches neuronales Netz, dem einige Verarbeitungsschritte vorgeschaltet sind. Namentlich handelt es sich bei diesen Schritten um die Convolution (auf Deutsch etwa „Faltung“) und das Subsampling (auf Deutsch etwa „Herabskalierung“). Die folgenden grundlegenden Beschreibungen von CNNs sind an die Beschreibungen in [Bis06, Kapitel 5.5.6; S. 267 ff] angelehnt.

2.8.3 Convolution

Eine Aufgabe, für die CNNs oft verwendet werden, ist die Klassifizierung von digitalen Bildern. CNNs sind für diesen Zweck besonders geeignet, da sie eine wichtige Eigenschaft von digitalen Bildern berücksichtigen: Dass die Werte von Pixeln, die nah beieinanderliegen im Allgemeinen stärker korreliert sind als die Werte von Pixeln, die weiter entfernt voneinander liegen. Dies wird berücksichtigt, indem Merkmale beziehungsweise die Position derselben durch den Schritt der Convolution extrahiert werden. Dies geschieht durch Anwendung von Filtern auf das Eingabebild, wodurch sogenannte Feature Maps (auf Deutsch etwa „Merkmalskarten“) erstellt werden. Diese bilden im Idealfall ab, wo im betrachteten Bild sich ein bestimmtes durch den entsprechenden Filter gesuchtes Merkmal befindet.

2.8.4 Subsampling

Im Subsampling-Schritt werden die im Convolution-Schritt erstellten Feature Maps auf kleinere Repräsentationen derselben verkleinert, indem beispielsweise jeweils Flecken der Größe zwei mal zwei Pixel auf einen Pixel in der korrespondierenden herabskalierten Repräsentation abgebildet werden. Dies geschieht beispielsweise, indem aus den Werten jedes dieser Flecken das arithmetische Mittel oder der Maximalwert berechnet wird. Falls das arithmetische Mittel verwendet wird, spricht man von „Mean Pooling“ (engl., auf Deutsch etwa „Durchschnitts-Zusammenfassung“). Falls der Maximalwert verwendet wird spricht man von „Max Pooling“ (engl., auf Deutsch etwa „Maximums-Zusammenfassung“). Auf diese Weise wird die Menge an weiterzuverarbeitender Information erheblich reduziert, während die ermittelten Positionen der extrahierten Merkmale ohne nennenswerten Informationsverlust weiter bestehen.

2.8.5 Zusammenführung

Oft enthalten CNNs mehrere Paare von Convolution- und Subsamplingschritten. Im Anschluss an diese enthält ein CNN eine neuronale Netzstruktur, wie sie in 2.8.1 beschrieben wird. Die Eingabegrößen dieser Struktur sind alle resultierenden Werte der letzten verwendeten Subsampling-Schicht.

2.9 Transfer Learning

Der folgende Abschnitt stellt keine allgemeine Definition des Begriffs „Transfer Learning“ dar, sondern führt nur seine für diese Arbeit relevante Bedeutung ein.

Das in dieser Arbeit verwendete CNN ist eine angepasste Version eines CNNs, welches mithilfe eines Datensatzes trainiert wurde, der digitale Trainingsbilder von US-amerikanischen Verkehrsschildern enthält. Die so gefundenen Gewichte wurden als Initialgewichte für das für diese Arbeit durchgeführte weitere Training mit Bildern von deutschen Verkehrsschildern verwendet. Ein solches Weiterverwenden bereits trainierter Modelle wird auch als Transfer Learning (auf deutsch etwa „übertragendes Lernen“) bezeichnet.

Kapitel 3

Konzept

3.1 Anforderungen

Die Anforderungen an die im Rahmen dieser Arbeit implementierten Funktionen lauten wie folgt:

1. Eine möglichst exakte Bestimmung von
   (a) der realen Höhe und Breite der sich im Sichtbereich des Ego-Fahrzeugs befindlichen Verkehrsobjekte und
   (b) der Position und Identifikation von sich an der Fahrstrecke befindlichen Verkehrsschildern.
2. Die Erweiterung des in der vorangegangenen Masterarbeit entwickelten Modells (siehe 2.4) auf geeignete Weise, um
   (a) die durch 1. gewonnenen Informationen in diesem Modell möglichst in abstrahierter Form zu speichern und
   (b) die Möglichkeit zu schaffen, allgemeine statische Informationen in diesem Modell ebenfalls zu speichern.
3. Die Daten, die zur Bestimmung der Größen verwendet werden, sollen aus den Sensoren des Ego-Fahrzeugs stammen. Für diese Arbeit werden simulierte Daten verwendet. Auf diese werden beispielhaft die entwickelten Funktionen angewandt.

3.2 Annahmen

Die im Bezug auf die konzeptionelle Entwicklung und Implementierung getroffenen Annahmen lauten wie folgt:

1. Zu jedem Zeitpunkt kann die aktuelle Position des Ego-Fahrzeugs bestimmt werden. Da viele moderne Fahrzeuge über GPS-Systeme verfügen, ist diese Annahme gerechtfertigt.

2. Das Ego-Fahrzeug besitzt Sensoren, mit dessen Hilfe der relative Abstand (p) und die ungefähre Richtung (α, Θ) von den sich im Sichtfeld des Systems befindlichen Verkehrsobjekten zu jedem Zeitpunkt bestimmt werden können. In 2.1.3 werden die gängigsten Fahrzeugsensoren vorgestellt. Eine solche Positionsbestimmung könnte mithilfe mehrerer der vorgestellten Technologien durchgeführt werden. Für diese Arbeit ist es lediglich relevant, festzuhalten, dass es Technologien gibt, die dieses Ziel erreichen können und dass diese Annahme insofern gerechtfertigt ist. Aus gegebenen räumlichen Polarkoordinaten (p, α, Θ) des betrachteten Verkehrsobjekts können die kartesischen Koordinaten (x, y, z) des betrachteten Verkehrsobjekts eindeutig bestimmt werden. Hieraus folgt, dass zu jedem Zeitpunkt zumindest die ungefähren relativen Positionen im Raum der sich im Sichtfeld des Detektionssystems befindlichen Verkehrsobjekte bestimmt werden können.

3. Das Ego-Fahrzeug besitzt zwei Kameras. Eine, die nach vorne gerichtet ist und eine, die nach hinten gerichtet ist.

3.3 Beschreibung

3.3.1 Verkehrsobjekt-Größenerkennung

3.3.1.1 Modell

Im Folgenden werden die geometrischen Zusammenhänge, die in 2.5.2 vorgestellt wurden aufgegriffen und mithilfe von diesen die Beschaffenheit des für die Verkehrsobjekt-Größenerkennung verwendeten mathematischen Modells hergeleitet.

Es sei alles so definiert wie in 2.5.2, das heißt es seien x_{Real}, y_{Real} und z_{Real} die Koordinaten eines Punktes im Raum, wobei der Koordinatenursprung im Zentrum der Linse einer Kamera ist, die ein Bild aufnimmt, das den besagten Punkt enthält. Die y-Achse zeigt hierbei aus der Sicht der Kamera nach links und die z-Achse zeigt aus der Sicht der Kamera nach oben. Die x-Achse zeigt in Blickrichtung der Kamera. Des Weiteren seien y_{Bild} und z_{Bild} die Koordinaten desselben Punktes im von der Kamera aufgenommenen Bild, wobei sich der Koordinatenursprung im Zentrum des Bildes befindet und die Richtungen der y- und z-Achse denen im Raum entsprechen.

Zu Erinnerung: Für eine Kamera mit konventioneller rechtliniger Linse (engl.: rectilinear lens) ohne Verzerrung gelten nach 2.3 und 2.4 folgende Zusammenhänge:

$$y_{Bild} = \frac{y_{Real}}{x_{Real}} * f \tag{3.1}$$

$$z_{Bild} = \frac{z_{Real}}{x_{Real}} * f \tag{3.2}$$

wobei mit f die Brennweite der Linse der Kamera bezeichnet sei.

Somit kann, wie in 2.5.3 festgestellt wurde, bei bekanntem Abstand x_{Real} sowie bekannter Brennweite f mithilfe der Gleichungen 3.1 und 3.2 aus der Position eines Punktes im Kamerabild die Position desselben Punktes im Raum exakt bestimmt werden und umgekehrt.

Seien nun die im Bild der Kamera gemessene Breite beziehungsweise Höhe eines Objektes gegeben durch $Breite_{Bild} = y_{2,Bild} - y_{1,Bild}$ beziehungsweise $Höhe_{Bild} = z_{2,Bild} - z_{1,Bild}$ und die reale Breite beziehungsweise Höhe des Objektes gegeben durch $Breite_{Real} = y_{2,Real} - y_{1,Real}$ beziehungsweise $Höhe_{Real} = z_{2,Real} - z_{1,Real}$, wobei die Punkte $(x, y, z)_{2,Real}$ und $(x, y, z)_{1,Real}$ identische Werte für die x-Komponente besitzen, also in x-Richtung gleich weit von der Kameralinse entfernt liegen. Dann gilt wegen 3.1 beziehungsweise 3.2 Folgendes:

$$Breite_{Bild} = y_{2,Bild} - y_{1,Bild} = \frac{y_{2,Real}}{x_{Real}} * f - \frac{y_{1,Real}}{x_{Real}} * f = \frac{y_{2,Real} - y_{1,Real}}{x_{Real}} * f = \frac{Breite_{Real}}{x_{Real}} * f$$

$$\iff Breite_{Bild} = \frac{Breite_{Real}}{x_{Real}} * f \tag{3.3}$$

$$Höhe_{Bild} = z_{2,Bild} - z_{1,Bild} = \frac{z_{2,Real}}{x_{Real}} * f - \frac{z_{1,Real}}{x_{Real}} * f = \frac{z_{2,Real} - z_{1,Real}}{x_{Real}} * f = \frac{Höhe_{Real}}{x_{Real}} * f$$

$$\iff Höhe_{Bild} = \frac{Höhe_{Real}}{x_{Real}} * f \tag{3.4}$$

Mit 3.3 und 3.4 kann also für ein sich im Sichtfeld der Kamera befindliches Objekt, dessen abzumessende Oberfläche orthogonal zur Sichtachse steht, bei bekanntem Abstand x_{Real} sowie bekannter Brennweite f aus der Breite beziehungsweise Höhe im Kamerabild die reale Breite beziehungsweise Höhe exakt bestimmt werden und umgekehrt.

Da sich diese Arbeit darauf beschränkt die Fahrt auf autobahnähnlichen Streckenabschnitten zu betrachten (siehe letzter Absatz 1.2), kann die oben formulierte Bedingung, dass sich die abzumessende Oberfläche in orthogonaler Ausrichtung zur Sichtachse befindet zumindest approximativ als gegeben angenommen werden, da die Fahrtrichtungen der betrachteten Fahrzeuge nur geringe Abweichungen von den Blickrichtungen der Kameras[1] des Ego-Fahrzeugs aufweisen.

Bei handelsüblichen Kameras ist die Brennweite beziehungsweise sind die einstellbaren Brennweiten der Linse normalerweise in den Produktspezifikationen angegeben oder in irgendeiner Form auslesbar. In dieser Arbeit wurde auf Bilddaten aus der Simulationssoftware CarMaker zurückgegriffen. In CarMakers IPGMovie wird durch die Simulationssoftware kein Wert für die Brennweite angegeben. Da mit 3.3 und 3.4 bei gegebenen Werten

[1] Hiermit sind die Blickrichtungen „nach vorne“ und „nach hinten“ gemeint.

für $Breite_{Bild}$,$Breite_{Real}$,$Höhe_{Bild}$, $Höhe_{Real}$ und x_{Real} jedoch ein lineares Gleichungssystem gegeben ist, welches eindeutig bestimmt ist, kann mithilfe eines Trainingsdatensatzes, der eine angemessene Anzahl von Datenpunkten bestehend aus den genannten Größen enthält, die Brennweite f approximiert werden. Zu diesem Zweck wird f so bestimmt, dass das resultierende Modell eine möglichst geringe Abweichung zu den Trainingsdaten aufweist. Die für die vorliegende Arbeit gewählte Metrik um diese Abweichung zu quantifizieren ist die Summe der quadratischen Abweichungen der durch das Modell berechneten Breiten und Höhen zu den wahren Breiten und Höhen. Es folgt eine Darstellung zweier Minimierungsprobleme, mithilfe derer ein beziehungsweise zwei geeignete Modellparameter bestimmt wurden.

Das erste Optimierungsproblem $P1$ unterscheidet nicht zwischen der Transformation, die für die Breitenberechnung stattfindet, und der Transformation, die bei der Höhenberechnung stattfindet. n ist die Anzahl an Beobachtungen im Trainingsdatensatz, wobei jede Beobachtung i ein Breiten-Paar ($Breite_{i,Bild}$, $Breite_{i,Real}$), ein Höhen-Paar ($Höhe_{i,Bild}$, $Höhe_{i,Real}$) und eine Distanz x_{Real} liefert. Das Optimierungsproblem $P1$ ist wie folgt definiert:

$$P1 : \min_{f} \sum_{i=1}^{n} ((Breite_{i,Bild} - \frac{Breite_{i,Real}}{x_{i,Real}} * f)^2 + (Höhe_{i,Bild} - \frac{Höhe_{i,Real}}{x_{i,Real}} * f)^2) \quad (3.5)$$

Durch das Lösen von $P1$ erhält man einen Wert für f, der für die gegebenen Daten ein im Sinne der Summe der quadratischen Abweichung optimales Modell definiert. Wie in 5.2 erklärt wird., ist dieser Ansatz der Ansatz, den der Autor für die weitere Verwendung empfiehlt.

Das zweite Optimierungsproblem $P2$ unterscheidet zwischen der Betrachtung der Breite und der Betrachtung der Höhe in dem Sinne, dass für Breite und Höhe zwei separate Modellparameter f_{Breite}und $f_{Höhe}$bestimmt werden. Dies entspricht nicht dem wahren, in 2.5 und in diesem Abschnitt weiter oben beschriebenen mathematischen Zusammenhang. Der Autor wollte durch Verfolgung diesen Ansatzes jedoch herausfinden, ob er zu besseren Ergebnissen als $P1$ (3.5) führt. Die zugrundeliegende Überlegung ist hier, dass es zwischen der Breiten- und Höhenmessung unterschiedlich ausgeprägte Messungenauigkeiten geben könnte und somit das auf diese Weise stattfindende Versehen des Modells mit einem weiteren Freiheitsgrad zu einer besseren Modellgüte führen könnte[2]. Dass dies nur bedingt der Fall ist, wird in 5.2 diskutiert. Das Optimierungsmodell $P2$ sei wie folgt definiert, wobei wie oben n die Anzahl an Beobachtungen im Trainingsdatensatz ist und jede Beobachtung i ein Breiten-Paar ($Breite_{i,Bild}$, $Breite_{i,Real}$), ein Höhen-Paar ($Höhe_{i,Bild}$, $Höhe_{i,Real}$) und eine Distanz x_{Real} liefert:

$$P2 : \min_{f_{Breite}, f_{Höhe}} \sum_{i=1}^{n} ((Breite_{i,Bild} - \frac{Breite_{i,Real}}{x_{i,Real}} * f_{Breite})^2 + (Höhe_{i,Bild} - \frac{Höhe_{i,Real}}{x_{i,Real}} * f_{Höhe})^2) \quad (3.6)$$

[2]Man beachte, dass das weiter oben genannte lineare Gleichungssystem auch mit zwei Modellparametern noch eindeutig bestimmt ist, da zwei Gleichungen (3.3 und 3.4) zur Verfügung stehen.

Im Falle, dass $P2$ Verwendung findet, werden die Gleichungen 3.1 und 3.2 entsprechend angepasst:

$$y_{Bild} = \frac{y_{Real}}{x_{Real}} * f_{Breite} \tag{3.7}$$

$$z_{Bild} = \frac{z_{Real}}{x_{Real}} * f_{Höhe} \tag{3.8}$$

3.3.1.2 Ablauf

Im Folgenden wird der Ablauf des implementierten Programms zur Verkehrsobjekt-Größenerkennung konzeptionell beschrieben. Die Daten, die verwendet werden, sind die Position des Ego-Fahrzeugs und die relativen Positionen der sich im Sichtfeld des Ego-Fahrzeugs befindlichen Verkehrsobjekte während des gesamten betrachteten Zeitraums sowie von Vorder- und Rückkamera aufgenommene, den ganzen Zeitraum abdeckende Bilderserien.

Im ersten Schritt wird pro Ein- und Austritts-Ereignis eines Verkehrsobjektes in den Sichtbereich des Ego-Fahrzeugs ein separates sogenanntes Detektionsobjekt definiert.

Als nächstes wird für jedes Bild detektiert, ob und wo sich in diesem ein oder mehrere Verkehrsobjekte befinden. Deren abzumessende Vorder- beziehungsweise Hinterseiten im Bild werden somit mit Bounding-Boxen versehen. Zur Verkehrsobjekt-Detektion wird der in 2.7 vorgestellte Aggregate Channel Features-Detektor verwendet.

Im folgenden Schritt werden den Bounding-Boxen die entsprechenden Detektionsobjekte zugeordnet: Dies wird erreicht, indem berechnet wird, wo im Bild sich welches Detektionsobjekt befinden müsste. In diesem Schritt werden einige Bounding-Boxen als Messfehler aussortiert; dazu Weiteres in Abschnitt 4.3.

Mithilfe der Abmessung der verbleibenden Bounding-Boxen (in Pixeln), den zugehörigen Abständen in Blickrichtung (x-Komponente der relativen Position) zu den jeweiligen Verkehrsobjekten (in Metern) und der Brennweite f der Kamera beziehungsweise f_{Breite} und $f_{Höhe}$ (f, f_{Breite} und $f_{Höhe}$ besitzen die Einheit Pixel und sind bestimmt durch Approximierung der Lösungen der Optimierungsprobleme $P1$ und $P2$) werden die realen Höhen und die reale Breiten der betrachteten Verkersobjekte geschätzt.

Aus den über die gesamte Fahrstrecke gesammelten Schätzwerten für Höhe und Breite werden jeweils die arithmetischen Mittel für die einzelnen Detektionsobjekte berechnet. Diese Durchschnitts-Schätzwerte für Höhe und Breite und die zugehörigen Identifikationen der entsprechenden Detektionsobjekte stellen das Ergebnis der Verkehrsobjekt-Größenerkennung dar.

3.3.2 Verkehrsschilderkennung: Ablauf

Im Folgenden wird der Ablauf des implementierten Programms zur Verkehrsschilderkennung konzeptionell beschrieben. Die Daten, die verwendet werden, sind die Position des

Ego-Fahrzeugs während des gesamten betrachteten Zeitraums sowie eine von der Vorderkamera aufgenommene, den ganzen Zeitraum abdeckende Bilderserie.

Im ersten Schritt wird für jedes Bild bestimmt, ob und wo sich in diesem Verkehrsschilder befinden. Dieser Schritt wird in dieser Arbeit als Verkehrsschild-Detektion bezeichnet. Das Resultat dieses Schrittes ist eine Matrix, dessen Inhalt die Bounding-Boxen beschreibt, welche die detektierten Verkehrsschilder umschließen. Zur Verkehrsschild-Detektion wird der in 2.7 vorgestellte Aggregate Channel Features-Detektor verwendet.

Als nächstes wird jedem Bild, in dem mindestens eine Bounding-Box erstellt wurde, der Zeitpunkt zugeordnet, zu dem es aufgenommen wurde.

Dann wird für jede dieser Bounding-Boxen bestimmt, welches Verkehrsschild sich innerhalb dieser befindet. Dieser Schritt wird in dieser Arbeit als Verkehrsschild-Klassifikation bezeichnet. Zur Verkehrsschild-Klassifikation wird ein Convolutional Neural Network (2.8.2) verwendet.

Schließlich werden für jedes Bild, in dem erfolgreich ein oder mehrere Verkehrsschilder erkannt wurden, dieses Verkehrsschild oder diese Verkehrsschilder den entsprechenden Ortspunkten entlang der Strecke zugeordnet. Dies geschieht mithilfe der Positionsdaten des Ego-Fahrzeugs. Die Positionen und Identifikationen der Verkehrsschilder und die zugehörigen Zeitpunkte stellen das Ergebnis der Verkehrsschilderkennung dar.

3.3.3 Integration der Ergebnisse in das Datenmodell

Im Folgenden wird beschrieben, auf welche Weise die Ergebnisse der Verkehrsobjekt-Größenerkennung und der Verkehrsschilderkennung in das in 2.4 vorgestellte Datenmodell integriert werden.

Für die Integration der Ergebnisse der Verkehrsobjekt-Größenerkennung werden in jeder Verkehrsobjekt-Matrix und in der Matrix des Ego-Fahrzeugs am unteren Ende zwei neue Zeilen angelegt, in die die jeweiligen geschätzten Höhen und Breiten geschrieben werden. Dieser Wert ist in allen Verkehrsobjekt-Matrizen (Ego-Fahrzeug eingeschlossen) über die Zeit konstant, da für jedes Detektionsobjekt ein Schätzwert (= Durchschnittswert aus allen Schätzwerten des jeweiligen Detektionsobjektes) bestimmt wurde.

Für die Integration der Ergebnisse der Verkehrsschilderkennung wurden einige Ansätze in Erwägung gezogen, die im Folgenden beleuchtet werden. Ein wichtiges Kriterium hierbei ist, dass nicht nur die Ergebnisse der im Rahmen dieser Arbeit entstandenen Implementierung mit der gewählten Lösung kompatibel sind, sondern, dass diese eine allgemein anwendbare Möglichkeit darstellt, statische Informationen im betrachteten Datenmodell zu speichern.

Der erste Ansatz, der in Betracht gezogen wurde, ist eine vollständige, den gesamten betrachteten Zeitraum mit einer gewissen Auflösung abbildenden Speicherung der Belegungen der Variablen, die mit den statischen Informationen zusammenhängen. Dieser Ansatz hätte den Vorteil, dass kontinuierliche Größen wie zum Beispiel die Spurbreite oder die Fahrbahnkrümmung ohne Informationsverlust festgehalten werden könnten. Nachteilig ist jedoch, dass durch diesen Ansatz die Forderung einer gewissen Abstrahierung der Originalinformationen verfehlt wird.

Der zweite betrachtete Ansatz ist dadurch gekennzeichnet, dass anders als beim ersten Ansatz eine Abstrahierung der Daten stattfindet. Es werden jeweils nur Informationen in Relation zu Orts- beziehungsweise Zeitpunkten gespeichert, an denen sich statische Objekte befinden oder zu denen sich eine statische Größe verändert. Für die Speicherung dieser Informationen wurden drei Optionen in Betracht gezogen, von denen die dritte als die Option gewählt wurde, die im Rahmen der vorliegenden Arbeit implementiert wurde:

Im Rahmen der ersten Option würde eine neue Matrix erstellt werden. Diese enthält analog zu den bereits vorhandenen Matrizen Informationen über Zeitpunkte in Spaltenform. Diese neuen Zeitpunkte sind jene, zu denen das Ego-Fahrzeug durch Ortspunkte gefahren ist, an denen sich ein statisches Objekt befindet oder sich eine statische Größe verändert.

Im Rahmen der zweiten Option würden die Informationen, die im Rahmen der ersten Option in eine neue Matrix geschrieben würden, an geeigneten Stellen in die Matrix des Ego-Fahrzeugs integriert werden. Das finale Modell enthielte also eine Matrix weniger als in Option 1.

Die dritte Option, welche für diese Arbeit ausgewählt wurde, wird im Folgenden beschrieben:

Es wird eine neue Matrix angelegt, deren Spalten die Punkte entlang der Fahrtstrecke repräsentieren, an denen sich ein statisches Element des Fahrszenarios befindet oder sich eine statische Information des Szenarios im Streckenverlauf verändert. Beispielsweise wird für die hier vorgestellte Verkehrsschilderkennung für jeden Ortspunkt, an dem sich ein Verkehrsschild befindet, eine Spalte erstellt. Des Weiteren könnte beispielsweise auch für jeden Ortspunkt, an dem sich die Anzahl an Fahrbahnspuren verändert, eine Spalte erstellt werden.

Die Zeilen der Matrix haben folgende Inhalte:

- Die erste Zeile, deren korrespondierende Variable mit s bezeichnet sei, enthält die Positionen entlang der Fahrbahn der statischen Elemente (wie zum Beispiel der Verkehrsschilder) beziehungsweise die Positionen der Veränderungen statischer Informationen (wie zum Beispiel die Veränderung der Anzahl an Fahrspuren).
- Die zweite Zeile, deren korrespondierende Variable mit t_{Ego} bezeichnet sei, enthält die Zeitpunkte, zu denen das Ego-Fahrzeug die eine Zeile weiter oben festgehaltenen Ortspunkte passiert hat.
- Die dritte Zeile, deren korrespondierende Variable mit *type* bezeichnet sei, enthält die Typenbezeichnungen der statischen Elemente beziehungsweise der statischen Informationen, die sich an den zwei Zeilen weiter oben festgehaltenen Ortspunkten befinden beziehungsweise verändern. Im Falle der Verkehrsschilderkennung ist diese Typenbezeichnung immer „0“, wobei „0“ für „Verkehrsschild“ steht.
- Die vierte Zeile, deren korrespondierende Variable mit *value* bezeichnet sei, enthält die Identifikationen der entsprechenden statischen Elemente beziehungsweise ab diesem Ortspunkt geltende veränderte statische Informationen (wie zum Beispiel eine veränderte Anzahl an Fahrbahnspuren). Im Falle der Verkehrsschilderkennung werden die Identifikationen der erkannten Verkehrsschilder als natürliche Zahlen kodiert gespeichert.

Ganz links werden dieser Matrix einige weitere Spalten hinzugefügt, in denen einige initiale Werte definiert werden. Im Falle der Verkehrsschilderkennung kann dies zum Beispiel die am Ortspunkt $s = 0$ geltende Geschwindigkeitsbegrenzung sein. An dieser Stelle können beliebige initiale Werte für statische Elemente und Informationen des Szenarios definiert werden wie beispielsweise die Anzahl an Fahrbahnspuren, die am Ortspunkt $s = 0$ gilt.

Die drei beschriebenen Optionen des zweiten Ansatzes bringen im Vergleich mit dem ersten Ansatz den Nachteil mit sich, dass kontinuierliche Größen wie zum Beispiel die Breite oder die Krümmung der Fahrbahn nur unter Informationsverlust festgehalten werden können. Eine Speicherung kontinuierlicher Größen kann in kodierter Form geschehen, sodass zum Beispiel ein Zustand „0“ mit einer Fahrbahnkrümmung im reellen Intervall [a, b) und ein Zustand „1“ mit einer Fahrbahnkrümmung im reellen Intervall [c, d) und so weiter korrespondieren würde. Der Vorteil, dass durch den zweiten Ansatz eine Abstrahierung der Daten erreicht wurde überwog aus Sicht des Autors jedoch diesen Nachteil.

Die dritte Option des zweiten Ansatzes wurde gewählt, da sie verglichen mit der ersten und zweiten Option die kompakteste Darstellung der Daten ermöglicht. Anhang A enthält eine detaillierte Beschreibung der in diesem Abschnitt kurz beschriebenen Ansätze und Optionen.

Kapitel 4

Implementierung

4.1 CarMaker

CarMaker ist eine Simulations-Software des Unternehmens IPG, mit der personalisierte Fahrszenarien simuliert werden können. In CarMaker können unter anderem die Fahrmanöver und Eigenschaften der Verkehrsteilnehmer, die Fahrstrecke und die Ausgabedaten frei ausgewählt werden. Abbildung 4.1 zeigt die graphische Oberfläche von CarMaker 8.0.

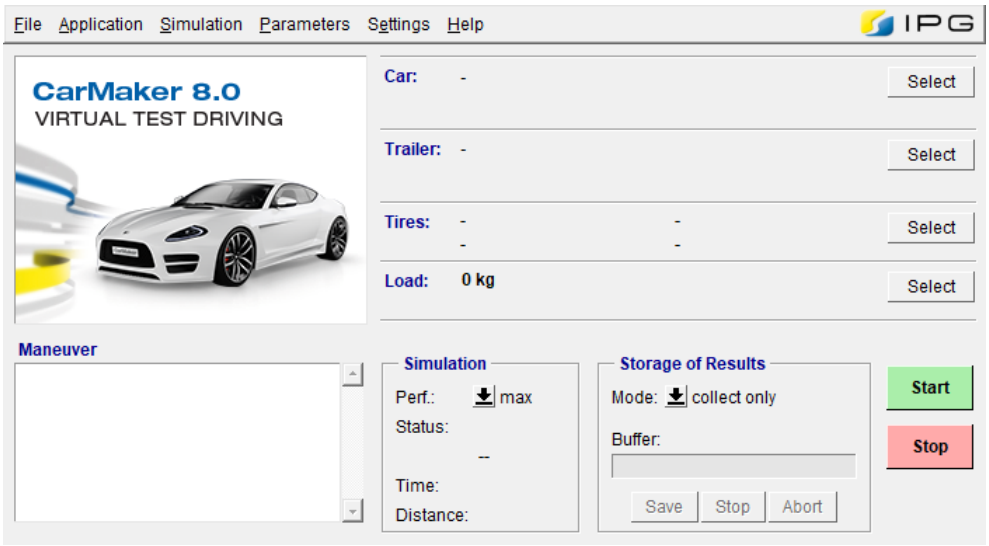

Abbildung 4.1: Graphische Oberfläche der Simulationssoftware CarMaker 8.0

Über ein Menü kann ausgewählt werden, welche Daten erhoben werden und in welcher Frequenz die Erhebung stattfindet. Das Menü hierfür ist in Abbildung 4.2 dargestellt. Zur Visualisierung der Fahrszenarien steht IPGMovie zur Verfügung. Hier können die Position, Ausrichtung und weitere Eigenschaften einer simulierten Kamera eingestellt werden und Bildsequenzen in gängigen Video- und Bildformaten exportiert werden. Abbildung

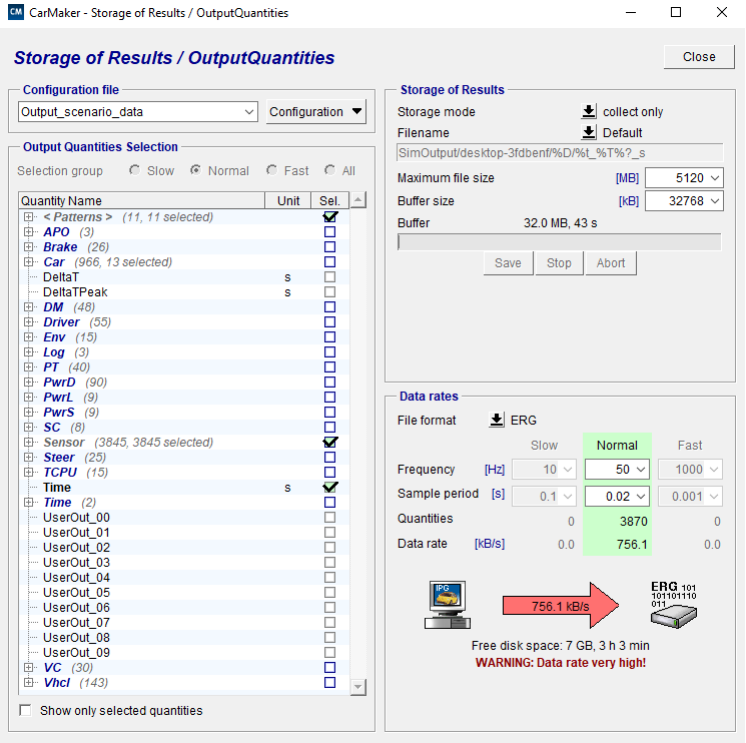

Abbildung 4.2: Menü für Datenexport in CarMaker 8.0

4.3 zeigt ein Bildschirmfoto von IPGMovie und dem Menü zum Einstellen der Kameraoptionen.

CarMaker wurde im Rahmen dieser Arbeit verwendet, um Fahrszenarien zu erstellen und zu visualisieren und die so gewonnenen Daten zur Weiterverarbeitung zu exportierten. Diese simulierten Daten wurden im Rahmen dieser Arbeit verwendet, um die Performanz der implementierten Funktionen zu bestimmen.

4.2 Auswahl relevanter Daten

Für diese Arbeit wurden folgende Daten aus CarMaker beziehungsweise IPGMovie exportiert und für die weitere Verarbeitung verwendet:

Als Äquivalent zu Daten, die aus einer kontinuierlichen Bestimmung der Position des Ego-Fahrzeugs entlang der Fahrstrecke resultieren, wurde die CarMaker-Größe „Car.Road.sRoad“ verwendet.

Als Äquivalent zu Daten, die aus einer kontinuierlichen Bestimmung der relativen Positionen der Verkehrsobjekte bezüglich des Ego-Fahrzeugs resultieren, wurden die CarMaker-Größen „Sensor.Object.OB01.Obj.T*.NearPnt.ds.x|y|z“[1] verwendet.

[1] „*“ steht für die den jeweiligen Verkehrsobjekten zugeordneten natürlichen Zahlen, „|“ steht für „beziehungsweise“

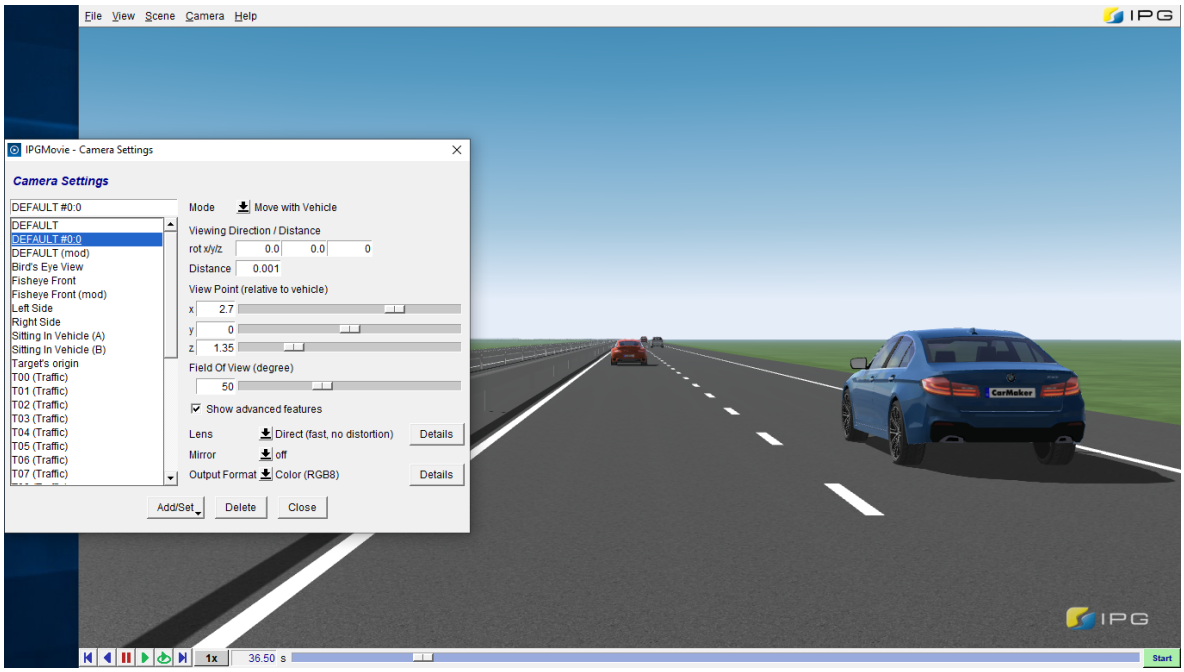

Abbildung 4.3: IPGMovie und Menü zum Einstellen der Kameraoptionen. Eigene Darstellung; erstellt mit IPGMovie.

Als Äquivalent zu Kamerabildern wurden simulierte Kamerabilder aus IPGMovie verwendet. Hierbei wurden eine nach vorne gerichtete und eine nach hinten gerichtete Kamera simuliert; die gewählten Kameraeinstellungen sind in Tabelle 4.1 dargestellt.

4.3 Beschreibung der Implementierung der einzelnen Funktionen

4.3.1 Verkehrsobjekt-Größenbestimmung

4.3.1.1 Erstellung des Trainingsdatensatzes

Mit dem Ziel, die Modellparameter f beziehungsweise f_{Breite} und $f_{Höhe}$ zu bestimmen wurden Trainingsdaten erstellt. Hierzu wurden zunächst in CarMaker mehrere Fahrszenarien simuliert und in IPGMovie dargestellt. Diese Szenarien zeichnet aus, dass in ihnen neben dem Ego-Fahrzeug nur jeweils ein anderes Verkehrsobjekt enthalten ist, welches durch die nach vorne gerichtete Kamera (vgl. Tabelle 4.1) aus verschiedenen Richtungen und Perspektiven zu sehen ist. Bei jedem dieser Szenarien wurde für das im Kamerabild sichtbare Verkehrsobjekt ein anderes Automodell gewählt. Je Szenario wurde mit IPGMovie eine Bildsequenz über den gesamten Zeitraum, in dem sich das Verkehrsobjekt im Sichtfeld der Kamera befand, gespeichert. Dies geschah mit einer Frequenz von einem Bild pro Sekunde und mit einer Bildauflösung von $1920 * 1080$ Pixel. Diese Bilder - zusammen mit den zu jedem Aufnahmezeitpunkt gemessenen Abständen in x-Richtung zwischen der

	Nach vorne gerichtete Kamera	Nach hinten gerichtete Kamera
view point x	2.7 m	0.8 m
view point y	0	0
view point z	1.35 m	1.35 m
rot x	0	0
rot y	0	0
rot z	0	180
Distance	0.001	0.001
Field of view	50°	50°
Lens	Direct (fast, no distortion)	Direct (fast, no distortion)
Mirror	off	off
Output Format	Color (RGB8)	Color (RGB8)
Mode	Move with Vehicle	Move with Vehicle

Tabelle 4.1: Die verwendeten Kameraeinstellungen: Die Variablenbezeichnungen wurden zur besseren Nachvollziehbarkeit ohne Veränderungen aus IPGMovie übernommen. Eigene Darstellung.

Kamera und den betrachteten Verkehrsobjekten - sowie die wahren Breiten und Höhen der Verkehrsobjekte bilden die Grundlage für die Erstellung des Trainingsdatensatzes.

Für die Erstellung des Trainingsdatensatzes wurde das Matlab-Skript „create_training_dataset“ erstellt. In diesem finden sich detaillierte Erklärungen und Kommentare zu den einzelnen Berechnungsschritten. Das Ergebnis nach Ausführung des Skriptes ist eine Tabelle mit Trainingsdaten, die mit dem jeweiligen Szenario korrespondiert. Durch mehrmaliges Ausführen des Skriptes für die Bild-, Abstands- und Abmessungsdaten von verschiedenen Szenarien erhält man also mehrere Datensätze. Diese wurden zusammengefügt. Zur Verdeutlichung zeigt Tabelle 4.2 beispielhaft einige Zeilen des so erstellten Trainingsdatensatzes. Der für die im nächsten Abschnitt beschriebene Bestimmung der Modellparameter verwendete und auf diese Weise erstellte Datensatz enthält 538 Beobachtungen, also 538 solcher in Tabelle 4.2 beispielhaft dargestellter Zeilen.

4.3.1.2 Bestimmung der Modellparameter

Um optimale Werte für die Modellparameter f beziehungsweise f_{Breite} und $f_{Höhe}$ zu bestimmen wurde das Matlab-Skript „main_training“ erstellt. Der Zweck dieses Skriptes ist es, die Optimierungsprobleme $P1$ und $P2$ (3.3.1.1) zu lösen beziehungsweise Lösungen derselben zu approximieren. Das Lösen von $P1$ liefert einen im Sinne der Summe der quadratischen Abweichungen optimalen Wert für die Brennweite f. Die Lösungen von $P2$ sind im Sinne der Summe der quadratischen Abweichungen optimale Werte für die zwei Modellparameter f_{Breite} und $f_{Höhe}$. Das genannte Skript enthält hierzu detaillierte Erklärungen und Kommentare.

width image [px]	height image [px]	dist_x [m]	width ground truth [m]	height ground truth [m]	description
95	76	40.03	1.85	1.27	"BMW_5_2017_Blue"
146	117	30.15	1.85	1.27	"BMW_5_2017_Blue"
208	167	19.44	1.85	1.27	"BMW_5_2017_Blue"
48	38	118.43	2.01	1.53	"Chevrolet_Silverado1500_2013"
48	38	111.61	2.01	1.53	"Chevrolet_Silverado1500_2013"
44	35	105.89	2.01	1.53	"Chevrolet_Silverado1500_2013"

Tabelle 4.2: Beispielhafte Darstellung einiger Zeilen des Trainingsdatensatzes. Alle Werte wurden für diese Darstellung auf die zweite Nachkommastelle gerundet. Eigene Darstellung.

4.3.1.3 Anwendung

Als Szenario für die fortlaufende Validierung der Verkehrsobjekt-Größenbestimmung während des Implementierungsprozesses diente das Fahrszenario, das durch die CarMaker-Datei „Ground_Truth_Highway_two_lane_traffic.dat“ definiert ist[2]. In diesem Fahrszenario fährt das Ego-Fahrzeug auf einer Autobahn mit zwei mal zwei Fahrspuren. Im Zuge des Szenarios überholt das Ego-Fahrzeug drei andere Fahrzeuge und wechselt währenddessen mehrmals die Fahrspur. Insgesamt sind im Fahrszenario acht Fahrzeuge zu sehen, von denen sieben auf der rechten Spur und eins auf der linken Spur fährt. Wichtig hierbei ist, dass das Fahrszenario Szenen enthält, in denen sich zwei Fahrzeuge nebeneinander im Sichtfeld der Kameras des Ego-Fahrzeugs befinden. Das Szenario wurde auf diese Weise konzipiert, um zu bestätigen, dass die vorliegende Implementierung in der Lage ist, pro Bild mehrere erstellte Bounding-Boxen zu verarbeiten und ihre Abmessungen den richtigen Verkehrsobjekten zuzuordnen.

Die mit CarMakers IPGMovie erstellten Kamerabilder wurden für diese Anwendung mit einer Frequenz von einem beziehungsweise zwei Bildern pro Sekunde und mit einer Bildauflösung von $1920 * 1080$ Pixel gespeichert. Da sowohl die Anwendung mit einem Bild pro Sekunde als auch die Anwendung mit zwei Bildern pro Sekunde zufriedenstellende Ergebnisse lieferten, beschränken sich die folgenden Beschreibungen auf die Anwendung mit einem Bild pro Sekunde, da eine niedrigere Bildrate eine erhebliche Zeitersparnis bei der Fahrzeugdetektion mit sich bringt.

Die Funktionsweise der Verkehrsobjekt-Größenbestimmung wird im Folgenden anhand dieses Beispielszenarios beschrieben. Hierbei werden die Funktionen der einzelnen Codeabschnitte nacheinander erklärt. Die Skripte, die den Code zur Verkehrsobjekt-Größenbestimmung enthalten, heißen „main_application_model_3_4“ und „main_application_model_2“, wobei ersteres die Implementierung des Modells mit

[2]Zusätzlich zu dieser Validierung wurde die finale Implementierung mit einem weiteren Szenario getestet. Hierzu näheres in 5.1.2.1

einem Modellparameter f und letzteres die Implementierung des Modells mit zwei Modellparametern f_{Breite} und $f_{Höhe}$ darstellt.

Im Abschnitt „Input by user“ ist der Nutzer dazu aufgefordert einige Einstellungen vorzunehmen, wie zum Beispiel die Positionen der verwendeten Kameras zu definieren.

Als nächstes folgt das Einlesen der Daten und die Konvertierung in die richtigen Formate.

Dann werden die „Detektionsobjekte“ definiert: Pro Ein- und Austrittsereignis eines Verkehrsobjektes in den Sichtbereich des Ego-Fahrzeugs wird ein separates „Detektionsobjekt“ definiert. So wird die Tatsache abgebildet, dass in der Realität für ein Verkehrsobjekt, welches sich in einem Zeitintervall [t1, t2] im Sichtbereich des Ego-Fahrzeugs befindet, im Zeitintervall [t2, t3] nicht im Sichtbereich befindet und im Zeitintervall [t3, t4] wieder in diesem befindet, die Detektion in [t1, t2] nicht mit Gewissheit der Detektion in [t3, t4] zugeordnet werden kann.

Als nächstes findet eine Sortierung der Detektionsobjekte nach derselben Logik, die auch für das Datenmodell, das in [Kha19] entwickelt und in 2.4 vorgestellt wurde, Verwendung fand, statt. So wird die spätere korrekte Zuordnung der berechneten Abmessungen der Detektionsobjekte zu den entsprechenden Einträgen im durch die Implementierung in [Kha19] erstellten Tensor erreicht.

Nun werden in allen zur Verfügung stehenden Kamerabildern Verkehrsobjekte detektiert oder, präziser ausgedrückt, die Bounding-Boxen, die diese umschließen, erstellt. Diese werden mitsamt des Zeitpunktes, mit dem das entsprechende Kamerabild korrespondiert, gespeichert. Zur Detektion der Verkehrsobjekte wird ein bereits trainierter Aggregate Channel Features-Detektor verwendet (2.7). Dieser wird durch Matlab im Rahmen der Automated Driving Toolbox™ bereitgestellt, ist unter dem Namen „vehicleDetectorACF“ abrufbar und wurde mit realen Bildern trainiert. Im Zuge der Implementierung dieses Programmteils wurde außerdem ein anderer von Matlab bereitgestellter Detektor in Betracht gezogen, welcher unter dem Namen „vehicleDetectorFasterRCNN“ aus der zu Matlab zugehörigen Deep Learning Toolbox™ abrufbar ist und ein CNN zur Detektion verwendet. Da dieser Detektor jedoch sowohl im Bezug auf die Position und die Größe der detektierten Bounding-Boxen als auch im Bezug auf die benötigte Berechnungszeit im Vergleich zum Aggregate Channel Features-Detektor schlechtere Ergebnisse lieferte, entschied der Autor letzteren zu verwenden[3]. Für einen Vergleich von „Aggregate Channel Features“ mit anderen gängigen Technologien sei auf [DABP14] verwiesen: Hier wird aufgezeigt, dass ACF sowohl im Bezug auf die Genauigkeit als auch im Bezug auf die Berechnungszeit sehr gute Ergebnisse liefert. Der Schritt der Detektion beziehungsweise der Bounding-Box-Erstellung ist eine Quelle von Messfehlern, in dem Sinne, dass erstellte Bounding-Boxen zu groß oder zu klein sein können oder an Orten im Bild erstellt werden können, an denen sich kein Fahrzeug befindet.

Alle erstellten Bounding-Boxen werden entweder einem Detektionsobjekt zugeordnet oder im weiteren Verlauf nicht mehr berücksichtigt. Dieser Schritt wird im Folgenden in einigen Teilschritten beschrieben:

[3]Zum Testen der Performanz von „vehicleDetectorFasterRCNN“ wurde das Matlab-Skript „create_training_dataset_rcnn“ erstellt und verwendet.

1. Hierzu wird zuerst für die Detektionsobjekte, die sich im Sichtbereich der Kamera befinden, bestimmt, wo im Kamerabild diese theoretisch abgebildet sein müssten. Hierfür werden die Gleichungen 3.1 und 3.2 beziehungsweise 3.7 und 3.8 und die Positionen der Detektionsobjekte verwenden. Zur Erinnerung: Es gilt $y_{Bild} = \frac{y_{Real}}{x_{Real}} * f$ und $z_{Bild} = \frac{z_{Real}}{x_{Real}} * f$, falls das Modell gewählt wurde, das durch Optimierungsproblem $P1$ geschätzt wurde, und $y_{Bild} = \frac{y_{Real}}{x_{Real}} * f_{Breite}$ und $z_{Bild} = \frac{z_{Real}}{x_{Real}} * f_{Höhe}$, falls das Modell gewählt wurde, das durch Optimierungsproblem $P2$ geschätzt wurde (vgl. 3.3.1.1).

2. Die Zentren der Bounding-Boxen werden berechnet. Hierbei ist zu beachten, dass sich diese Position auf das Koordinatensystem der Bilder aus IPGMovie bezieht, dessen Basisvektoren nach rechts und nach unten zeigen. Deshalb werden die Positionen der Zentren der Bounding-Boxen noch in ihre Darstellung im in den Abschnitten 2.5.2 und 3.3.1.1 beschriebenen und in Abbildung 2.6 visualisierten Koordinatensystem umgerechnet.

3. Schließlich findet die Zuordnung in den folgenden zwei Schritten statt, wobei die zwei Schritte für jede Bounding-Box durchgeführt werden:

   (a) Falls die theoretischen Positionen im Bild mehrerer Kandidaten sehr nah beieinander liegen, wird nur der Kandidat berücksichtigt, der sich näher an der Kamera befindet. So wird der Fall behandelt, dass sich Verkehrsobjekte aus Sicht des Ego-Fahrzeugs auf einer ähnlichen Sichtachse, also hintereinander befinden.

   (b) Anschließend wird der Kandidat auswählt und der jeweiligen Bounding-Box zugeordnet, dessen theoretische Position sich gemäß des euklidischen Abstands am nächsten am Zentrum der Bounding-Box befindet. Falls sich kein Kandidat in ausreichender Nähe zur Bounding-Box befindet (dies ist in der vorliegenden Implementierung der Fall, wenn alle euklidischen Abstände zwischen den theoretischen Positionen der Kandidaten und dem Bounding-Box-Zentrum größer als $2 * Breite_{Boundingbox_{aktuell}}$ sind), wird diese Bonding-Box als Detektionsfehler betrachtet und gelöscht. Dieser Schritt stellt somit eine erste Behandlung von Messfehlern dar.

Der nächste Abschnitt im Skript ist von optionaler Natur und in der beiliegenden Version mithilfe einer „if 0 ... end“-Konstruktion deaktiviert. Sein Zweck ist die visuelle Verifizierung des Zuordnungsprozesses, indem jede detektierte Bounding-Box im korrespondierenden Kamerabild gemeinsam mit den theoretischen Bildpositionen der Verkehrsobjekte angezeigt wird, wobei die Markierung des der Bounding-Box zugeordneten Verkehrsobjektes farblich von den anderen Markierungen unterschieden wird. Eine solche Visualisierung ist in Abbildung 4.4 beispielhaft dargestellt. Durch das Setzen jeweils eines Breakpoints[4] in Zeile 683 für die Bilder der Vorderkamera und Zeile 715 für die Bilder der Hinterkamera

[4]Ein Breakpoint wird in Matlab verwendet um in Skripten und Funktionen Stellen zu markieren, an denen während der Ausführung derselben pausiert werden soll. Der Benutzer kann einen Breakpoint durch klicken auf die waagrechten Striche rechts von den Zeilenzahlen am linken Rand des Fensters setzen.

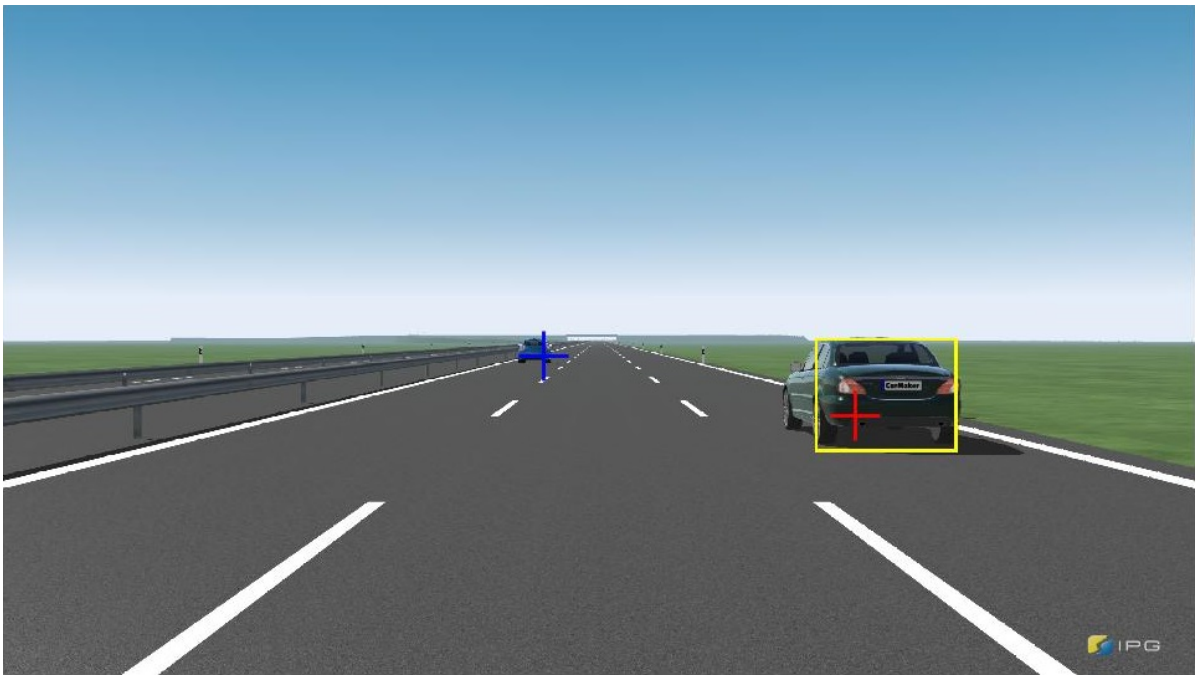

Abbildung 4.4: Darstellung einer Visualisierung des Zuordnungsprozesses, der jeweils einer erstellten Bounding-Boxen das richtige Fahrzeug zuordnet. Die betrachtete Bounding-Box ist in gelb dargestellt und das Fahrzeug, welches dieser Bounding-Box zugeordnet wurde, ist mit einem roten Kreuz markiert. Im Hintergrund ist ein weiteres Fahrzeug, welches nicht ausgewählt wurde, zu sehen, welches mit einem blauen Kreuz markiert ist. Eigene Darstellung erstellt mit IPG CarMaker und Matlab.

kann iterativ für jede der detektierten Bounding-Boxen die beschriebene Visualisierung erstellt und angezeigt werden.

Es folgt die zweite Behandlung von Messfehlern: Aus dem Betrachten der beschriebenen Visualisierung erwuchs die Vermutung, dass in Situationen, in denen das Ego-Fahrzeug andere Fahrzeuge überholt oder von diesem überholt wird - diese im Kamerabild also von einem seitlichen Blickwinkel aus abgebildet werden -, der Detektor die Tendenz hat, Bounding-Boxen zu erstellen, die etwas zu breit sind. Um dem zu begegnen, können Bounding-Boxen, die mit Situationen korrespondieren, in denen das zugeordnete Verkehrsobjekt einen kleinen Abstand in x-Richtung zur Kamera aufweist, gelöscht werden. Die Schranken für den Abstand in x-Richtung von der Vorder- beziehungsweise Rückkamera werden am Anfang des Skriptes durch die Belegung der Variablen „limit_front“ und „limit_rear“ vom Nutzer bestimmt. Durch die Ausführung des hier vorgestellten Codes mit einer Vielzahl von Werten für „limit_front“ und „limit_rear“ wurde die Auswirkung dieser Schranken auf die Modellgüte untersucht. Hierbei kam der Autor zu dem Schluss, dass die positive Auswirkung solcher Schranken sehr geringfügig ist und entschied somit, die Werte der beiden Variablen auf Null zu setzen, wodurch sie gar keine Auswirkung mehr haben.

Als nächstes werden aus den Breiten und Höhen der nach den vorangegangenen Schritten

verbleibenden Bounding-Boxen die reellen Breiten und Höhen der zugehörigen Detektionsobjekte geschätzt: Hierzu werden wieder die Gleichungen 3.1 und 3.2 beziehungsweise 3.7 und 3.8 verwendet, aus denen sich durch Umstellen $y_{Real} = \frac{y_{Bild} * x_{Real}}{f}$ und $z_{Real} = \frac{z_{Bild} * x_{Real}}{f}$ beziehungsweise $y_{Real} = \frac{y_{Bild} * x_{Real}}{f_{Breite}}$ und $z_{Real} = \frac{z_{Bild} * x_{Real}}{f_{Höhe}}$ ergibt.

Der nächste Abschnitt im Matlab-Skript enthält eine Behandlung von Ausreißern in den im vorherigen Schritt geschätzten Breiten und Höhen: Hierbei werden alle geschätzten Breiten und Höhen gelöscht, für die der absolute Abstand d zum Median der für das jeweilige Detektionsobjekt geschätzten Breiten beziehungsweise Höhen gilt: $d > tol * stDev$, wobei tol eine vom Nutzer definierte positive reelle Zahl und $stDev$ die Standardabweichung der für das jeweilige Detektionsobjekt geschätzten Breiten beziehungsweise Höhen bezeichne. Wie schon im Bezug auf die zweite Behandlung von Messfehlern weiter oben, stellte auch hier der Autor fest, dass diese Behandlung von Ausreißern einen sehr geringen positiven Effekt mit sich bringt. Da eine solche Behandlung von Ausreißern jedoch generell wünschenswert ist, um Resultate grober Messfehler nicht in das Endergebnis aufzunehmen, entschied der Autor, die Schranke tol für das Löschen von Breiten und Höhen auf den relativ hohen Wert 4 zu setzen. Dies bedeutet, dass jede Breite beziehungsweise Höhe, welche sich außerhalb des Bereichs mit Radius von vier Breite- beziehungsweise Höhe-Standardabweichungen des entsprechenden Detektionsobjektes um den Breite- beziehungsweise Höhe-Median des entsprechenden Detektionsobjektes befinden würde, gelöscht würde.

Als nächstes wird für jedes Detektionsobjekt und für Breite und Höhe separat das arithmetische Mittel gebildet. Die Resultate werden als Zwischenergebnis mitsamt Index des jeweiligen Detektionsobjektes in den Instanzen „widths_array“ und „heights_array“ gespeichert. Die so resultierenden Schätzer für die Breiten beziehungsweise Höhen der einzelnen Detektionsobjekte sind also gegeben mit $\widehat{Breite_i} = \frac{1}{n_i} \sum_{j}^{n_i} Breite_{j,berechnet}$ und $\widehat{Höhe_i} = \frac{1}{n_i} \sum_{j}^{n_i} Höhe_{j,berechnet}$, wobei i den Index des jeweiligen Detektionsobjektes und n_i die Anzahl an berechneten Breiten beziehungsweise Höhen für Detektionsobjekt i bezeichnet. Der Zweck der in diesem Schritt erfolgten Mittelung über die berechneten Breiten und Höhen ist es, den Effekt von im Detektionsschritt entstandenen Messfehlern (in diesem Fall vor allem von zu großen oder zu kleinen Bounding-Boxen) auf die Endergebnisse zu minimieren.

Es folgt die Validierung der Ergebnisse: Falls der Nutzer die Variable „testing“ im Abschnitt „Input by user“ mit dem Wert 1 belegt hat, werden mittels der Funktion „read_DAT_file.m“ aus der das Szenario definierenden Datei „Ground_Truth_Highway_two_lane_traffic.dat“ die wahren Breiten und Höhen der Verkehrsobjekte ausgelesen. Aufgrund der geschätzten Werte und der wahren Werte werden einige Validierungskennzahlen berechnet. Hierauf wird in 5.1 näher eingegangen.

Schlussendlich werden die in „widths_array“ beziehungsweise „heights_array“ enthaltenen Endergebnisse zur in 2.4 vorgestellten Datenstruktur hinzugefügt, welche auf Basis des hier verwendeten Szenarios und durch die in [Kha19] vorgestellte Implementierung erstellt wurde. Zu beachten ist, dass hierbei jeweils die gesamte „Breite-Zeile“ beziehungsweise die gesamte „Höhe-Zeile“ der Matrix eines Verkehrsobjektes i mit diesem einen Schätzwert $\widehat{Breite_i}$ beziehungsweise diesem einen Schätzwert $\widehat{Höhe_i}$ befüllt wird, die gesamte Zeile

also jeweils für alle Spalten den gleichen Wert enthält. Da die Spalten der Verkehrsobjekt-Matrizen mit verschiedenen Zeitpunkten korrespondieren und die Breiten und Höhen von Verkehrsobjekten zeitlich invariante Größen sind, ist diese Darstellungsform konsistent.

4.3.2 Verkehrsschilderkennung

4.3.2.1 Training eines ACF-Detektors zur Verkehrsschilddetektion

Zur Detektion von Verkehrsschildern wurde für diese Arbeit der in 2.7 vorgestellte Aggregate Channel Features-Detektor verwendet. Anders als bei der Verkehrsobjekt-Größenbestimmung (4.3.1) lag kein bereits fertig trainierter ACF-Detektor vor, weshalb ein neuer Detektor mithilfe geeigneter Daten trainiert wurde. Als Trainingsdatensatz fungierte der „German Traffic Sign Detection Benchmark (GTSDB)"-Datensatz, welcher durch Houben et al. in [HSS+13] vorgestellt wird. Bei dem Datensatz handelt es sich um 900 digitale, reale Bilder, die verschiedene Verkehrssituationen und Verkehrsschilder zeigen. Als Ground Truth-Informationen liegen für jedes Bild die Positionen und Abmessungen der in den Bildern enthaltenen Verkehrsschilder bei. Mithilfe dieser Ground Truth-Informationen und des Matlab-Skriptes „ts_detection_build_training_set" wurde eine geeignete Tabelle mit Trainingsdaten erstellt. Mit dem Matlab-Skript „ts_detection_train_acf_detector" wurde schließlich ein ACF-Detektor trainiert, der in der Lage ist, in beliebigen digitalen Bildern abgebildete Verkehrsschilder mit Bounding-Boxen zu versehen. Das Detektionsresultat ist anhand eines Beispielbildes, welches in CarMakers IPGMovie erstellt wurde, in Abbildung 4.5

Abbildung 4.5: Beispielhafte Anwendung des ACF-Detektors auf ein in CarMakers IPGMovie erstelltes Bild. Eigene Darstellung; erstellt mit IPGMovie und Matlab.

dargestellt.

4.3.2.2 Training des CNNs zur Verkehrszeichenerkennung

Zur Klassifikation der durch den ACF-Detektor markierten Verkehrsschilder wird ein Convolutional Neural Network (siehe 2.8.2) verwendet, dessen Architektur auf der des im Matlab-Beispiel [Mat] als „RecognitionNet“ bezeichneten CNNs basiert. Für diese Anwendung wurden zwei Schichten - die letzte verborgene Schicht und die Ausgabeschicht - ausgetauscht. Diese Veränderung wurde vorgenommen, damit das CNN in der Lage ist, Verkehrsschilder 43 verschiedenen Klassen zuzuordnen. In Abbildung 4.6 sind die Schich-

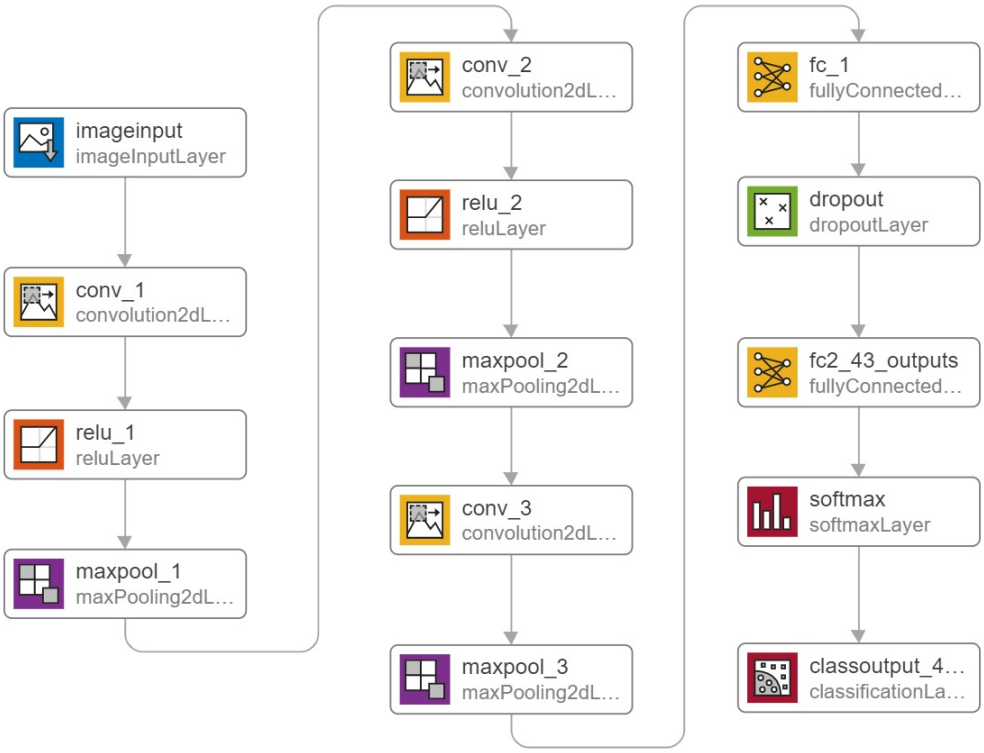

Abbildung 4.6: Schematische Darstellung der Schichten des zur Klassifikation verwendeten CNNs sowie der verwendeten Aktivierungsfunktionen. Die mit „dropout“ bezeichnete Schicht sorgt dafür, dass jede aus der vorherigen Schicht kommende Größe lediglich mit einer Wahrscheinlichkeit von 0.9 an die nächste Schicht weitergereicht wird. Dies ist eine gängige Praxis um eine Überanpassung des Modells an die Trainingsdaten zu verhindern. Eigene Darstellung; erstellt mit Matlab.

ten des verwendeten CNNs, sowie die verwendeten Aktivierungsfunktionen schematisch dargestellt.

Als Trainingsdatensatz fungierte hier der „German Traffic Sign Recognition Benchmark“-Datensatz, welcher durch Stallkamp et al. in [SSSI11] vorgestellt wurde. Bei diesem handelt es sich um mehr als 39000 digitale, reale Bilder, die durch eine Ordnerstruktur 43

verschiedenen Klassen zugeordnet sind und je ein mit diesen Klassen korrespondierendes Verkehrsschild enthalten. Dieser Trainingsdatensatz wurde zum Zweck der fortlaufenden Validierung während des Trainings in einen Trainings- und Validierungsdatensatz aufgeteilt. Die hierfür gewählte Aufteilung beträgt 80% für den Trainings- und 20% für den Validierungsdatensatz. Als Validierungsdatensatz wurde ein Teil des Trainingsdatensatzes und nicht der ebenfalls vorhandene separate Testdatensatz verwendet, um sicherzustellen, dass nach dem Beenden des Trainings die Möglichkeit besteht, die Performanz des CNNs in einem finalen Test mit einem Datensatz zu überprüfen, der während des Trainings keinerlei Rolle gespielt hat. Da der Trainingsdatensatz aus vielen Serien von mehreren zeitlich aufeinanderfolgenden Bildern besteht, wurde bei der Aufteilung in Trainings- und Validierungsdatensatz darauf geachtet, dass alle zeitlich zusammenhängenden Bilderserien jeweils vollständig entweder zum Trainings- oder zum Validierungsdatensatz hinzugefügt wurden. Ohne diese Maßnahme würde - aufgrund der starken Ähnlichkeit der Bilder in den zeitlich zusammenhängenden Bilderserien untereinander - der Validierungsdatensatz eine hohe Ähnlichkeit zum Trainingsdatensatz aufweisen und somit das Validierungsergebnis verfälscht werden.

Für das Training des CNNs wurde das in 2.9 beschriebene Transfer Learning-Vorgehen gewählt. Als Initialwerte der Gewichte des CNNs wurden somit die Gewichte des trainierten „RecognitionNet“ [Mat] verwendet. Für das Training wurde die Anpassungsgeschwindigkeit der Gewichte der ausgetauschten Schichten zehn mal so hoch eingestellt wie die Anpassungsgeschwindigkeit der Gewichte der unveränderten Schichten. Anschließend wurde das CNN für mehrere Epochen mit dem oben genannten Trainingsdatensatz trainiert, bis schließlich eine Validierungsgenauigkeit von mehr als 93.9% erreicht wurde. Die Validierungsgenauigkeit wurde während des Trainingsprozesses mithilfe des Validierungsdatensatzes in regelmäßigen zeitlichen Abständen als der Quotient der Anzahl an richtigen Klassifizierungen und der Gesamtanzahl an Klassifizierungen bestimmt.

4.3.2.3 Anwendung

Als Szenario für die fortlaufende Validierung der Verkehrsschilderkennung während des Implementierungsprozesses diente das Fahrszenario, das durch die CarMaker-Dateien „Ground_Truth_Highway_two_lane_traffic_road_signs.dat“ und „Simple_Highway_circle_road_signs.rd5“ definiert ist[5]. Dieses ist mit dem für die Größenerkennung verwendeten Szenario identisch, bis auf den Zusatz, dass in diesem am rechten Rand der Fahrbahn insgesamt 38 Verkehrsschilder zu sehen sind, die alle jeweils paarweise voneinander verschieden sind. Die Menge dieser Verkehrsschilder stellt die Menge der Verkehrsschilder dar, die sowohl im in 4.3.2.2 vorgestellten Trainingsdatensatz enthalten als auch in CarMaker darstellbar sind. Die für diese Anwendung mit CarMakers IPGMovie erstellten Kamerabilder wurden mit einer Frequenz von fünf Bildern pro Sekunde und mit einer Bildauflösung von $1920 * 1080$ Pixel gespeichert. Niedrigere Bildfrequenzen führten zu nicht zufriedenstellenden Ergebnissen. Der Grund hierfür liegt in der Art und Weise, wie durch die hier beschriebenen Funktionen mutmaßlich richtig

[5]Zusätzlich zu dieser Validierung wurde die finale Implementierung mit einem weiteren Szenario getestet. Hierzu näheres in 5.1.2.2

erkannte Verkehrsschilder von mutmaßlichen Detektions- oder Klassifikationsfehlern unterschieden werden. Dies wird in diesem Abschnitt weiter unten beschrieben.

Zur Ausführung der Verkehrsschilddetektion und -klassifikation dient das Matlab-Skript „main_ts_detection_and_recognition“. Dieses beginnt mit dem Abschnitt „Input by user“, in dem der Benutzer einige Parameter wie zum Beispiel die Postion der am Ego-Fahrzeug montierten Kamera und falls gewünscht Initialbelegungen für statische Variablen mit geeigneten Werten belegen kann.

Darauf folgt die Detektion von Verkehrsschildern mittels des ACF-Detektors, dessen Training in 4.3.2.1 beschrieben wird. Die Ausgabe dieses Schrittes ist je Kamerabild eine (manchmal leere) Menge von Bounding-Boxen. Das CNN, das in 4.3.2.2 vorgestellt wird, führt im Anschluss für die durch diese Bounding-Boxen definierten Bildausschnitte die Klassifizierung durch. Hierfür werden die Bounding-Boxen jeweils um einige Pixel vergrößert, um ihre Ähnlichkeit zu den Trainingsbildern des CNNs zu gewährleisten und um sicherzustellen, dass sich jeweils das ganze Verkehrsschild im Bildausschnitt befindet. Außerdem werden Bounding-Boxen, deren Größe eine gewisse Schranke unterschreiten, nicht weiterverarbeitet. Dieses Vorgehen wurde gewählt, da sehr kleine Bounding-Boxen meist weit entfernte Verkehrsschilder markieren, die im Bild durch ihre Entfernung zur Kamera mit einer sehr niedrigen Auflösung dargestellt sind, und somit oft zu Fehlklassifizierungen führen.

Im Anschluss an Detektion und Klassifikation werden je Bild die Ergebnisse graphisch dargestellt. Dies geschieht, indem die erstellten Original-Bounding-Boxen und die vergrößerten Bounding-Boxen, welche den klassifizierten Bildausschnitten entsprechen, sowie die Klassifikationsergebnisse im entsprechenden Bild eingefügt werden und dann das Resultat visualisiert wird. Eine solche Visualisierung zeigt Abbildung 4.7. In dieser Abbildung wird ebenfalls deutlich, dass die vorliegende Implementierung in der Lage ist mehrere in einem Bild sichtbare Verkehrsschilder zu verarbeiten.

Auf die Detektion und Klassifikation folgen einige Berechnungsschritte, die den Zweck haben, richtig erkannten Verkehrsschildern die entsprechenden Zeit- und Ortspunkte entlang der Fahrstrecke zuzuordnen und Detektions- oder Klassifikationsfehler von der Weiterverarbeitung auszuschließen. Hierzu wird zunächst jedem Bild, in dem ein oder mehrere Verkehrsschilder erkannt wurden, der Zeitpunkt zugeordnet, zu dem dieses Bild aufgenommen wurde. Dann werden für jedes dieser Bilder folgende Bedingungen geprüft:

- Bedingung 1: Für zwei aufeinanderfolgenden Bilder a und b ist ein bestimmtes Verkehrsschild in a erkannt worden und in b nicht.
- Bedingung 2: In mindestens drei von den vier direkt vor a und b aufgenommenen Bildern ist dieses Verkehrsschild erkannt worden.
- Bedingung 3: In mindestens drei von den vier direkt nach a und b aufgenommenen Bildern ist das Verkehrsschild nicht erkannt worden.

Alle Zeitpunkte, die zu Bildern a zugehörig sind und für die diese drei Bedingungen erfüllt sind, werden mitsamt der Identifikation des entsprechenden Verkehrsschildes gespeichert.

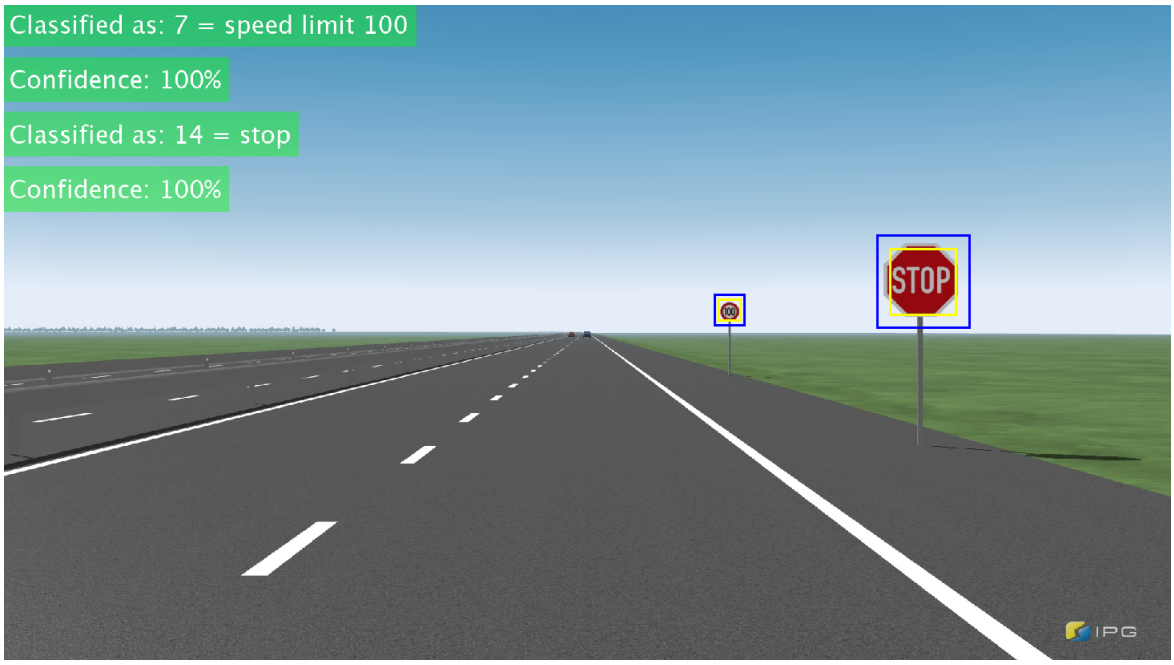

Abbildung 4.7: Visualisierung von Detektions- und Klassifikationsergebnissen. Die originalen Bounding-Boxen sind in gelb, die vergrößerten in blau dargestellt. Eigene Darstellung; erstellt mit IPGMovie und Matlab.

Als nächstes werden alle diese Zeitpunkte um eine Konstante „delay“ erhöht, um eine möglichst gute Übereinstimmung der so bestimmten Zeitpunkte $\hat{t}_i$ und den tatsächlichen Zeitpunkten t_i, zu denen das Ego-Fahrzeug an den entsprechenden Verkehrsschildern vorbeigefahren ist, zu erreichen. Die Konstante „delay“ wurde bestimmt, indem das gesamte Skript „main_ts_detection_and_recognition“ für eine Vielzahl von „delay“-Werte ausgeführt und der Wert gewählt wurde, der den Betrag der Summe der Abweichungen $(\hat{t}_i - t_i)$([6]) minimiert[7].

Um schließlich den Zeitpunkten $\hat{t}_i$ die richtigen Ortspunkte entlang der Fahrstrecke zuzuordnen, wird zunächst das Bewegungsprofil des Ego-Fahrzeugs geladen und durch Interpolieren der in diesen Daten enthaltenen Ortspunkte an den relevanten Stellen die Positionen des Ego-Fahrzeugs zu den Zeitpunkten $\hat{t}_i$ approximiert.

Es folgt ein Abschnitt, der zur Evaluierung der Verkehrsschilddetektion, -klassifikation und der Zuordnung zu den richtigen Zeit- und Ortspunkten dient. Hierauf wird in 5.1 näher eingegangen. Schlussendlich werden die Ergebnisse auf die Weise, die in 3.3.3 beschrieben ist, in das in 2.4 vorgestellte Datenmodell integriert.

[6] Also die Größe $|\sum_i (\hat{t}_i - t_i)|$

[7] Die geschah, indem das Skript zuerst in eine Funktion umgewandelt und unter dem Namen „main_ts_detection_and_recognition_as_fct“ gespeichert wurde und anschließend aus dem Skript „run_main_ts_detection_and_recognition_as_fct“ heraus über eine Schleife mit einer Vielzahl von „delay“-Werten ausgeführt wurde.

Kapitel 5

Auswertung

5.1 Beschreibung der Validierungs- und Testkriterien und -prozesse und Validierungsergebnisse

5.1.1 Die mittlere absolute Abweichung (MAE)

Zur Beschreibung der Schätzgenauigkeit der Verkehrsobjekt-Größenbestimmung sowie der Verkehrsschild-Positionsbestimmung wurde die Metrik der mittleren absoluten Abweichung (MAE) gewählt. Diese ist wie folgt definiert:

$$MAE = \frac{1}{n} \sum_{j=1}^{n} | \, y_j - \hat{y_j} \, | \tag{5.1}$$

Als gängige Alternative stünde eine weitere Metrik, der root mean square error (RMSE, engl., auf Deutsch „Quadratwurzel der mittleren quadratischen Abweichung“), zur Verfügung, welcher wie folgt definiert ist:

$$RMSE = \sqrt{\frac{1}{n} \sum_{j=1}^{n} (y_j - \hat{y_j})^2}$$

Für die vorliegende Arbeit wurde die mittlere absolute Abweichung aus den folgenden zwei Gründe gewählt:

1. Interpretierbarkeit: Die mittlere absolute Abweichung lässt sich auf direkte und intuitive Weise interpretieren: Eine mittlere absolute Abweichung von a ($a \in \mathbb{R}, a \geq 0$) bedeutet schlicht, dass die Schätz- oder Berechnungsresultate $\hat{y_j}$ durchschnittlich um a von den wahren Werten y_j entfernt liegen. Dies ist beim RMSE nicht der Fall.

2. Gewichtung: RMSE gewichtet durch die vor der Summierung stattfindenden Quadrierungen betraglich höhere Abweichungen $|y_j - \hat{y_j}|$ stärker als betraglich niedrige. MAE hingegen gewichtet alle Abweichungen gleich. Der Autor geht für diese Arbeit davon aus, dass für zwei Abweichungen a_1 und a_2 mit $|a_2| = |b * a_1|$, $(b \in \mathbb{R}, b > 1)$ a_2 als genau „b mal so schlimm“ zu bewerten ist wie a_1 und somit eine im Betrag der Abweichungen steigende Gewichtung, wie es beim RMSE der Fall wäre, nicht angebracht wäre. Deshalb wird die mittlere absolute Abweichung auch unter Betrachtung dieses Aspektes dem RMSE vorgezogen.

5.1.2 Validierungs- und Testvorgehen und Ergebnisse

Im Folgenden wird beschrieben, auf welche Weise die einzelnen Komponenten der im vorherigen Kapitel vorgestellten Implementierungen auf ihre Leistungsfähigkeit überprüft wurden.

5.1.2.1 Verkehrsobjekt-Größenerkennung

Für den ACF-Detektor, der zur Verkehrsobjekt-Detektion verwendet wurde, fand keine gesonderte Validierung statt. Es wurde jedoch die Gesamtleistung der Verkehrsobjekt-Größenerkennung überprüft. Dies wird im Folgenden beschrieben:

Als Validierungsdatensatz für die gesamte Verkehrsobjekt-Größenerkennung dient das Fahrszenario, das durch die CarMaker-Datei „Ground_Truth_Highway_two_lane_traffic_road_signs.dat“ definiert ist. Die wahren Abmessungen der Verkehrsobjekte werden während der Ausführung des Matlab-Skripts „main_application_model_3_4“ beziehungsweise „main_application_model_2“, falls die Variable „testing“ vom Benutzer mit dem Wert 1 belegt wurde, aus dieser CarMaker-Datei automatisch ausgelesen und mit den berechneten Abmessungen verglichen. Hierzu wird die mittlere absolute Abweichung (5.1) für die folgenden Kategorien einzeln berechnet:

1. Breiten bestimmt mit Hilfe der nach vorne gerichteten Kamera,
2. Höhen bestimmt mit Hilfe der nach vorne gerichteten Kamera,
3. Breiten bestimmt mit Hilfe der nach hinten gerichteten Kamera,
4. Höhen bestimmt mit Hilfe der nach hinten gerichteten Kamera,
5. Breiten gesamt und
6. Höhen gesamt.

Die Ergebnisse dieser Berechnungen sind für die Implementierung, die nur einen Modellparameter f (siehe 3.3.1.1) verwendet, in Tabelle 5.1 und für die Implementierung, die zwei Modellparameter f_{Breite} und $f_{Höhe}$ (siehe 3.3.1.1) verwendet, in Tabelle 5.2 dargestellt.

	MAE Breite	MAE Höhe	Anzahl berechneter Breite-Höhe-Paare
nach vorne gerichtete Kamera	17,98 cm	12,93 cm	176
nach hinten gerichtete Kamera	12,60 cm	13,76 cm	68
Gesamt	16,48 cm	13,16 cm	244

Tabelle 5.1: Validierungsergebnisse der Verkehrsobjekt-Größenerkennung für die Implementierung des Modells mit einem Modellparameter f. Alle Werte wurden auf die zweite Nachkommastelle gerundet. Eigene Darstellung.

	MAE Breite	MAE Höhe	Anzahl berechneter Breite-Höhe-Paare
nach vorne gerichtete Kamera	18,11 cm	10,31 cm	176
nach hinten gerichtete Kamera	13,49 cm	13,45 cm	68
Gesamt	16,82 cm	11,19 cm	244

Tabelle 5.2: Validierungsergebnisse der Verkehrsobjekt-Größenerkennung für die Implementierung des Modells mit zwei Modellparametern f_{Breite} und $f_{Höhe}$. Alle Werte wurden auf die zweite Nachkommastelle gerundet. Eigene Darstellung.

Bei den bisher beschriebenen Validierungsergebnissen handelt es sich um Größen, die auf Basis eines einzelnen Szenarios berechnet wurden. Dieses wurde während der Implementierung der in dieser Arbeit beschriebenen Funktionen fortlaufend zur Validierung der aktuellen Performanz hinzugezogen. Aus diesem Grund wurde noch ein weiteres, zusätzliches Szenario erstellt, welches auf die finale Version der Implementierung angewandt wurde. Nach diesem finalen Test wurden keine weiteren Änderungen am vorliegenden Code vorgenommen. Dieses weitere Szenario ist durch die CarMaker-Datei „scenario_test_dimension_estimation.dat“ definiert. Das Szenario ist eine vom Autor abgeänderte Version des in CarMaker 8.0 enthaltenen Beispielszenarios „Cruising_3lanes“. Dieses Beispielszenario wurde für diesen Zweck so angepasst, dass in ihm mehrere Verkehrsobjekte links und rechts am Ego-Fahrzeug vorbeifahren. Eine Szene aus diesem Testszenario ist in Abbildung 5.1 dargestellt. Die hierdurch entstandenen finalen Testergebnisse sind in Tabelle 5.3 für die Implementierung des Modells, in dem nur ein Modellparameter f verwendet wurde und in Tabelle 5.4 für die Implementierung des Modells, in dem zwei Modellparameter f_{Breite} und $f_{Höhe}$ verwendet wurden, dargestellt.

5.1.2.2 Verkehrsschilderkennung

Zunächst wurde eine isolierte Validierung des ACF-Detektors, welcher zur Verkehrsschilddetektion Verwendung fand und dessen Training in 4.3.2.1 beschrieben ist, durchgeführt. Diese wird im Folgenden beschrieben:

Die Überprüfung der Performanz des verwendeten ACF-Detektors wurde für diese Arbeit auf eine visuelle Überprüfung anhand von in IPGMovie erstellten

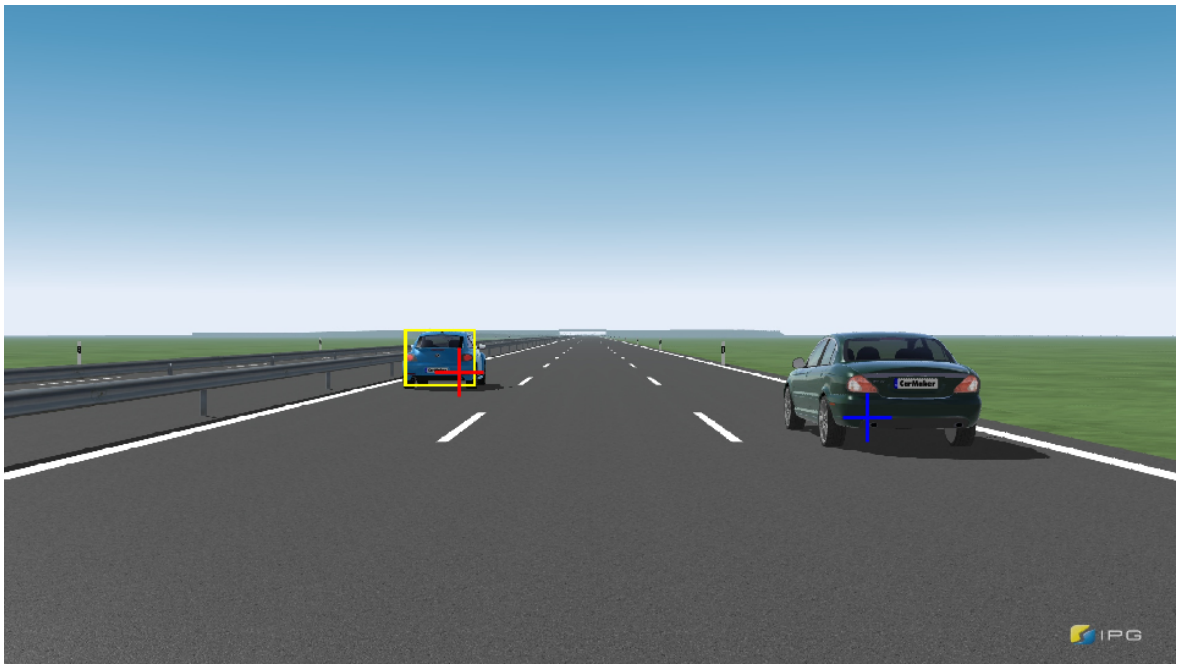

Abbildung 5.1: Szene aus dem zum finalen Testen der Größenerkennung verwendeten Szenario. Zu sehen sind die in diesem Arbeitsschritt verarbeitete Bounding-Box (in gelb) und die berechneten Positionen der zwei sichtbaren Fahrzeuge im Bild (in rot und blau), wobei die berechnete Position des Fahrzeugs, welches der Bounding-Box zugeordnet wurde, in rot hervorgehoben ist. Eigene Darstellung; erstellt mit IPGMovie und Matlab.

Bildern beschränkt. Zu diesem Zweck wurde der Detektor auf das durch die CarMaker-Dateien „Ground_Truth_Highway_two_lane_traffic_road_signs.dat“ und „Simple_Highway_circle_road_signs.rd5“ definierte Fahrszenario angewandt und die resultierenden Bounding-Boxen in den entsprechenden Bildern visualisiert. Ein Verkehrsschild gilt als zufriedenstellend detektiert, wenn in mindestens vier von fünf aufeinanderfolgenden Bildern eine Bounding-Box um dieses Schild erstellt wurde, da dies Voraussetzung dafür ist, dass die vorliegende Implementierung ein solches Verkehrsschild weiterverarbeitet (siehe 4.3.2.3). Das genannte Szenario enthält 38 Verkehrsschilder. Davon wurden 32 nach obiger Definition zufriedenstellend detektiert. Die Detektionsrate ergibt sich somit als der Quotient aus der Anzahl zufriedenstellend detektierter Verkehrsschilder und der Gesamtanzahl an Verkehrsschildern zu $\frac{32}{38} = 84,21\ \%$.

Auch die Leistung des CNNs, welches zur Verkehrsschildklassifikation verwendet wird und dessen Training in 4.3.2.2 beschrieben ist, wurde isoliert überprüft:

Mittels eines vom Trainings- und Validierungsdatensatz unabhängigen Testdatensatzes, welcher 12630 reale Kamerabilder enthält, die alle jeweils ein Verkehrsschild zeigen, wurde die Erkennungsrate des CNNs als der Quotient der Anzahl der korrekten Klassifikationen und der Gesamtanzahl an Klassifikationen bestimmt. Die so ermittelte Rate der korrekten Verkehrsschildklassifikationen beträgt 95,68 %. Dieser Wert wurde auf die zweite Nachkommastelle gerundet.

Des Weiteren wurde mittels einer manuell erstellten Tabelle, welche alle Bildindi-

	MAE Breite	MAE Höhe	Anzahl berechneter Breite-Höhe-Paare
nach vorne gerichtete Kamera	18,15 cm	12,75 cm	21
nach hinten gerichtete Kamera	13,68 cm	14,83 cm	5
Gesamt	17,29 cm	13,15 cm	26

Tabelle 5.3: Finale Testergebnisse der Verkehrsobjekt-Größenerkennung für die Implementierung des Modells mit einem Modellparameter f. Alle Werte wurden auf die zweite Nachkommastelle gerundet. Eigene Darstellung.

	MAE Breite	MAE Höhe	Anzahl berechneter Breite-Höhe-Paare
nach vorne gerichtete Kamera	15,96 cm	11,44 cm	21
nach hinten gerichtete Kamera	12,22 cm	10,65 cm	5
Gesamt	15,24 cm	11,29 cm	26

Tabelle 5.4: Finale Testergebnisse der Verkehrsobjekt-Größenerkennung für die Implementierung des Modells mit zwei Modellparametern f_{Breite} und $f_{Höhe}$. Alle Werte wurden auf die zweite Nachkommastelle gerundet. Eigene Darstellung.

zes der durch Simulation des „Ground_Truth_Highway_two_lane_traffic_road_signs“-Szenarios gewonnenen Bilder und die Identifikationen der Verkehrsschilder, die in den entsprechenden Bildern zu sehen sind, enthält[1], sowie mit den Resultaten aller während der Ausführung der Verkehrsschild-Größenerkennung durchgeführten Klassifikationen die Rate der korrekten Verkehrsschildklassifikationen in der betrachteten Anwendung berechnet. Diese ergibt sich als Quotient aller korrekten Klassifikationen und aller durchgeführten Klassifikationen zu 81,46 %. Dieser Wert ist auf die zweite Nachkommastelle gerundet.

Im Folgenden wird die Validierung der gesamten Verkehrsschilderkennung beschrieben:

Als Validierungsdatensatz für die Bestimmung der Gesamt-Performanz der Verkehrsschilderkennung dient ebenfalls das durch die CarMaker-Dateien „Ground_Truth_Highway_two_lane_traffic_road_signs.dat“ und „Simple_Highway_circle_road_signs.rd5“ definierte Fahrszenario. Die in diesem Szenario vorhandene Umgebung der Fahrbahn besteht aus der die Straße umgebenden grün eingefärbten Ebene in einigen Teilen des Szenarios und aus Wald in den anderen Teilen. Die wahren Positionen der Verkehrsschilder wurden mithilfe des Skriptes „ts_detection_and_recognition_build_ground_truth_array“, welches zu diesem Zweck erstellt wurde, automatisch ausgelesen. Die so erstellte Tabelle wird während der Ausführung des Skriptes „main_ts_detection_and_recognition“ zur Validierung der Gesamt-Performanz desselben herangezogen.

Um die Genauigkeit der Verkehrsschilderkennung zu quantifizieren wird zunächst ein To-

[1]Die hierfür manuell erstellte Ground Truth-Tabelle trägt den Namen „ground_truth_all_classifications.xlsx“

leranzwert *tol* definiert, welcher eine Oberschranke für den Abstand eines erkannten Verkehrsschildes von seiner wahren Position darstellt. Wird ein Verkehrsschild im Umkreis mit Radius *tol* um den Punkt lokalisiert, an dem sich dieses Verkehrsschild tatsächlich befindet, wird dieses Schild als korrekt erkannt markiert. Die Erkennungsrate wird als Quotient aus der Anzahl der nach dem beschriebenen Schema als korrekt erkannt markierten Verkehrsschilder und der Gesamtanzahl der zu erkennenden Verkehrsschilder berechnet. Für die vorliegende Implementierung wurde ein Toleranzwert von $tol = 15\ [m]$ gewählt.

Schließlich wird für die Verkehrsschilder, die korrekt erkannt wurden, die mittlere absolute Abweichung (5.1) von den wahren Werten für zwei Kategorien einzeln berechnet:

1. Für die Zeitpunkte, zu denen das Ego-Fahrzeug an den Verkehrsschildern vorbeigefahren ist und
2. für die Positionen der Verkehrsschilder entlang der Fahrstrecke.

Für die Verkehrsschilderkennung insgesamt gibt es also drei Validierungskriterien:

1. Die Erkennungsrate: Das „Ground_Truth_Highway_two_lane_traffic_road_signs“-Szenario enthält 38 zu erkennende Verkehrsschilder, von denen 30 nach dem oben beschriebenen Schema als korrekt erkannt markiert werden. Die Erkennungsrate liegt für dieses Szenario also bei $\frac{30}{38} = 78,95\ \%$.
2. Die mittlere absolute Abweichung der berechneten Zeitpunkte zu den tatsächlichen Zeitpunkten, zu denen das Ego-Fahrzeug am entsprechenden Verkehrsschild vorbeifährt, welche 0.13 s beträgt
3. und die mittlere absolute Abweichung der Verkehrsschildpositionen zu den wahren Positionen, welche 4.58 m beträgt.

Diese drei Werte wurden auf die zweite Nachkommastelle gerundet.

Bei den bisher beschriebenen Validierungsergebnissen handelt es sich um Größen, die auf Basis eines einzelnen Szenarios berechnet wurden. Dieses enthält nur 38 zu erkennende Verkehrsschilder und wurde außerdem während der Implementierung der in dieser Arbeit beschriebenen Funktionen fortlaufend zur Validierung der aktuellen Leistung hinzugezogen. Aus diesen zwei Gründen wurde noch ein weiteres, zusätzliches und wesentlich größeres Szenario erstellt, welches auf die finale Version der Implementierung angewandt wurde. Nach diesem finalen Test wurden keine weiteren Änderungen am vorliegenden Code vorgenommen.

Dieses weitere Szenario ist durch die Dateien „Larger_scenario_road_signs.dat“ und „road_larger_scenario_road_signs.rd5“ definiert und enthält 114 zu erkennende Verkehrsschilder sowie drei verschiedene Abschnitte, die sich durch ihre Szenerie unterscheiden: Im ersten Abschnitt besteht die Szenerie lediglich aus der die Straße umgebenden grün eingefärbten Ebene. Im zweiten Abschnitt ist die Straße von Bäumen umgeben und im dritten Abschnitt von Hochhäusern. Die drei Abschnitte enthalten jeweils dieselben 38 Verkehrsschilder wie das weiter oben beschriebene Szenario. Die Reihenfolge dieser Verkehrsschilder ist je Abschnitt unterschiedlich und zufällig gewählt. Jeweils die Hälfte

der Verkehrsschilder befindet sich auf der rechten Seite der Fahrbahn, die andere Hälfte auf der linken Seite und somit wesentlich weiter entfernt von der Trajektorie des Ego-Fahrzeugs, welches auf der rechten Fahrspur fährt. In Abbildung 5.2 sind vier aus diesem

Abbildung 5.2: Beispielhafte Darstellung einiger aus dem Testszenario resultierender Kamerabilder mit eingeblendeten Bounding-Boxen und Klassifikationsergebnissen. Oben links: Szenarioteil ohne Bäume oder Gebäude; oben rechts: Szenarioteil mit Bäumen; unten links: Szenarioteil mit Gebäuden; unten rechts: Fehlgeschlagene Detektion eines Verkehrsschildes im Szenarioteil mit Gebäuden. Die Original-Bounding-Boxen sind in gelb, die vergrößerten und für die Weiterverarbeitung verwendeten in blau dargestellt. Eigene Darstellung; erstellt mit IPGMovie und Matlab.

Szenario resultierende Kamerabilder mit eingeblendeten Bounding-Boxen und Klassifikationsergebnissen dargestellt.

Für dieses Szenario wurden lediglich die Größen zur Validierung der Verkehrsschilderkennung insgesamt berechnet[2]. Die Gesamt-Erkennungsrate beträgt hier 74,56 %. Die mittlere absolute Abweichung der berechneten Zeitpunkte zu den tatsächlichen Zeitpunkten, zu denen das Ego-Fahrzeug am entsprechenden Verkehrsschild vorbeifährt, beträgt 0.14 s. Und die mittlere absolute Abweichung der Verkehrsschildpositionen zu den wahren Positionen beträgt 3.87 m.

Zusätzlich wurde bestimmt, auf welche Weise sich die Gesamt-Erkennungsrate von 74,56 % aus den Erkennungsraten der einzelnen drei beschriebenen Szenarioabschnitten ergibt.

[2]Auf die oben beschriebene isolierte Betrachtung der Klassifikationsperformanz wurde hier verzichtet, da andere Aspekte dieser Arbeit höher priorisiert wurden als das aufwändige Erstellen der entsprechenden Ground Truth-Tabelle - ein Prozess, der bisher nicht automatisiert wurde. Auf die visuelle Überprüfung der Detektionsperformanz wurde hier aus dem gleichen Grund verzichtet.

Diese Analyse ergab, dass die Erkennungsrate für den ersten, nicht bewaldeten oder bebauten Abschnitt 76,32 % beträgt. Für den zweiten, mit Bäumen versehenen Abschnitt ist sie ebenfalls 76,32 %. Und für den dritten, mit Hochhäusern versehenen Abschnitt beträgt sie 71,05 %. Alle genannten Werte wurden auf die zweite Nachkommastelle gerundet.

5.2 Diskussion der Validierungs- und Testergebnisse

Der folgende Abschnitt enthält eine Diskussion der Ergebnisse der Validierungen und der finalen Tests, die im vorherigen Abschnitt präsentiert wurden.

5.2.1 Verkehrsobjekt-Größenerkennung

Vergleicht man die Validierungsgrößen, die in den Tabellen 5.1 und 5.2 dargestellt sind, stellt man fest, dass das Resultat der Implementierung, die zwei Modellparameter f_{Breite} und $f_{Höhe}$ verwendet, nur in einem der zwei „Gesamt"-Größen bessere Werte aufweist als das Resultat der Implementierung, die nur einen Modellparameter f verwendet. Es kann also nicht von einer Verbesserung der Modellgüte durch das Hinzufügen eines weiteren Freiheitsgrades gesprochen werden, wie in 3.3.1.1 zunächst vermutet wurde. Es kann also gefolgert werden, dass für die weitere Verwendung die Implementierung mit nur einem Modellparameter f zu empfehlen ist, da sie eine ähnliche Modellgüte wie die Implementierung mit zwei Parametern aufweist, jedoch den tatsächlich vorliegenden mathematischen Zusammenhang wesentlich realistischer abbildet.

Vergleicht man die finalen Testergebnisse, die in den Tabellen 5.3 und 5.4 dargestellt sind, mit den Validierungsergebnissen in den Tabellen 5.1 und 5.2, so stellt man fest, dass diese sich nicht stark voneinander unterscheiden und die Validierungsergebnisse somit als bestätigt angesehen werden können. Zu bemerken ist allerdings, dass bei alleiniger Betrachtung der finalen Testergebnisse festgestellt werden kann, dass die Implementierung des Modells mit zwei Modellparametern f_{Breite} und $f_{Höhe}$ strikt bessere Testergebnisse liefert als die Implementierung des Modells mit einem Modellparameter f. Da die Verbesserung jedoch nicht sehr groß ist und außerdem die Anzahl berechneter Breite-Höhe-Paare für das Testszenario im Vergleich zum Validierungsszenario wesentlich kleiner ist, hält der Autor dennoch an der weiter oben formulierten Empfehlung fest, für die weitere Verwendung das Modell mit einem Parameter zu wählen, da es den wahren mathematischen Zusammenhang wesentlich realistischer abbildet.

Bei Betrachtung dieser Ergebnisse kann insgesamt festgestellt werden, dass die im Rahmen dieser Arbeit implementierte Größenerkennung zufriedenstellend funktioniert.

5.2.2 Verkehrsschilderkennung

Aus der Betrachtung der Detektionsrate von $\frac{32}{38} = 84,21$ % und der Gesamt-Erkennungsrate von $\frac{30}{38} = 78,95$ % (siehe 5.1.2.2) wird ersichtlich, dass der Großteil der

fehlgeschlagenen Erkennungen auf eine mangelhafte Detektion zurückzuführen sind: Bei sechs der acht nicht erkannten Verkehrsschilder schlägt schon die Detektion fehl. Verkehrsschilder die vermehrt zu Problemen bei der Detektion führen sind die Schilder, die sich farblich stark vom Großteil der Verkehrsschilder unterscheiden. Hierbei handelt es sich um die vier Schilder, die nur schwarz und weiß enthalten also das „Restriktionen enden“-Schild, das „Geschwindigkeitsberenzung 80 km/h endet“-Schild, das „Überholverbot endet“-Schild und das „Überholverbot für Lastwagen endet“-Schild sowie einige Schilder, die blau und weiß sind: Hier führen vor allem das „rechts oder geradeaus-“, das „links oder geradeaus“- und das „Links halten“-Schild zu schlechten Detektionsresultaten.

Lediglich zwei der acht nicht erkannten Verkehrsschilder wurden zufriedenstellend detektiert. Der Grund für die nicht erfolgreiche Erkennung liegt bei diesen zwei Verkehrsschildern also in einer mangelhaften Klassifikation oder einer nicht zielführenden Weiterverarbeitung in der auf die Klassifikation folgenden Programmlogik. Ein Verkehrsschild, das oft zu falschen Klassifikationen führt, ist das „Links halten“-Schild. Dieses wird meist als „Rechts halten“ klassifiziert. Den Grund hierfür vermutet der Autor darin, dass der verwendete Trainingsdatensatz im Vergleich zur Anzahl an Bildern mit „Links halten“-Schildern fast sieben mal so viele Trainingsbilder für „Rechts halten“ enthält. Des Weiteren ist ein Großteil der Bilder vom „Links halten“-Schild von geringer Qualität: Viele dieser Bilder sind verwackelt oder das Schild ist mit Aufklebern beklebt oder mit Sprühfarbe bemalt. Ähnlich verhält es sich mit dem „links abbiegen“-Schild: Dieses wird oft als „rechts abbiegen“ klassifiziert. Jedoch ist hier der Unterschied zwischen den Anzahlen an Trainingsbildern weniger hoch. Die Anzahl der Trainingsbilder für „rechts abbiegen“ ist etwa 1,6 mal so hoch wie die für „links abbiegen“.

In Abbildung 5.3 sind die Verkehrsschilder, die vermehrt zu Problemen bei der Detektion

Abbildung 5.3: Darstellung der Verkehrschilder, die bei denen oft die Detektion fehlschlägt (erste 7) oder bei denen es oft zu Fehlklassifikationen kommt (letzte zwei). Einzeldarstellungen übernommen aus dem verwendeten „German Traffic Sign Recognition Benchmark“-Datensatz, welcher durch Stallkamp et al. in [SSSI11] vorgestellt wird.

oder Klassifikation geführt haben, dargestellt.

Der Zweck der auf Detektion und Klassifikation folgenden Programmlogik ist das Verarbeiten richtiger und das Aussortieren falscher Klassifikationen[3]. Die Rate der korrekten Verkehrsschildklassifikationen ergibt sich, wie in 5.1.2.2 erläutert wird, zu 81,46 %. Nehmen wir kurz an es würden nach der Detektion und Klassifikation keine weiteren Verarbeitungsschritte mehr folgen. Dann könnte die erwartete Gesamt-Erkennungsrate als das Produkt der vorliegenden Detektionsrate und der Rate der korrekten Verkehrsschildklassifikationen mit $84,21\% * 81,46\% = 68,6\%$ ausgedrückt werden. Die vorliegende Gesamt-Erkennungsrate liegt mit 78,95 % jedoch mehr als zehn Prozentpunkte über diesem Wert, woraus gefolgert werden kann, dass die auf Detektion und Klassifikation folgende Programmlogik einen nicht unerheblichen Teil der irrtümlich erstellten Bounding-Boxen und falsch klassifizierten Verkehrsschilder erfolgreich als solche identifiziert und aussortiert.

Aus den Validierungsergebnissen der Verkehrsschilderkennung kann insgesamt gefolgert werden, dass es sowohl bei der Verkehrsschilddetektion, als auch bei der Verkehrsschildklassifikation noch Potenzial zur Verbesserung gibt:

Obwohl der Trainingsdatensatz für den Verkehrsschilddetektor Bilder von allen im betrachteten Szenario enthaltenen Verkehrsschildern enthält, funktioniert die Detektion von sechs Verkehrsschildern nicht zufriedenstellend. Dies könnte daran liegen, dass der Trainingsdatensatz eine nicht unerhebliche Unbalanciertheit zu Ungunsten ebendieser sechs Verkehrsschilder aufweist. Der Trainingsdatensatz enthält insgesamt 1213 Bilder, die 43 verschiedene Verkehrsschilder zeigen. Wäre die Anzahl an Bilder für jedes Verkehrsschild gleich, würde der Datensatz etwa 28 Bilder je Verkehrsschild enthalten. Die sechs Verkehrsschilder, die nicht zufriedenstellend detektiert wurden, sind jedoch mit nur 19, acht, zwei, sechs, sieben und elf Trainingsbildern jeweils unterrepräsentiert.

Dass die Rate der korrekten Verkehrsschildklassifikationen, die durch die Anwendung des verwendeten CNNs auf das „Ground_Truth_Highway_two_lane_traffic_road_signs“-Szenario bestimmt wurde, mit 81,46 % mehr als 14 Prozentpunkte unter der mit Realbildern bestimmten Rate der korrekten Verkehrsschildklassifikationen liegt ist ebenfalls bemerkenswert. Auch hier vermutet der Autor, wie weiter oben schon einmal kurz im Zusammenhang mit dem „links halten“-Verkehrsschild erwähnt wurde, dass dies an der Unbalanciertheit der verwendeten Datensätze liegen könnte. Ebenfalls ein möglicher Grund könnte sein, dass das verwendete CNN per Transfer Learning auf Basis eines mit Bildern von US-amerikanischen Verkehrsschildern trainierten CNNs trainiert wurde (siehe 4.3.2.2).

Der Autor vermutet somit, dass sich die aktuell bei 78,95 % liegende Gesamt-Erkennungsrate der Verkehrsschilderkennung durch geeignete Vergrößerung der verwendeten Trainingsdatensätze in den entsprechenden Kategorien und Verbesserungen am Trainingsprozess und an der Architektur des zur Klassifikation verwendeten CNNs wesentlich steigern ließe.

Durch den im letzten Absatz von 5.1.2.2 beschriebenen, anhand eines größeren Szenarios durchgeführten finalen Test bestätigen sich die vorher bestimmten mittleren absoluten Abweichungen sowohl im Bezug auf die Zeit, also auch im Bezug auf die Strecke. Für das

[3]Die Regeln, die hierbei zum Einsatz kommen, werden in 4.3.2.3 beschrieben.

Validierungsszenario betrugen diese 0.13 s und 4.58 m und für das für den finalen Test eingesetzte Szenario 0.14 s und 3.87 m.

Im Bezug auf die Gesamt-Erkennungsrate bestätigt sich das vorher bestimmte Validierungsergebnis lediglich für Szenerien, die keine Häuser enthalten: Für solche Szenerien liegt das Testergebnis mit 76,32 % nur leicht unter der für das Validierungsszenario bestimmten Gesamt-Erkennungsrate von 78,95 %. Für den Abschnitt, in dem die Fahrbahn von Hochhäusern umgeben ist, liegt die Test-Erkennungsrate jedoch mit 71,05 % deutlicher unter der Validierungs-Erkennungsrate des zuerst betrachteten Szenarios. Der Autor vermutet deshalb, dass eine optisch komplexere Umgebung, wie die im dritten Abschnitt des Testszenarios dargestellten Hochhäuser, die Verkehrsschilderkennung erschwert.

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

In der vorliegende Arbeit wurde die Erweiterung eines Datenmodells zur Übertragung realer Fahrzeugdaten in eine Simulationsumgebung beschrieben, mit dem Ziel, in dieses neben dynamischen Informationen auch Informationen von statischer Natur, wie zum Beispiel die Positionen von Verkehrsschildern oder die Anzahl der Fahrbahnspuren, einzubetten. Außerdem wurde die konkrete Entwicklung und Implementierung von Funktionen beschrieben, um aus Sensordaten Informationen über Verkehrsschilder und die Höhe und die Breite von beteiligten Fahrzeugen zu extrahieren und in das Modell zu übertragen. Des Weiteren wurde in dieser Arbeit der Zweck der Modellerweiterung und der Entwicklung und Implementierung der Funktionen erläutert sowie die zum Verständnis der Arbeit notwendigen Grundlagen beschrieben und eine Auswertung der Ergebnisse präsentiert.

Die finale Version der Erkennung von Höhen und Breiten von beteiligten Fahrzeugen besitzt aus Sicht des Autors eine zufriedenstellende Erkennungsqualität. Die Verkehrsschilderkennung bietet in ihrer zum Zeitpunkt der Abgabe bestehenden Version noch Potenzial zu Verbesserung. Hierauf wir im folgenden Abschnitt 6.2 näher eingegangen.

6.2 Ausblick

Wie in 2.5.1 beschrieben existieren neben der für die Verkehrsobjekt-Größenerkennung verwendeten konventionellen rechtlinigen Linse ohne Verzerrung noch weitere gängige Linsentypen. In zukünftigen Weiterführungen dieser Arbeit oder von Aspekten dieser Arbeit könnte somit die Implementierung so angepasst werden, dass sie mit weiteren Linsentypen kompatibel ist. Die hierfür relevanten Projektionsfunktionen sind in Tabelle 2.2 dargestellt.

Des Weiteren ist die dieser Arbeit beiliegende Implementierung nicht darauf ausgelegt, neben den Abmessungen von Personenkraftwagen auch die von Lastkraftwagen zu bestim-

men. Dies liegt daran, dass der verwendete ACF-Detektor (2.7) nur mithilfe von Bildern von Personenkraftwagen trainiert wurde. Dem Autor gelang es des Weiteren nicht, einen Trainingsdatensatz zu finden, der von ausreichender Qualität ist und genügend Bilder von Lastkraftwagen enthält. Das Erstellen eines solchen Datensatzes könnte somit Teil zukünftiger Anstrengungen sein, um diesen Aspekt der vorliegenden Arbeit zu erweitern und zu verbessern. Im Falle, dass ein solcher Trainingsdatensatz existiert und, dass ein verwendeter Detektor erfolgreich mit ihm trainiert würde, wäre es sinnvoll im Anschluss eine einfache Klassifikation zu implementieren, die anhand der geschätzten Höhen und Breiten die entsprechenden Fahrzeuge entweder der Klasse „PKW“ oder der Klasse „LKW“ zuordnet. Dies könnte durch einen einfachen Entscheidungsbaum der Tiefe eins geschehen.

Außerdem könnten sich zukünftigen Arbeiten damit beschäftigen, wie weitere Aspekte statischer Information wie zum Beispiel die Spurbreite, die Anzahl an Spuren und die Krümmung der Fahrbahn zum Datenmodell hinzugefügt werden können.

Bei der im Rahmen der vorliegenden Arbeit implementierten Verkehrsschilderkennung existiert aus der Sicht des Autors noch Potenzial zur Verbesserung: Der Autor vermutet, dass durch die Vergrößerung der verwendeten Trainingsdatensätze an geeigneten Stellen oder durch andere Maßnahmen, um der erheblichen Unbalanciertheit der Trainingsdatensätze entgegenzuwirken, sowohl die Verkehrsschilddetektion als auch die Verkehrsschildklassifikation erheblich verbessert werden können. Durch Verbesserungen an der Architektur des zur Klassifikation verwendeten CNNs könnte die Klassifikationsperformanz vermutlich ebenfalls verbessert werden. Auch wäre es aus Sicht des Autors interessant und lohnenswert, zu testen, inwieweit sich die Klassifikationsperformanz ändert, wenn das verwendete CNN mit anderen, eventuell zufällig gewählten Initialgewichten trainiert würde. Zur Erinnerung: Für die vorliegenden Implementierung wurden als Initialgewichte der unverändert übernommenen Schichten des Klassifikators die Gewichte des bereits mit Bildern von US-amerikanischen Verkehrsschildern trainierten CNNs verwendet, welches als Vorlage für den hier verwendeten Klassifikator diente (siehe 4.3.2.2).

Literaturverzeichnis

[Bet05] Felix Bettonvil. Fisheye lenses. *WGN, Journal of the International Meteor Organization*, 33:9–14, 01 2005. 2.5.1, 2.5.1, 2.2, A.2.3

[Bis06] Christopher Bishop. *Pattern recognition and machine learning*. Springer, 2006. 2.7.1, 2.8, 2.8.1, 2.8.1, 2.8.1, 2.7, 2.8.2, A.2.3

[DABP14] P. Dollar, R. Appel, S. Belongie, and P. Perona. Fast Feature Pyramids for Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(08):1532–1545, aug 2014. 2.7, 2.7.1, 2.7.2, 2.7.2, 5, 6, 7, 4.3.1.3

[Dix18] Richard Dixon. *Trends in der Automobil-Sensorik*, chapter 1, pages 17–28. T. Tille, The address of the publisher, 1 edition, 4 2018. 2.1.3, 2.1.3

[DTPB09] Piotr Dollar, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral Channel Features. In *Proceedings of the British Machine Vision Conference*, pages 91.1–91.11. BMVA Press, 2009. doi:10.5244/C.23.91. 2.7.1

[Eym19] G. Eymann. Automatisiertes Fahren: Sensortechniken im Check, 05 2019. [Abgerufen 17.02.2020]. 2.1.3, 2.1, 2.1.3, A.2.3

[FS97] Yoav Freund and Robert E Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997. 2.7.1

[GAA+12] T. M. Gasser, C. Arzt, M. Ayoubi, A. Bartels, L. Bürkle, J. Eier, F. Flemisch, D. Häcker, T. Hesse, and W. Huber et al. "Rechtsfolgen zunehmender Fahrzeugautomatisierung - Gemeinsamer Schlussbericht der Projektgruppe", Berichte der Bundesanstalt für Straßenwesen, Fahrzeugtechnik Heft F 83. 01 2012. 2.1.2

[HSS+13] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks (submitted)*, 2013. 4.3.2.1

[Kha19] Mohamed Amine Khalef. Entwicklung und Implementierung eines Konzeptes zur Metabeschreibung von Fahrszenarien mit Fokus auf die Interaktion mit anderen Verkehrsteilnehmern. Master's thesis, Karlsruher Institut für Technologie, 08 2019. 1.2, 2.4, 2.1, 2.4, 2.3, 2.4, 2.5, 4.3.1.3, 4.3.1.3, A.2.3, A.2.3

[Lam16] Peter Lambert. Statistisches Bundesamt: Verkehrsunfälle Zeitreihen 2014. Technical report, Statistisches Bundesamt, 7 2016. Wiesbaden, Forschungsbericht. 2.2

[LDG+16] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature Pyramid Networks for Object Detection. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2016. 2.7.1

[LM00] Huan Liu and Hiroshi Motoda. *Motoda, H.: Feature Selection for Knowledge Discovery and Data Mining. Kluwer Academic, USA*. 01 2000. 2.7.1

[Mat] Traffic Sign Detection and Recognition. https://de.mathworks.com/help/gpucoder/examples/code-generation-for-traffic-sign-detection-and-recognition-networks.html. Aufgerufen: 2020-03-26. 4.3.2.2, 4.3.2.2

[Res17] Andreas Reschka. *Fertigkeiten- und Fähigkeitengraphen als Grundlage des sicheren Betriebs von automatisierten Fahrzeugen im öffentlichen Straßenverkehr in städtischer Umgebung*. PhD thesis, Jul 2017. 2.3

[Sch17] Fabian Schuldt. *Ein Beitrag für den methodischen Test von automatisierten Fahrfunktionen mit Hilfe von virtuellen Umgebungen*. PhD thesis, Apr 2017. 2.2, 1, 2.3

[SSSI11] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460, 2011. 4.3.2.2, 5.3, A.2.3

[UMR+15] Simon Ulbrich, Till Menzel, Andreas Reschka, Fabian Schuldt, and Markus Maurer. Definition der Begriffe Szene, Situation und Szenario für das automatisierte Fahren. 09 2015. 2.3, 2.3, 2.2, A.2.3

[Ver15] Verband der Automobilindustrie e. V. Automatisierung: Von Fahrerassistenzsystemen zum automatisierten Fahren, 09 2015. 2.1.1, 2.1.2

[WW15] Walther Wachenfeld and Hermann Winner. *Die Freigabe des autonomen Fahrens*, pages 440–463. 01 2015. 2.2

Anhang A

Detaillierte Beschreibung der verschiedenen Ansätze zur Modellerweiterung

Wie in 3.3.3 angegeben, enthält dieser Abschnitt eine detaillierte Beschreibung der in 3.3.3 kurz beschriebenen Ansätze und Optionen. Hierbei sind zwei Modellerweiterung zu unterscheiden:

1. Erweiterung A: Erweiterung der Ego- und Objektmatrizen um jeweils zwei Zeilen, die ihre Breite und Höhe enthalten
2. Erweiterung B: Erweiterung des Tensors mit dem Ziel statische Objekte und statische Informationen wie z.B. Verkehrsschilder oder die aktuelle Anzahl der Spuren in demselben abzubilden

Im Folgenden wird nur noch Erweiterung B am Beispiel des Erkennens von Verkehrsschildern und der Verarbeitung und Speicherung diesbezüglicher Informationen betrachtet. Für Erweiterung B gibt es zwei Ansätze, wobei Ansatz 2 nochmal in drei Optionen unterteilt ist. Ansatz 2, Option 3 wurde als die Option ausgewählt, die implementiert wurde.

A.1 Erster Ansatz

- Es wird eine neue Matrix erstellt
- 1. Zeile: Alle Zeitpunkte wie sie aus CarMaker kommen (äquidistant, z.B. 50 Zeitpunkte pro Sekunde)
- 2. Zeile: Die Position des Ego-Fahrzeugs in Längsrichtung zu diesem Zeitpunkt (folglich nicht äquidistant)
- 3. bis n-te Zeile: Variablen wie "aktive Geschwindigkeitsbegrenzung", "Überholverbot für Lastwagen ja/nein", etc.

- Vorteile gegenüber einer Speicherung in der Form [Zeile 1: Weg (äquidistant); Zeile 2: Zeit (nicht äquidistant)]:
  - Daten aus CarMaker werden zeitbasiert (zeitlich-äquidistant) aufgezeichnet und können somit direkt verwendet werden.
  - Falls man nur einen bestimmten Zeitabschnitt betrachtet (aus dem Tensor also eine "Scheibe" herausschneiden möchte) existieren die "Schnittzeitpunkte" exakt genauso in der neuen Matrix.
  - Sehr spitzfindiger Vorteil: $f_1 : t \to s$ ist eine Abbildung (unmöglich in der Zeit zurück zu reisen), $f_2 : s \to t$ ist keine Abbildung (außer wir definieren, dass rückwärts fahren nicht erlaubt ist). Uneindeutigkeit ist somit ausgeschlossen (auch ohne die Annahme zu treffen, dass das Ego-Fahrzeug nur vorwärts fährt).
- Informationsgehalt:
  - Belegung der statischen Variablen wird für jeden Zeitpunkt und zugehörigen Ortspunkt festgehalten.
  - Es ist für jeden Zeitpunkt sofort ersichtlich, welche Verkehrsschilder, Spuranzahl, etc. für das Ego-Fahrzeug vorliegen.
  - Es ist für jeden Ortspunkt sofort ersichtlich, welche Verkehrsschilder, Spuranzahl, etc. vorliegen.
- Vorteil gegenüber dem zweiten Ansatz (siehe unten): Kontinuierliche Größen wie z.B. die Spurbreite oder die Fahrbahnkrümmung können ohne Informationsverlust festgehalten werden.
- Nachteil gegenüber dem zweiten Ansatz: Da alle Zeitpunkte bzw. Wegpunkte gespeichert werden, verfehlt dieser Ansatz das Ziel das Fahrszenario in abstrahierter Form darzustellen.

A.2 Zweiter Ansatz

- Aus Sicht des Ego-Fahrzeugs: Speicherung der Orts- und Zeitpunkte von statischen Objekten bzw. Veränderungen statischer Informationen; ein statisches Objekt ist z.B. ein Verkehrsschild, eine Veränderung statischer Information ist zum Beispiel das Hinzukommen einer weiteren Fahrbahnspur. Solche Veränderungspunkte werden im Folgenden zur Vereinfachung "statisches Objekt" genannt. -
- Kontinuierliche Größen wie die Fahrbahnkrümmung oder -breite können in Zustände kodiert werden:
  - Bspw.: Zustand 0 $\iff$ Fahrbahnkrümmung $\in$ [a, b), Zustand 1 $\iff$ Fahrbahnkrümmung $\in$ [c, d), ...
  - Da dadurch jedoch Informationen verloren gehen, ist der erste, die ganze Strecke abdeckende Ansatz (siehe oben) für die Speicherung solcher Größen geeigneter.

Im Folgenden werden drei Optionen für den zweiten Ansatz beschrieben:

A.2.1 Zweiter Ansatz, Option 1

Weitere Matrix zum Tensor hinzufügen und alle Matrizen so erweitern, dass sie die selben Dimensionen haben um ursprüngliche Tensorstruktur zu erhalten:

- Neue Matrix erstellen von gleicher Gestalt wie die des Ego-Fahrzeugs und der Verkehrsobjekte
- Es werden zu allen Matrizen am unteren Ende neue Variablen hinzugefügt: s_Ego, type, value
- Alle Matrizen werden so erweitert, dass sie alle "alten" Spalten bzw. Zeitpunkte aus den Ego- und Verkehrsobjektmatrizen enthalten und zusätzlich die Zeitpunkte enthalten, zu denen das Ego-Fahrzeug an statischen Objekten vorbeigefahren ist.
- Variablenbelegung der neuen Matrix:
  - "Alte" Variablen: -9999 überall
  - "Neue" Variablen: In Spalten, die mit Zeitpunkten korrespondieren, an denen das Ego-Fahrzeug an statischem Objekt vorbeigefahren ist, mit Werten belegt. Sonst -9999
- Variablenbelegung der Matrizen des Ego-Fahrzeugs und der Verkehrsobjekte:
  - "Alte" Variablen: Keine Veränderung, aber in neu hinzugekommenen Spalten -9999
  - "Neue" Variablen: -9999 überall
- Ganz links enthalten die Matrizen einige Spalten, die die Initialbelegungen der Variablen definiert; bspw. [t = 0; ...; s_Ego = 0; type = „Verkehrsschild“; value = „Geschwindigkeitsbegrenzung 120“] und [t = 0; s_Ego = 0; type = „Spuranzahl“; value = 3]
- Alle im Modell enthalten Matrizen enthalten dieselben Variablen und Zeitpunkte und somit die gleichen Dimensionen. $\Longrightarrow$ Modell wurde um eine Matrix erweitert und jede Matrix hat Zeilen und Spalten hinzugewonnen.

Vorteile:

- Gegenüber Ansatz 1 und Ansatz 2, Option 3:
  - Alle Matrizen haben dieselben Dimensionen.
  - Bisherige Tensorstruktur bleibt erhalten.
- Gegenüber Ansatz 1: Es findet eine Abstraktion der Szenarioinformationen statt.

Nachteil gegenüber Ansatz 1: Kontinuierliche Größen nur als Zustände kodiert speicherbar

A.2.2 Zweiter Ansatz, Option 2

Statische Informationen in die Matrix des Ego-Fahrzeugs schreiben und alle Matrizen so erweitern, dass sie die selben Dimensionen haben um ursprüngliche Tensorstruktur zu erhalten:

- Im Prinzip werden alle Informationen, die in Option 1 in eine separate Matrix geschrieben werden, in Option 2 an die entsprechenden Stellen in die Matrix des Ego-Fahrzeugs geschrieben. Genauer:
- Es wird keine neue Matrix erstellt.
- Es werden zu allen Matrizen am unteren Ende neue Variablen hinzugefügt: s_Ego, type, value
- Alle Matrizen werden so erweitert, dass sie alle "alten" Spalten bzw. Zeitpunkte enthalten und zusätzlich die Zeitpunkte enthalten, zu denen das Ego-Fahrzeug an statischen Objekten vorbeigefahren ist.
- Variablenbelegung in Matrix des Ego-Fahrzeugs:
  - "Alte" Variablen: Keine Veränderung, aber bei neu hinzugekommenen Spalten -9999
  - "Neue" Variablen: In neu hinzugekommenen Spalten Informationen über das entsprechende statischen Objekt. In alten Spalten -9999
- Variablenbelegung in Matrizen der Verkehrsobjekte:
  - "Alte" Variablen: Keine Veränderung, aber in neu hinzugekommenen Spalten -9999
  - "Neue" Variablen: -9999 überall
- Ganz links enthalten die Matrizen einige Spalten, die die Initialbelegungen der Variablen definiert; bspw. [t = 0; ...; s_Ego = 0; type = „Verkehrsschild“; value = „Geschwindigkeitsbegrenzung 120“] und [t = 0; s_Ego = 0; type = „Spuranzahl“; value = 3]
- Alle im Modell enthalten Matrizen enthalten dieselben Variablen und Zeitpunkte und haben somit die gleichen Dimensionen. $\Longrightarrow$ Jede Matrix hat Zeilen und Spalten hinzugewonnen. Es wurde keine weitere Matrix erstellt. Der Informationsgehalt ist jedoch der selbe wie bei Option 1.

Vorteil gegenüber Option 1:

- Die entsprechenden Zeilen und Spalten in der Matrix des Ego-Fahrzeugs sind in Option 1 leer (mit -9999 belegt). Die leeren Felder zu füllen und eine Matrix weniger im Tensor zu speichern benötigt weniger Speicherplatz.

- Die Informationen über die statischen Objekte werden aus Sicht des Ego-Fahrzeugs gesammelt. Sie auch in der Matrix des Ego-Fahrzeugs zu speichern wirkt insgesamt logischer.

Vorteile:

- Gegenüber Ansatz 1 und Ansatz 2, Option 3:
  - Alle Matrizen haben dieselben Dimensionen.
  - Bisherige Tensorstruktur bleibt erhalten.
- Gegenüber Ansatz 1: Es findet eine Abstraktion der Szenarioinformationen statt.

Nachteile:

- Viele Felder, die -9999, also keine Information enthalten.
- Gegenüber Ansatz 1: Kontinuierliche Größen nur als Zustände kodiert speicherbar

A.2.3 Zweiter Ansatz, Option 3

Weitere Matrix zum Modell hinzufügen, die die Ortsdimension darstellt und nur Punkte abbildet die mit Änderungen statischer Informationen korrespondieren:

- Matrizen des Ego-Fahrzeugs und der Verkehrsobjekte werden so gelassen wie sie sind.
- Neue Matrix wird erstellt.
- In dieser werden die Positionen von statischen Objekten und Veränderungen statischer Information mit dem zugehörigen Zeitpunkt, dem Typ der Variable, die neu belegt wird, und ihrem neuen Wert festgehalten.
- 1. Zeile (s): Positionen von Änderungen statischer Informationen entlang der Fahrbahn; bspw. Position eines Verkehrsschildes oder eines Punktes, an dem sich die Spuranzahl von drei Spuren auf vier erhöht
- 2. Zeile (t_Ego): Der Zeitpunkt, an dem das Ego-Fahrzeug durch diesen Ortspunkt gefahren ist
- 3. Zeile (type): Art der statischen Information; bspw. „Verkehrsschild“ oder „Spuranzahl“
- 4. Zeile (value): Wert der statischen Information; bspw. im Falle von type = „Verkehrsschild“: „Geschwindigkeitsbegrenzung 100" oder im Falle von type = „Spuranzahl“: „4“

- Ganz links enthält die Matrix einige Spalten, die die Initialbelegungen der Variablen definiert; bspw. [s = 0; t_Ego = 0; type = „Verkehrsschild“; value = „Geschwindigkeitsbegrenzung 120“] und [s = 0; t_Ego = 0; type = „Spuranzahl“; value = 3]

Vorteile:

- Im Gegensatz zu Ansatz 1 findet eine Abstraktion der Szenarioinformationen statt.
- Im Vergleich zu Ansatz 2, Option 1 und 2 werden wesentlich weniger neue Zeilen und Spalten erstellt.

Nachteil gegenüber Ansatz 1: Kontinuierliche Größen nur als Zustände kodiert speicherbar

Tabellenverzeichnis

Abbildungsverzeichnis