BERGISCHE
UNIVERSITÄT
WUPPERTAL

Fakultät für Maschinenbau und Sicherheitstechnik

Masterthesis

im Studiengang **Qualitätsingenieurwesen**

beim Fachgebiet für **Verkehrssicherheit und Zuverlässigkeit**

zur Erreichung des akademischen Grades

Master of Science

Thema:	**Simulationsbasierte Fehlerinjektion von Perzeptionssensoren in autonomen Fahrzeugen**
Autor*in:	**Aaron Blickle**
MatNr:	1434453
Bearbeitungszeitraum:	**22. Januar 2024** bis **3. Juni 2024**
Erstprüfer*in:	Jun.-Prof. Dr. Antoine Tordeux
Zweitprüfer*in:	Dr.-Ing Fabian Plinke

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die von mir eingereichte Abschlussarbeit (Bachelor-/Master-Thesis) selbstständig verfasst und keine andere als die angegebene Quellen und Hilfsmittel benutzt sowie Stellen der Abschlussarbeit, die anderen Werken dem Wortlaut oder Sinn nach entnommen wurden, in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Ich bin damit einverstanden, dass die Arbeit durch Dritte eingesehen und unter Wahrung urheberrechtlicher Grundsätze zitiert werden darf.

Ort und Datum: _____

Unterschrift: _____

Danksagung

Zusammenfassung

Die Entwicklung von autonomen Fahrzeugen ist seit vielen Jahren ein großes wissenschaftliches Feld. Durch die komplexen Zusammenhänge verschiedener Systemen werden viele Daten benötigt um die Systeme zu entwickeln, prüfen und zu verifizieren. Dabei arbeiten Hochleistungsrechner mit künstlichen Intelligenzen, welche Daten von diversen Perzeptionssensoren bekommen und auf Basis deren Verarbeitung verschieden Aktuatoren kontrollieren. Da die benötigte Datenmenge nicht alleine über Testfahrten und die damit generierten Daten erzeugt werden kann, ist die Generierung von synthetischen Fahrdaten mittels Simulatoren im Laufe der Zeit zu einem wichtigen Werkzeug der Forschung geworden. Diese Simulatoren weisen allerdings immer noch viele Einschränkungen auf. Simulatoren für die Perzeptionssensorik unterliegen dabei häufig einem Funktionsparadigma, wodurch Sensoren immer fehlerfrei abgebildet und sind auch nur vereinfacht im Vergleich zur Realität dargestellt werden. Da in der Realität allerdings Fehler (Signalstörung der Sensorik, Verschmutzung der Sensorabdeckung etc.) zum Alltag gehören, ist es für zukünftige Arbeiten notwendig ein Fehlerparadigma zu etablieren und so die Auswirkungen von Fehlern bei der Entwicklung und Validierung von entsprechenden Systemen zu berücksichtigen. Die vorliegende Arbeit soll einen Lösungsansatz für dieses Problem vorbringen. Dafür wird zunächst eine grundlegende Recherche betrieben wird. Die gefundenen Fehlereinflüsse werden in Kategorien entsprechend ihrer Auswirkung unterteilt. Aufbauend auf diesen Kategorien sind Fehlermodelle für Simulatoren definiert und entwickelt worden, um die Auswirkungen in den synthetisch generierten Daten abbilden zu können. Die realistische Darstellung der Fehlerauswirkungen in den generierten Daten wird validiert und abschließend zeigt die Arbeit auf, wie der veränderte Simulator in der Praxis angewendet werden kann. Hier konnte in einem ersten Test gezeigt werden, dass veränderte Daten der Perzeptionssensorik einen Einfluss auf die Implementierung eines Open Source Adaptive Lane Keeping System (ALKS) haben.

Inhaltsverzeichnis

1 Einleitung

Alle großen Automobilhersteller forschen aktuell an autonomen Fahrzeugen. Während ein gewisser Grad an Automatisierung hier bereits erreicht ist, ist es noch ein langer Weg bis zur Zulassung komplett autonom fahrender Fahrzeuge im Straßenverkehr. Dabei wird die Automatisierung durch die Society of Automotive Engineers (SAE) in 5 Level eingeteilt. Die Level 0, 1 und 2 erfordern dabei noch die volle Aufmerksamkeit des Fahrers, da dieser das Fahrzeug noch vollständig steuert und lediglich von dem System unterstützt wird. So hat Mercedes bereits Fahrzeuge mit Level-3-Systemen in Form eines Adpative Lane Keeping System (ALKS) zugelassen bekommen, welche es erlauben, dass sich der Fahrer in bestimmten Bereichen wie z.B. bei Stau auf der Autobahn nicht mehr auf den Verkehr konzentrieren muss. Ein ALKS ist dabei für die Steuerung des gesamten Fahrzeuges innerhalb einer definierten Situation zuständig. Definiert wird eine solche Situation als eine sogenannte Operational Design Domain (ODD), welche zusammenfasst unter welchen Bedingungen ein ALKS operieren kann und darf. Eine mögliche ODD wäre eine Autobahnfahrt bei Tageslicht mit guten Sichtverhältnissen (kein Nebel, wenig Regen) bei hohem Abstand zu anderen Verkehrsteilnehmern. Eine andere ODD hingegen könnte sich nur durch einzelne Bedingungen unterscheiden, so könnte eine andere ODD die selbe Fahrsituation bei Nacht oder mit einem geringeren Abstand zu anderen Verkehrsteilnehmern beschreiben Außerhalb dieser Situationen wird das Fahrzeug weiterhin durch den Fahrer gesteuert. VW führt aktuell Tests mit Level-4-Systemen durch bei denen auch größere Strecken autonom zurückgelegt werden können und der Fahrer nur noch für wenige Situationen benötigt wird[1]. Mit Level-5-Systemen benötigt das Fahrzeug keinen Fahrer mehr und ist völlig autonom.
In diese Arbeit wird sich auf Level 3 Systeme fokussiert, da nur diese eine vollautomatisierte Fahrfunktion aufweisen und damit relevant für den Rahmen der Arbeit sind. Damit ein solcher Grad der Automatisierung erreicht werden kann, ist es notwendig, dass das Fahrzeug seine Umgebung präzise wahrnimmt. So müssen der Straßenverlauf als auch andere Verkehrsteilnehmer, Verkehrsschilder usw. frühzeitig und zuverlässig erkannt werden. Dafür werden in automatisierten und autonomen Fahrzeugen Perzeptionssensoren verbaut. Hier wird bei den meisten Herstellern auf eine Verknüpfung von Radio detection and ranging (Radar)-, Light detection and ranging (LiDAR)- und Kamera-Sensoren gesetzt. Damit autonome Fahrzeuge zugelassen werden können, muss sichergestellt sein, dass diese entsprechende Sicherheits- und Qualitätsstandards erfüllen. Dies bedeutet, dass die in autonomen Fahrzeugen verwendeten Systeme in Form von Algorithmen und Künstlichen Intelligenz (KI) trainiert, getestet und zugelassen werden müssen. KI-Systeme müssen mit großen und vielfältige Datenmengen trainiert werden, damit sie zuverlässig und sicher in auch ihnen unbekannten Szenarien agieren können. Auf Grund der Menge und Diversität der Szenarien gegeben durch die erforderliche Kombination

[1] https://www.adac.de/rund-ums-fahrzeug/ausstattung-technik-zubehoer/autonomes-fahren/technik-vernetzung/aktuelle-technik/ Abgerufen am 17.05.2024

aus diversen Umgebungsverhältnissen (Straßen, Wetter, Verkehrsteilenehmer etc.) in unterschiedlichen Fahrsituationen, ist es sehr aufwändig und kostspielig diese Daten zu generieren. Zusätzlich sollten diese Daten auch seltene und spezielle Szenarien wie z.B. Unfälle mit Beteiligung des autonomen Fahrzeuges, aber auch Daten in denen das autonome Fahrzeug passiver Teilnehmer eines Unfalls oder Risikoszenarios ist, beispielsweise durch das bilden einer Rettungsgasse etc. beinhalten. An diesem Punkt ist es daher nötig Daten synthetisch generieren zu können [1], um der Anforderung der Datendiversität gerecht zu werden und somit auch zu ermöglichen, dass die KI-Systeme mit anderen Daten getestet werden können, als mit denen sie trainiert worden sind.
Zusätzlich lassen sich in synthetischen Daten verschiedene Szenarien einfacher verbinden. So sind z.B. wetterabhängige Daten einfacher zu generieren, da keine Abhängigkeit vom tatsächlichen Wetter gegeben ist, da dies einfach im Simulator eingestellt werden kann (insofern dieser über diese Einstellungsmöglichkeit verfügt). Für die Generierung dieser Daten wird dabei auf verschiedene Simulatoren zurückgegriffen. Allerdings gibt es auch hier Unterscheide zwischen den Simulatoren. Einige sind dabei spezialisiert auf die Berechnung von Trajektorien und damit auf das Manövrieren des Fahrzeuges, während andere auf die Erstellung und Verarbeitung der Daten der Perzeptionssensorik fokussiert sind. Diese Simulatoren spiegeln die Verkehrsumgebung sowie die Sensoren in autonomen Fahrzeugen wieder und sind so in der Lage realitätsnahe Daten für die verschiedenen Sensoren zu generieren.
Aktuell unterliegen Open-Source Simulatoren weitestgehend einem Funktionsparadigma, was bedeutet, dass die Umwelt lediglich in einem idealisierten Zustand abgebildet wird. Dies bedeutet z.B. dass Verkehrsschilder immer perfekt abgebildet werden und nicht durch Verschmutzung, Vandalismus oder sonstiges beeinträchtigt werden. Zusätzlich bedeutet dies, dass Sensoren immer fehlerfrei funktionieren und keine Störungen durch die Umgebung oder andere Einflüsse erfahren. Dies führt dazu, dass mögliche Effekte und Aspekte der Realität in diesen synthetisch generierten Daten nicht auftauchen. Beispielsweise spiegelt sich so der Einfluss von diversen Wetterereignissen wie Regen, Schnee sowie unterschiedliche Bodenverhältnisse auf die Sensorik und das Fahrzeug in der Simulation nicht wieder. Zusätzlich können Ausfälle oder Fehlverhalten wie unter anderem Datenübertragungsfehler, fehlerhafte Positionierungen der Sensorik sowie Angriffe nur schwierig in der Praxis erzeugt werden, da solche Zustände nur selten auftreten oder sie explizit herbeigeführt werden müssen.
Aktuelle Simulatoren sind nicht in der Lage diese Fehler und Angriffe abzubilden. Damit eine Betrachtung dieser möglich wird, muss ein Wechsel zum Fehlerparadigma erfolgen. Unterschiedliche Simulatoren bieten bereits erste Ansätze um dieses Problem anzugehen. CARLA[2] und Cogna[3] bieten so die Möglichkeit Wettereinflüsse auf Kamerasensoren zu berücksichtigen. Cogna, unter anderem, ist weiterhin in der Lage Verschmutzungen auf der Kameralinse zu simulieren. Für den CARLA-Simulator wurde bereits bei

[2] https://carla.org/ Abgerufen am 17.05.2024
[3] https://www.cognata.com/ Abgerufen am 17.05.2024

den CARLA Talks 2020 von einem externen Sensor Interface, mit der Berücksichtigung von physikalischen Prinzipien gesprochen[4]. Allerdings gibt es bezüglich diesem Interface keine weiteren Veröffentlichungen. Ein solches System ist bereits im Aurelion-Simulator[5] eingebunden.
Allerdings wird der direkte Einfluss von z.B. simulierten Wetter auf LiDAR- und Radar Sensoren momentan nicht oder nur teilweise betrachtet. Dennoch gibt es Ausarbeitungen die Wettereffekte wie Regen auf die vom LiDAR generierten Daten betrachten wie in [2] vorgestellt. Dabei sollen die Auswirkungen von Regen nachträglich in den Datensatz eingebunden und so ein synthetischer Datensatz generiert werden. Dafür wurde dort ein Algorithmus entwickelt, welcher den wetterbedingten Einfluss auf die LiDAR-Daten betrachtet und den Datensatz unter Berücksichtigung eines definierten Wetterzustandes verändern kann.
Allerdings spiegelt dieser Ansatz nur einen einzelnen Einfluss auf die Daten wieder. Damit für die Zukunft umfangreiche Datensätze generiert werden können und das Fahrzeug nicht nur unter idealen Bedingungen, sondern auch in realistischeren Umgebungen und sogar in Fehlerfällen getestet werden kann, müssen mehr Bemühungen investiert werden ein solches Fehlerparadigma in Simulatoren zu integrieren. Während somit bereits mehr Einflüsse durch die physikalischen Gegebenheiten auf die Sensoren berücksichtigt werden, werden spezifische Fehler, wie z.B. Datenübertragungsfehler oder Fehler durch die Verschiebung der Sensorposition noch nicht umfangreich betrachtet. Das Ziel ist es so Modelle zu entwickeln und in einen Simulator einzubinden um Sensorfehler in synthetisch generierten Daten abzubilden. Dies bezüglich werden in Kapitel 2 die Grundlagen von Radar- und LiDAR-Sensoren betrachtet. Aufbauend darauf werden in Kapitel 3 die Umgebungen der Sensoren vorgestellt und dadurch entstehende Fehlermöglichkeiten analysiert. In Kapitel 4 werden diese Fehler anhand ihrer Effekte kategorisiert und in Kapitel 5 in Algorithmen umgewandelt und in den CARLA Simulator implementiert. Abschließend wird in Kapitel 6 aufgezeigt wie diese Umsetzung in der Praxis angewendet werden kann, um die Performanz eines ALKS zu überprüfen. Aufbauend auf den Strukturen die in der Praxis angewendet werden sollen erste Einflüsse auf diese untersucht und bewertet werden. Erste Arbeiten diesbezüglich wurden bereits in [3] veröffentlicht.

[4] https://carla.org/2020/06/09/talks_2020/ Abgerufen am 14.05.2024
[5] https://www.dspace.com/en/pub/home/products/sw/experimentandvisualization/aurelion_sensor-realistic_sim.cfm Abgerufen am 17.05.2024

2 Grundlagen

Für das Verständnis der später vorgestellten Modelle sowie deren Umsetzung ist es wichtig zuvor entsprechende Grundlagen zu betrachten, diese Grundlagen werden in den nachfolgenden Kapiteln vorgestellt.

2.1 Sensoren in Autonomen Fahrzeugen

Für die Umsetzung eines autonomen Fahrzeuges ist es wichtig Umgebungsinformationen zu sammeln. Damit autonomes Fahren möglich ist, muss das Fahrzeug seine eigene Umgebung erkennen. Dies erfolgt über die Verwendung von verschiedenen Sensoren deren Daten zu einer Gesamtwahrnehmung fusioniert werden. Diese Datenvereinigung wird auch Sensorfusion genannt.

2.1.1 Perzeptionssensoren

Heutzutage ist es von hoher Wichtigkeit Sensoren in Fahrzeugen zu verbauen. Während Assistenzsysteme bereits auf Sensoren angewiesen sind, wird der Bedarf an Sensoren für autonome Fahrzeuge weiter steigen. Autonome Systeme benötigen mehr Informationen als in herkömmlichen Fahrzeugen bereitgestellt wird. Damit entsprechende Systeme korrekte Entscheidungen Treffen ist es wichtig ihnen so viele Daten wie möglich zur Verfügung zu stellen. Dabei kann sich aber nicht nur auf einzelne Sensorarten verlassen werden. Wichtig ist hier eine Vielfalt an verschiedenen Sensoren und die Daten dieser zusammenspielen zu lassen. In heutigen Fahrzeugen werden primär

1. Radar,
2. LiDAR,
3. Kameras,
4. Ultraschallsensoren

verbaut. Diese Diversität ist wichtig um die Nachteile einzelner Sensorarten auszugleichen. Entsprechende Vor- und Nachteile der unterschiedlichen Sensorarten können der Tabelle 2.1 entnommen werden.
Wie hier zu sehen ist, gleichen sich Vor- und Nachteile einzelner Sensoren aus. So ist ein Radar z.B. sehr gut darin die Geschwindigkeiten von Objekten zu detektieren. Da ein Radar-Sensor allerdings nur eine durchschnittliche Auflösung bietet ist es wichtig diese Schwäche entsprechend auszugleichen. Dafür wäre z.B. die Verwendung einer Kamera eine Möglichkeit. So ist es z.B. aktuell auch keine Option nur LiDAR-Sensoren innerhalb eines Fahrzeuges zu verbauen, da das aktuelle System noch sehr viel Platz benötigt.

Sensor	LiDAR	Radar	Kamera	Ultraschall
Primäre Technologie	Laserstrahlen	Radiowellen	Licht	Schallwellen
Reichweite	ca 200 m	ca 250 m	ca 200 m	ca 5 m
Auflösung	gut	durchschnittlich	sehr gut	schlecht
Betroffen von Wetter	Ja	Ja	Ja	Ja
Betroffen von Beleuchtung	Nein	Nein	Ja	Nein
Erkennt Geschwindigkeiten	gut	sehr gut	schlecht	schlecht
Erkennt Entfernungen	gut	sehr gut	schlecht	gut
Größe	groß	klein	klein	klein

Tabelle 2.1 – Gegenüberstellung von einzelner Hauptmerkmale von Sensoren aus [4]

Dies zeigt, dass eine Sensorfusion innerhalb eines Fahrzeuges daher von großer Wichtigkeit ist. Dies stellt auch sicher, dass das Fahrzeug in verschiedenen Wetterbedingungen noch funktionsfähig bleibt, da alle vorgestellten Sensoren unterschiedlich stark durch diverse Wetterbedingungen beeinträchtigt werden. So stellt eine Sensorfusion, unter Beachtung der Datengüte der Sensoren, sicher, dass die bestmöglichen Informationen für nachfolgende Systeme generiert [4] werden. Für den weiteren Verlauf wird sich ausschließlich auf Radar- und LiDAR-Sensoren bezogen.

2.1.2 Positionierung von Sensoren

Wie im vorherigen Kapitel gezeigt, ist die Verwendung von Sensoren für autonome Fahrzeuge wichtig und es müssen weitere Aspekte bedacht werden. Wie Hartstern, Rack und Stork [5] aufzeigen ist die Positionierung von Sensoren für das frühzeitige Erkennen der Umgebung ein entscheidender Faktor.
Wie die Publikation zeigt konnten Objekte in den dort gefahrenen Simulationen, in gewissen Szenarien, unterschiedlich schnell erkannt werden. Einige Positionierungen um das Fahrzeug führten hier unter anderem zu einer früheren Erkennung von bis zu vier Sekunden, im direkten Vergleich mit anderen Sensorpositionierungen. Einige wichtige Aspekte können der Abbildung 2.1 entnommen werden.
Dort wird aufgezeigt, wie die Positionierung des Sensors das FOV eines Sensors beeinflusst. Auch ist dort zu erkennen, dass die Positionierung des Sensors über gewisse Blind Spots entscheidet und somit auch darüber was das Fahrzeug sehen kann. Dies wird damit untermauert, dass entsprechende Blind Spots zum Übersehen von Passanten jeglicher Größe führen kann [5]. Während tief positionierte Sensoren eher kleine Objekte nahe am Fahrzeug erkennen können, bieten höher positionierte Sensoren eine bessere Übersicht hinter bzw. über Objekten am Straßenrand.

2.2 Generelle Ausfallmodelle Elektrischer Bauteile

Es gibt viele unterschiedliche Arten und Ursachen wie elektrische Bauteile ausfallen oder fehlerhafte Daten generieren können. Diese können in verschiedensten Umfängen auf-

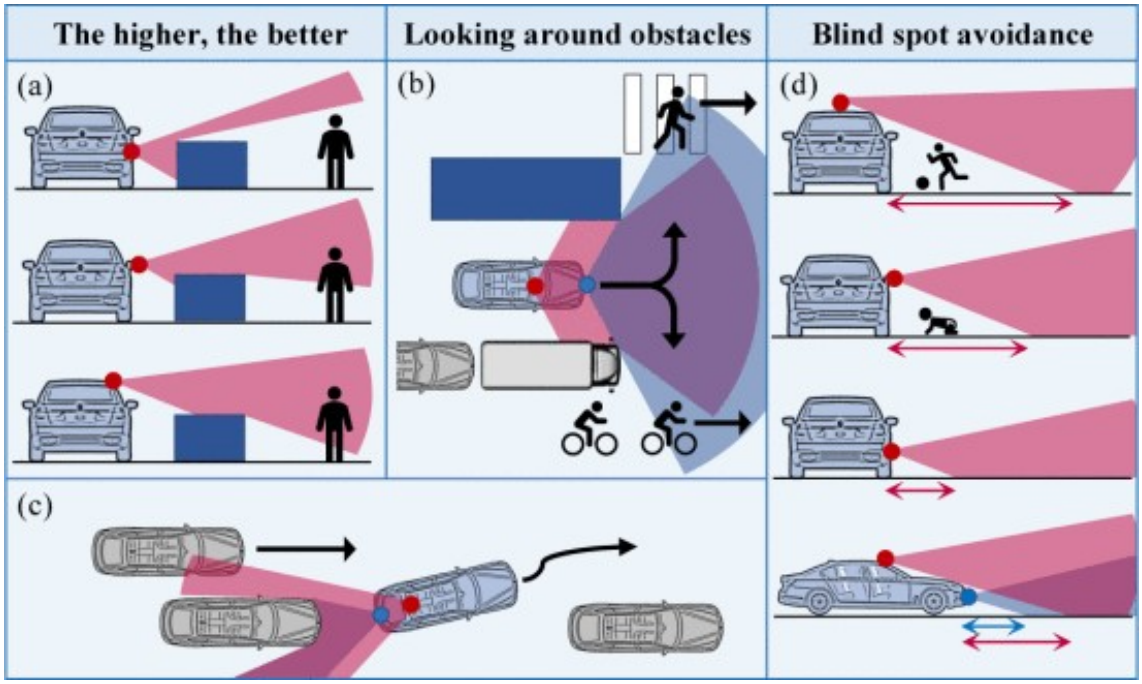


Abbildung 2.1 – Auswirkungen der Positionierung eines Sensors auf das Field of View (FOV)

treten und nur zu minimalen Änderungen in den Daten, aber auch zu dem komplett Ausfall eines elektronischen Bauteils führen. Die nachfolgenden Modelle werden dabei anhand eines Radar-Sensors erläutert. Die konkrete Umsetzung dieser Fehlermodelle wird in Kapitel 4 beschrieben.
Ausfälle und Fehler innerhalb von Sensoren lassen sich hier in drei primäre Kategorien unterteilt:

1. Interne Safety[1]
2. Externe Safety
3. Cybersecurity

Dabei werden die Fehler anhand ihrer ursächlichen Herkunft in die vorgestellten Kategorien eingeteilt. Dabei werden hier nur Fehler betrachtet die in einem Sensor auch auftreten können und nicht durch bestehende Qualitätsmanagementprozesse abgefangen werden.

2.2.1 Interne Safety

In dieser Kategorie werden alle Fehler zusammengefasst bei denen ein Defekt innerhalb des Sensors ursächlich für das Auftreten ist. Dies umfasst sämtliche Fehler die in der Hard- und Software auftreten können und die Funktionalität des Sensors einschränken,

[1] Da es im Englischen, anders als im Deutschen, eine Unterscheidung der Sicherheitsbegriffe gibt, werden fortlaufend die Englischen Begriffe verwendet.

aber auch Fehler innerhalb des Designs des Sensors werden dieser Kategorie zugeordnet.
So kann z.B. durch die Alterung von Bauteilen oder deren Abnutzung ein Fehler entstehen. Ein solcher Fehler wäre z.B. ein Wackelkontakt. Ein Wackelkontakt kann dazu führen, dass die Verbindung zwischen dem Sensor sowie umliegenden Systemen oder innheralb des Sensors gestört wird.
Ein möglicher Grund kann die Korrodierung von Kontakten innerhalb eines Bauteils sein[2]. Aber auch Fehler im Design eines Bauteils können solche Problematiken hervorrufen. So können Fehler in der Software sich auf vielfältige Art auswirken, von Rundungsfehlern bis hin zum Überlaufen von Speicherbereichen die im schlimmsten Fall zum Totalausfall des Programms führen. Damit können Software-Fehler auch ursächlich für einen Ausfall des Bauteils sein.

2.2.2 Externe Safety

Bei dieser Kategorie werden Fehler zusammengefasst bei denen nicht ein Fehler im Bauteil selbst vorhanden ist, sondern der Fehler durch falsche Verwendung, Beeinflussung von Außen oder durch Missbrauch ausgelöst wird. Während bei einem Datenübertragungsfehler aus dem vorhergegangen Kapitel 2.2.1 von einem Ausfall der Funktion ausgegangen werden kann, ist hier die Sollfunktion vorhanden. Wie bereits in Kapitel 2.1.1 aufgezeigt, unterliegt die Perzeptionssensorik einem wetterbedingten Einfluss. So wird die Datenqualität von z.B. LiDAR-Sensoren durch Regen stark beeinflusst, die interne Sensorfunktion entspricht jedoch ihrer Definition.

2.2.3 Cybersecurity

Für die Sicherstellung der Funktionsweise eines Bauteils ist es heutzutage nicht nur wichtig dieses, sowie seine direkte Umgebung zu betrachten, sondern auch seine Anfälligkeit gegenüber der gezielten Manipulation durch Dritte. Besonders wichtig ist dies für Bauteile die kritische Daten generieren oder verarbeiten. Wie in Kapitel 2.2.2 findet auch bei dieser Kategorie primär eine Veränderung der Daten statt. Während im vorherigen Kapitel die Veränderung durch Einflüsse der Umwelt erfolgte. Ist es hier der ausschlaggebende Punkt, dass die Daten auf eine gezielte Art manipuliert werden. Dabei können Angreifer einen sensorspezifischen Angriff durchführen, der je nach Szenario und Ziel die Funktionalität des Sensors manipulieren oder komplett verhindern kann. Durch die Manipulation eines Sensors können Informationen die in einem regulär Betrieb gesammelt werden unterdrückt oder verfälscht werden.
Im Rahmen der Cybersecurity werden hier zwei Angriffsarten betrachtet welche auch bei den untersuchten Sensorarten eine wichtige Rolle spielen können. Die hier betrachteten Angriffsszenarien werden im NIST Internal Report 8323r1 wie folgt definiert:

[2] https://www.imws.fraunhofer.de/de/specials/korrosion-elektronik.html Abgerufen am 15.04.2024

1. Jamming: „An attack that attempts to interfere with the reception of broadcast communications “([6] S.104).

2. Spoofing:„The deliberate inducement of a user or resource to take incorrect action “([6] S. 106).

Diese Angriffsarten können auf verschiedene Systeme innerhalb einer IT-Struktur durchgeführt werden. Beim Jamming geht es oftmals darum, durch Interferenzen das System lahmzulegen bzw. dessen Funktion einzuschränken. Beim Spoofing hingegen sollen gezielt falsche Informationen generiert werden. Im Rahmen des autonomen Fahrens kann dies bedeuten, dass der Angreifer Sensoren mit dem Ziel manipuliert, Objekte zu erkennen die nicht real existieren um z.B Notfallbremsungen oder ähnliches auszulösen. Je nach betrachtetem System sind verschiedene Angriffsarten wahrscheinlicher, da sie eher zu einem Erfolg führen oder einfacher umzusetzen sind. Genauere Informationen im Bezug auf die in dieser Arbeit betrachteten Sensoren Schwerpunktmäßig in Kapitel 3.3 diskutiert.

2.3 Betrachtete Sensoren

Für die Arbeit ist es wichtig die Funktionsweise und technischen Grundlagen der betrachteten Automobilsensoren genauer zu verstehen, im speziellen Radar und LiDAR. Dafür wird zunächst ein Überblick über elektromagnetische Wellen gegeben, welche die physikalische Basis der Radar- und LiDAR-Sensoren darstellt. Darauf aufbauend werden in den Unterkapiteln 2.3.2 und 2.3.3 die Sensoren, im Hinblick auf ihre Historie und den Einsatz in der Automobilindustrie vorgestellt.

2.3.1 Elektromagnetische Wellen

Eine elektromagnetische Welle, ist eine Welle aus gekoppelten elektrischen und magnetischen Feldern. Die Eigenschaften der Welle werden durch Wellenlänge und Frequenz bestimmt, über das elektromagnetische Spektrum dargestellt werden können. Darum gibt es unterschiedliche Wellen wie z.B.: Mikrowellen, Radiowellen, Lichtwellen.

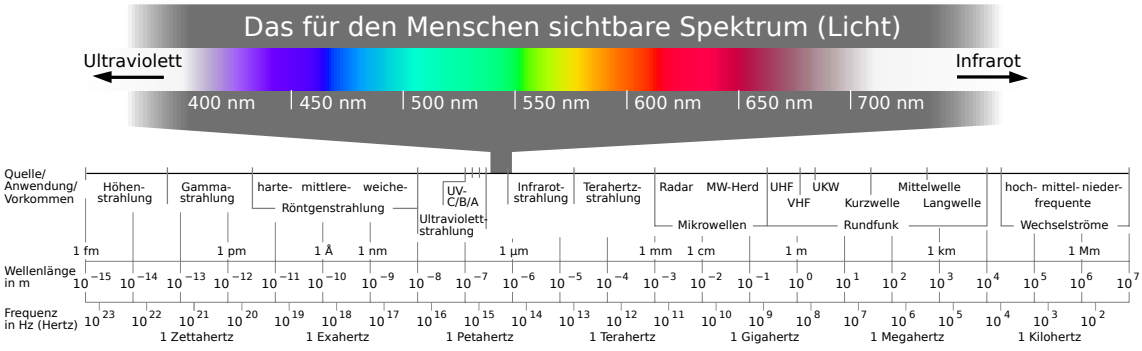


Abbildung 2.2 – Spektrum von elektromagnetischen Wellen [3]

Die Abbildung 2.2 visualisiert das elektromagnetische Spektrum. Wie der Abbildung zu entnehmen ist, befindet sich der für uns Menschen sichtbare Lichtanteil innerhalb eines Bereiches von 400 - 700 nm, mit einer Frequenz von unter einem Petaherz. Radarwellen sowie die Lichtwellen eines LiDAR-Sensors finden sich ebenfalls auf diesem Spektrum wieder. Radarwellen bewegen sich dabei in einem Frequenzbereich von etwa 10 - 100 Gigaherz bei einer Wellenlänge von etwa 1 mm bis 1 cm. LiDAR-Sensoren hingegen nutzen eine Frequenz im Bereich von etwa 1250 nm mit einer kleineren Wellenlänge von unter einem Petaherz [7]. Daraus folgt, dass sowohl Radar als auch LiDAR-Wellen für den Menschen nicht mehr sichtbar sind.
Durch die Ähnlichkeit der genutzten Wellen ähnelt sich auch die Funktionsweise beider Sensoren: Elektromagnetische Wellen werden ausgesendet, von getroffenen Objekten reflektiert und wieder vom Sensor aufgenommen. Die somit erhaltenen Daten werden aufbereitet um verschiedene Informationen über die getroffenen Objekte zu berechnen [8].

2.3.2 Radar

Im Folgenden wird das grundlegende Konzept eines Radars im Hinblick auf den Einsatz innerhalb eines Fahrzeuges erläutert.
Innerhalb der Automobilindustrie gibt es keinen einheitlichen Radar-Sensor. Verbaute Radarsensoren unterscheiden sich in spezifischen Parametern wie z.B. Frequenzbereich, Anzahl der Antennen etc. Dadurch ergeben sich unterschiedliche Eigenschaften wie abgedeckte Reichweiten, Form und Größe des FOV und weiteres. Dementsprechend ist die Auswahl des spezifischen Sensors abhängig von seiner Aufgabe und seinem Einsatzgebiet. Das grundlegende Funktionskonzept ist in allen Radar-Sensoren allerdings einheitlich. Während Radarwellen generell in einem Frequenzbereich von 10 bis 100 GHz definiert sind, werden im Automobilkontext lediglich Radarwellen innerhalb eines Frequenzbereichs von etwa 76 bis 81 GHz verwendet [9].
Wichtig ist zusätzlich zu erwähnen, dass es für Radar-Sensoren verschiedene Funktionsprinzipien gibt, die wie bereits erläutert, alle auf dem Senden und Empfangen von Radarwellen basieren. Das Grundlegende Prinzip des Radars ist dabei das in Kapitel 2.3.1 vorgestellte, welches sich in einem entsprechenden Zyklus wiederholt.
Moderne Radarsysteme im Automobilbereich sind allerdings sogenannte Frequency-modulated continuous wave (FMCW)-Radare.
Diese Sensoren bestehen aus einer oder mehrerer Antennen die für das Senden von Radarwellen verantwortlich sind. Zeitgleich besitzen sie mehrere Antennen die für das Empfangen von reflektierten Radarwellen genutzt werden. Diese Antennen sind mit dem Reciever verbunden und dort werden die empfangenen Daten an eine Signalverarbeitungseinheit weitergeleitet. Hier werden sämtliche Daten aufbereitet und an nachfolgen-

[3] https://de.wikipedia.org/wiki/Elektromagnetische_Welle#Spektrum Abgerufen am 26.07.2023

de Systeme des Fahrzeugs gesendet. Diese Aufbereitung der Daten wird oftmals Preprocessing genant. Ein Teil dieses Preprocessing kann dabei die Objekterkennung sein. So liefern Radar-Sensoren von z.B. Continental [4], Infineon[5] und Innosent[6] neben den generierten Punktwolken auch Informationen über die Objekte.
Allerdings unterscheidet sich die Arbeitsweise von FMCW-Radarsensoren von der prinzipiellen Funktionsweise von Radaren. FMCW-Systeme senden konstant Radarwellen aus und zusätzlich empfangen sie reflektierte Wellen konstant. Dies bedeutet, dass FMCW-Radarsensoren nicht in einem Zyklus arbeiten, sondern sämtliche Aspekte des Grundprinzips parallel ablaufen. Da die empfangene Radarwelle mit der gesendeten verglichen werden kann und über den berechneten Versatz der Welle, sowie Stauchungen oder Streckungen dieser, die gesuchten Informationen berechnet werden können, ist ein solcher Ansatz möglich. Die von einem Radarsensor gesammelten Informationen sind:

- Winkel und Richtung des Objektes,
- Distanz zum Objekt und
- Geschwindigkeit relativ zum Radar.

Für die Berechnung des Winkels des getroffenen Objektes wird berücksichtigt wie die reflektierte Radarwelle die Antennen erreicht. Unter Berücksichtigung der Informationen vom Erhalt der Welle kann die Position des reflektierenden Objektes berechnet werden. Die Distanz des Objektes wird über die verstrichene Zeit zwischen Senden und Empfangen der Radarwellen ermittelt. Dabei wird die zeitliche Verschiebung der gesendeten Welle im Vergleich zur erhaltenen Wellen betrachtet. Aus der Kombination dieser beiden Informationen kann die Position des Objektes relativ zum Radar berechnet werden.
Auf Basis des physikalischen Prinzips des Doppler-Effekts wird die Geschwindigkeit des Objektes berechnet. Während des Reflektionsvorganges der Welle an einem Objekt, wird diese Welle gestaucht oder gestreckt. Anhand dieser Stauchungen und Streckungen und dem entsprechenden Wissen über die gesendete Welle, ist ein Radarsensor in der Lage die Geschwindigkeit des getroffenen Objektes zu bestimmen.
Zusätzlich sind moderne Radarsensoren in der Lage die Radar Cross Section (RCS) zu bestimmen. Dabei ist die RCS ein Wert der die Reflektivitätseigenschaften eines Objektes angibt. Hier haben verschiedene Ausprägungen des Objektes einen Einfluss auf die RCS. Darunter fallen Ausprägungen wie:

- Form des Objektes,
- Materialeigenschaften des Objektes,
- Winkel unter dem des Objekt getroffen wird,

[4] https://www.continental-automotive.com/de/komponenten/radars/long-range-radars/advanced-radar-sensor-ars620.html Abgerufen am 08.05.2024
[5] https://www.infineon.com/cms/en/product/sensor/radar-sensors/ Abgerufen am 08.05.2024
[6] https://www.innosent.de/automotive/radar-360-detektion/ Abgerufen am 08.05.2024

- Absorption der Energie der eintreffenden Radarwelle

und weitere. Somit kann die RCS genutzt werden um zu bestimmen was für ein Objekt getroffen worden ist ([10] , [11]).

Radarhistorie Patentiert wurde das Radar 1904 von Christian Hülsmeyer. Bereits 1944 kamen dann die ersten voll elektronischen Scanradare, das FuMG 41/42 Mammut-1 zum Einsatz [12]. Über die nächsten Jahre wurde viel an den Radaren geforscht, bereits in den 60ern wurden die ersten Ansätze durchgeführt um die Radartechnologie in Fahrzeugen zu verwenden. Um die Radarsysteme dabei besser zu integrieren wurden sie im Laufe der Zeit verkleinert und dadurch auch die Frequenz der Radarwellen erhöht. Bereits 1992 wurden 1500 Busse mit Radarsystemen ausgestattet und fuhren über 250 Millionen Meilen, was eine erfolgreiche Reduzierung der Unfälle um 25% erreichte.
Aus verschiedenen Gründen mussten die Radarsensoren allerdings entfernt werden, da sie unter anderem mit der Funkfrequenz der Polizei überschnitten [13]. Allerdings wurden auch hier bereits Probleme erkannt für die immer noch keine perfekte Lösung gefunden worden ist. Zu den Anfangszeiten von Radaren waren Interferenzen zwischen Radaren kein nennenswertes Problem, da leidiglich wenige Radarsysteme verbaut wurden. Im Laufe der Jahre stieg die Anzahl verwendeter Radarsensoren massiv an wodurch sich die Anzahl an Interferenzen erhöhte.

2.3.3 LiDAR

Neben den bereits vorgestellten Radarsensoren werden auch LiDAR-Sensoren in Fahrzeugen verbaut. Wie bereits in Kapitel 2.3.1 vorgestellt basieren Radarsensoren und LiDAR-Sensoren auf ausgesendeten elektromagnetischen Wellen. Wie auch Radar-Sensoren können LiDAR-Sensoren unterschiedliche Konfigurationen bezüglich ihrer abgedeckten Reichweite, Frequenz und des FOV aufweisen.
Während Radar-Sensoren vielfach an ihr Einsatzgebiet anpassbar sind, ist dies aktuell bei LiDAR-Sensoren nicht der Fall. Aktuell weisen die meisten LiDAR-Sensoren ein 360° FOV auf und sind im Vergleich zu anderen Sensoren sehr teuer. Daher sind viele dieser LiDAR-Sensoren nur innerhalb ihrer verwendeten Frequenz und der abgedeckten Reichweite parametrisierbar. Da dies der aktuelle Stand der Technik ist, wird sich in dieser Arbeit auch ausschließlich auf solche Sensoren konzentriert. Mit zukünftigem Voranschreiten der Entwicklung von LiDAR-Sensoren ist es möglich, dass diese eine größere Konkurrenz zu Radar-Sensoren hervorbringen und somit günstiger und besser auf Einsatzgebiete angepasst werden können. Erste Prototypen können dabei mit einem FOV-System arbeiten wie es die bereits vorgestellten Radar-Sensoren bieten [7].
Im Gegensatz zu den FMCW-Radarsensoren weisen LiDAR-Sensoren weiterhin Zyklen auf, in denen eine Messung erfolgt. Dabei folgt der Zyklus eines LiDARs folgendem Ablauf:

1. Aussenden der Lichtwellen,
2. Empfangen von Lichtwellen,
3. Auswerten der Daten,
4. Ruhezeit des Sensors.

Wie in dem Ablauf zu sehen, wird der zugrundeliegende Zyklus um eine Ruhezeit erweitert. Die Ruhezeit, oft auch als „Dead Time“ bezeichnet, ist dabei notwendig um die Qualität der Messung sicherzustellen. Zusätzlich kann die „Dead Time“ optimiert werden um die Qualität der Messung weiter zu steigern. Weitere Informationen diesbezüglich können Feng [14] entnommen werden.
Wie auch mit Radarsystemen kann das LiDAR genutzt werden um die Position und Distanz einzelner Objekte festzustellen. Auch eine Berechnung der Geschwindigkeit eines Objektes ist theoretisch möglich, die beruht allerdings nicht wie beim Radar auf dem Dopplereffekt sondern auf einem deutlich komplexeren Verfahren welches an dieser Stelle nicht weiter erläutert wird, da es für den Verlauf der Arbeit nicht relevant und in der Praxis nicht gängig ist [7].
Wie bei Radar-Sensoren finden auch bei LiDAR-Sensoren oftmals ein Preprocessing statt. So liefern LiDAR-Sensoren von Valeo[7] und MicroVision[8] neben Punktwolken auch Informationen in Form von Objektlisten.

LiDAR-Historie Bereits in den 60er Jahren wurde die Methode hinter dem LiDAR verwendet um die Entfernung zwischen Erde und Mond zu bestimmen. Dabei sendete das Lincoln laboratory des MIT einen Laserstrahl zum Mond und stoppte die Zeit wie lange dieser benötigte um vom Mond reflektiert und auf der Erde wieder anzukommen. Auf der Grundlage der verstrichenen Zeit und der Verwendung der konstanten Lichtgeschwindigkeit konnte die zurückgelegte Strecke berechnet werden. Auf Basis dieser Methodik wurden die ersten rotierenden LiDAR entwickelt. Dabei wurde 2005 das erste LiDAR auf einem Fahrzeug, für ein Wüstenrennen, montiert. Dort gelang es Punktwolken mit diesem LiDAR aufzuzeichnen.
Aufbauend auf diesem Erfolg wurden 2007 sechs Rennfahrzeuge für ein Stadtrennen mit entsprechenden LiDAR-Systemen ausgestattet. Bei beiden Rennen wurde gezeigt, dass Punktwolken erstellt werden können, welche die reflektierenden Objekte in der Umgebung widerspiegeln.
Rotierende Systeme benötigen viel Platz und können damit nur auf dem Dach des Fahrzeuges montiert werden. So findet seit einigen Jahren auch eine Entwicklung statt, bei der die Systeme kleiner und kompakter gebaut werden. Durch eine solche Anpassung

[7] https://www.valeo.com/en/catalogue/cda/long-range-lidar-sensors-scala-gen-3/ Abgerufen am 08.05.2024
[8] https://microvision.com/products/mavin-n Abgerufen am 08.05.2024

steigt auch das Interesse von Automobilherstellern diese Sensoren im den eigenen Fahrzeugen zu verbauen [7].

2.4 Simulatoren: Autonomes Fahren

In der Entwicklung automatisierter und autonomer Fahrzeuge finden verschiedene Simulatoren Anwendung. In Anbetracht der Komplexität dieser Systeme, ist es nicht überraschend, dass eine Vielzahl von Simulatoren mit teilweise unterschiedlichen Zielen und Eigenschaften existiert. Daher ist es wichtig existierende Simulatoren voneinander abzugrenzen, um eine fundierte Entscheidung darüber treffen zu können, welcher Simulator in dieser Arbeit nachfolgend verwendet wird. Während einige Simulatoren in der Entwicklung auf einzelne Bereiche wie z.B. die Simulation von verschiedenen Trajektorien oder der Simulation von Bremswegen optimiert sind, wird hier ein Simulator mit größerem Umfang benötigt.
Für die Generierung von Sensordaten ist es essentiell, dass nicht nur das Fahrzeug im Verhalten zu einem anderen Verkehrsteilnehmer simuliert wird, sondern auch eine realistische und detailreiche Abbildung der Umwelt gegeben ist. Zu der Umwelt des Fahrzeuges gehören diverse andere Verkehrsteilnehmer, Wettereinflüsse, Straßenverhältnisse und weiteres. Zusätzlich simulieren sie nicht nur das Fahrzeug selbst, sondern auch seine verbaute Sensoren, welche meist in Position, Anzahl, etc. konfiguriert werden können. Allerdings gibt es auch bei der Simulation der Sensoren verschiedene Ansätze. So kann eine KI auf Basis unterschiedlich erzeugter Sensordaten trainiert werden. Ngo[15] kategorisiert diese Ansätze als:

1. Physikalische Modell Simulationen,
2. Ideale Modell Simulationen,
3. Datengesteuerte Modelle.

Physikalische Modell Simulationen sollen dabei eine Abbildung, die möglichst nahe an der Realität ist unter Beachtung physikalischer Begebenheiten, erschaffen. Aufgrund der sehr ausführlichen Umsetzung und Betrachtung dieser physikalischer Zusammenhänge entstehen entsprechend komplexe und aufwändige Simulationen mit hohem Rechen- und Speicherbedarf. Ideale Modell Simulationen hingegen versuchen ein angenähertes Modell der Realität abzubilden, um eine bessere Performanz innerhalb der Simulation zu gewinnen unter verzicht der Betrachtung bestimmter physikalischer Aspekte. Datengesteuerte Modelle, auf der anderen Seite, orientieren sich nicht an der physikalischen Funktionsweise des Sensors, sondern auf real gesammelten Daten. Für die Umsetzung dieser Arbeit, wäre es hier möglich Algorithmen zu entwickeln die entsprechende Datensätze um gewisse Fehlermodelle erweitern.

Allerdings ist es, wie bereits zuvor vorgestellt, nicht das Ziel bestehende Datensätze zu verändern sondern neue dynamisch zu generieren. Dies bietet den Vorteil, dass Fehler als direkte Reaktion der Umwelt simuliert werden können und eine verbundene KI live auf auf das Verhalten in Fehlerfällen getestet oder damit trainiert werden kann. Je nachdem wie der Simulator Sensoren simuliert müssen dabei verschiedene Aspekte betrachtet werden. Während bei physikalischen Modell Simulationen bereits physikalische Phänomene auftreten können, welche Ursache für später vorgestellte Fehler sein können, treten diese bei anderen Modell Simulationen nicht unbedingt auf. Darunter fällt z.B. das sogenannte Noise, wie in Abschnitt 2.3.2 aufgezeigt. Aufgrund dessen ist es wesentlich wichtig zu beachten wie der Simulator die Sensoren umsetzt, da dies die eigenen Anpassungen am Simulator beeinflussen wird. Zusätzlich ist dabei auch wichtig zu beachten, dass Ideale Modell Simulationen nicht in der Lage sind die Informationen einer RCS bereitzustellen.
Bei der Wahl des Simulators gilt es Vor- und Nachteile abzuwägen. Daher wurden einige Kriterien festgelegt anhand derer ein Vergleich stattfinden soll. Die Kriterien belaufen sich auf:

1. Manipulation,
   stellt dar wie effektiv die Implementierung der Fehlermodelle in den Simulator umgesetzt werden kann. Dabei ist es nötig, dass der Sensor selbst innerhalb des Simulators angepasst werden kann, sodass entsprechende Daten zur Simulationszeit, auf Basis des modellierten Fehlers generiert werden. So kann z.B. vorkommen, dass ein Sensor in der Simulation eine tatsächliche Blockade durch beispielsweise Schmutzpartikel vor der Linse erfährt.

2. Aktualität,
   beschreibt die Weiterentwicklung des Simulators. Dies bezieht sich darauf ob der Simulator regelmäßig von den Entwicklern mit Fehlerbehebungen und dem neusten Stand der Technik versorgt wird. Somit soll sichergestellt werden, dass eine langfristige Nutzung des Projektes möglich ist. Es kann so auch auf eine Kompatibilität mit zukünftig neu entwickelten Standards gehofft werden, als auch auf die Implementierung neuer Sensorarten, sollten sich hier entsprechende Neuerungen auf dem Markt etablieren.

3. Schnittstellen,
   für die entsprechende Weiterverarbeitung ist es relevant wie die Sensordaten des Simulators genutzt werden können. Es wäre wünschenswert, wenn eine Anbindung an gängige Programmiersprachen wie Python[9], Lua[10], C++ gegeben ist. Aber auch Schnittstellen zu anderen Standards wie Robot Operating System (ROS) und ASAM Open Simulation Interface (OSI) wären hier Wünschenswert.

[9] https://www.python.org Abgerufen am 01.06.2023
[10] https://www.lua.org Abgerufen am 01.06.2023

4. Umfang,
   beschreibt wie viele verschiedenen Szenarien in dem Simulator umgesetzt werden können. Hier wird sich schwerpunktmäßig am ASAM OpenDRIVE-Standard[11] und am ASAM OpenSCENARIO Standard[12] orientiert, welche es ermöglichen Szenarien, in Form von Straßenverläufen, vorzudefinieren und somit simulationsübergreifend zu teilen. Für die Umsetzung des Ziels ist dies besonders wichtig, da so Daten in verschiedenen Szenarien generiert werden können. So kann eine breite Abdeckung gewährleistet werden und die Basis für standardkonformes Testen sichergestellt werden.

5. Konfigurierbarkeit,
   beschreibt wie gut eine einzelne Simulation konfiguriert werden kann. Hierzu zählt z.B. der Umfang der Parametrisierung der Sensorik, welche maßgebend dafür ist, mit welchem Detailgrad ein realer Sensor modelliert werden kann. In der Praxis unterschieden sich dieselben Sensorarten durch verschiedene Aspekte, wie z.B. das FOV eines Radars je nach Einsatzbereich und der Aufgabe die der Sensor erfüllen soll. Daher ist es wichtig, dass dies konfiguriert werden kann um möglichst viele, unterschiedliche Sensoren imitieren zu können.

6. Sensor Simulation,
   beschreibt wie die Sensoren anhand der drei zuvor vorgestellten Ansätze umgesetzt sind, da dies einen entsprechenden Einfluss auf die zukünftige Arbeit und die Art der Implementierung der Fehler hat.

Simulator	Deep Drive	CARLA	Ansys AVxcelerate Sensors	SVL Simulator	Aurelion
Manipulation	Open Source	Open Source	Lizenziertes System	Open Source	Lizenziertes System
Aktualität	Mai 2018	Dezember 2022	Regelmäßige Updates	Januar 2022 Offiziell nicht weitergeführt	Regelmäßige Updates
Schnittstellen	PythonAPI	ROS-Bridge oder PythonAPI	Unbekannt	ROS-Bridge oder PythonAPI	Unbekannt
Umfang	kein OpenDRIVE Support	OpenDRIVE Support	eigener Szenarien Creator	OpenDRIVE Support	OpenDRIVE Support
Sensor Simulation	ideal	ideal	ideal	ideal	physikalisch

Tabelle 2.2 – Gegenüberstellung von einzelner Hauptmerkmale von verschiedenen Ideal Modell Simulatoren

[11] https://www.asam.net/standards/detail/opendrive// Abgerufen am 03.05.2024
[12] https://www.asam.net/standards/detail/openscenario/v200/ Abgerufen am 30.04.2024

In Tabelle 2.2 werden verscheidende Automobil-Simulatoren nach den zuvor vorgestellten Kategorien gegenübergestellt.

Für die Umsetzung in dieser Arbeit wurde sich für den CARLA-Simulator entschieden. Grund dafür ist eine aktive Weiterentwicklung der Software sowie seine Open Source Grundlage, welche es ermöglicht bestehenden Softwarecode frei einzusehen und zu verändern, während andere Open Source Simulatoren wie SVL Simulator oder Deep Drive nicht weiter aktualisiert werden. Bei Lizenesierten Systeme wie Ansys AVxcelerate Sensors und Aurelion kann eine Veränderung nur bedingt stattfinden, da sie nicht als Source Code ausgeliefert werden.
Eine Anwendung dieser Simulatoren wäre somit nur in angepasster Form möglich da generierte Daten im späteren Verlauf erst manipuliert werden könnten. Nachteilhaft an einer solchen Umsetzung ist eine schlechtere Performanz und die nicht vorhandene direkte Einflussnahme auf Objekte in der Simulation. Dies ist bei den Open Source Simulatoren nicht nötig, da hier entsprechende Anpassungen innerhalb der Simulator-Engine vorgenommen werden können und daher Livedatensätze generiert werden können und diese an das Verhalten der Simulation angepasst sind. Zusätzlich bietet der CARLA-Simulator mehrere Schnittstellen zum Abgreifen und weiterverarbeiten der Sensordaten. Auch bietet CARLA mit dem entsprechenden OpenDRIVE-Support eine große Anzahl an Szenarien die in dem Simulator dargestellt werden können. Da es sich hier allerdings um Ideale Modell Simulationen handelt wird davon ausgegangen, dass alles in einem ideal Zustand simuliert wird. Ein sehr markantes Beispiel ist dabei die Umsetzung der verschiedenen Sensoren innerhalb des Simulators. Bei der Betrachtung des Radar-Sensors ist erkennbar, dass das Prinzip von Radarwellen nur in sehr abstrahierter Form nachempfunden ist.
So basiert der Radar-Sensor auf dem Prinzip des Ray-Casting. Dabei werden vom Sensor aus einzelne Strahlen in die Welt ausgesendet. Darauf aufbauend werden mathematisch Schnittstellen mit Objekten in der Umgebung berechnet. Durch das Erkennen der Schnittstellen mit einem Objekt kann dies detektiert werden und dem Objekt zugehörige Informationen können gesammelt werden. Dazu zählen Informationen wie die Position des getroffenen Punktes und die Geschwindigkeit des Objektes. Dies zeigt somit deutlich, dass keine Berechnung der Geschwindigkeit anhand des Dopplereffektes erfolgt. Auch Informationen wie die RCS eines Objektes können damit nicht real bestimmt werden, was explizit bedeutet, dass ein Radar in einer idealen Modell Simulation keine RCS-Informationen generieren kann bzw. diesen Wert auf Basis der Kenntnis über das getroffene Objekt synthetisch bestimmen muss.
Gegenläufig bietet dieser Ansatz allerdings die Option generierte Daten direkt mit sogenannten Labeln zu versehen. Dabei werden die getroffenen Objekte in verschiedene Kategorien eingeteilt, wie z.B. Fahrzeug, Passant, Verkehrsschild oder Ähnliches. So können Informationen über das getroffene Objekt, wie z.B. den Typus des Objektes direkt mit gesammelt werden. Dies vereinfacht die Nachbearbeitung der Daten, da es nicht mehr nötig ist einzelne Punkte in die verschiedenen Kategorien händisch einzuteilen. Dies kann innerhalb einer Simulation bereits automatisch erfolgen. Zusätzlich muss allerdings auch hier bedacht werden, dass weniger Daten als in der Realität generiert werden

können. Da die Simulatoren auf Basis des Ray-Casting arbeiten ist es wichtig die Menge der gesendeten Rays auszubalancieren. Viele gesendeten Rays führen zu einer Reduktion der Performanz des Simulators, liefern aber eine bessere Auflösung in der Betrachtung der Umgebung. Daher ist es für entsprechende Simulationen wichtig die Parameter auszubalancieren und für die eigene Anwendung zu optimieren.
Bei der Umsetzung der Fehlermodelle ist es daher nötig diese realitätsnah zu halten. Wie einzelne Modelle hier angepasst und umgesetzt sind können den zugehörigen Kapiteln entnommen werden. Wichtig ist hier auch noch zu betonen, dass innerhalb der Simulation alle Sensoren bestimmten Limitierungen unterliegen. Die in dieser Arbeit betrachteten LiDAR-Sensoren basieren auf demselben Prinzip des Ray-Casting und unterliegen daher auch den bereits vorgestellten Aspekten.

2.4.1 CARLA-Simulator

Damit eine Umsetzung der Modelle erfolgen kann ist es nötig zu verstehen wie der CARLA-Simulator aufgebaut ist und wie mit ihm interagiert werden kann. Für ein besseres Verständnis der CARLA-Struktur wurde nachfolgend in Abbildung 2.3 der Aufbau von CARLA zusammen gefasst:

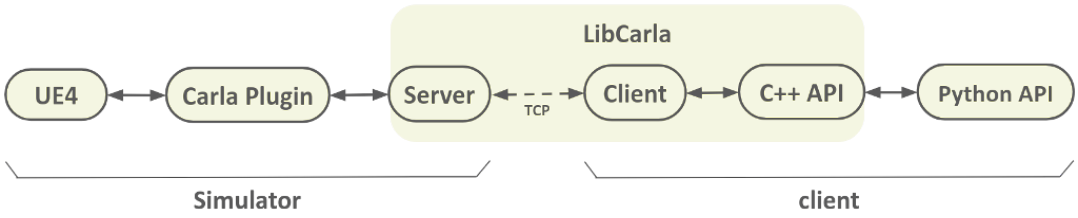


Abbildung 2.3 – Aufbau des Carla-Simulators [13]

Der CARLA-Simulator basiert auf der Unreal Engine[14] 4.26 eine Grafik-Engine die ursprünglich für die Spielentwicklung entwickelt worden ist. Diese Unreal Engine Version wird mit einem CARLA spezifischen Plugin erweitert um die gewünschten Funktionen und Interaktionen der Simulationsobjekte in der Unreal Engine umzusetzen. Zu den entsprechenden Objekten gehört alles was die Umwelt sowie die Simulation widerspiegeln soll wie z.B. Häuser, Verkehrszeichen, Sensoren, Fahrzeuge und vieles weiteres. Erweitert wird diese Serverstruktur zusätzlich mit Teilen der sogenannten LibCarla. Die LibCarla ist eine Library(Software-Bibliothek) und stellt das Bindeglied zwischen dem Simulationsserver sowie den damit verbunden Clients dar. Zusätzlich bildet die LibCarla den clientseitigen Part der Client-Server-Kommunikation ab. Clientseitig wird die LibCarla zusätzlich um eine C++-Application Interface (API) erweitert welche in der Lage ist mit Python zu interagieren. Somit wird erreicht, dass der Server über einzelne Python-Scripts angesprochen werden kann.

[13] https://carla.readthedocs.io/en/0.9.7/dev/how_to_add_a_new_sensor/ Abgerufen am 06.06.2023
[14] https://www.unrealengine.com/de abgerufen am 03.05.2024

Die in der Simulation verwendbaren Sensoren finden sich in verschiedener Art innerhalb der Pipeline wieder. Innerhalb des CARLA-Plugins findet sich sämtliche Logik der Sensoren und ihrer Art der Datenerfassung wieder. Die LibCarla nutzt diese Daten und stellt sicher, dass die entsprechenden Informationen an die einzelnen Clients weitergeleitet werden. Ein Client wird in diesem Szenario durch ein entsprechendes Python-Script, welches dessen Logik enthält, dargestellt.
Die Umsetzung der Fehlermodelle würde daher in erster Linie innerhalb des CARLA-Plugins durch die Manipulation der dortigen Sensoren erfolgen.
Zusätzlich zu der Integrierung der Fehler in den Server müssen auch Einstellungsmöglichkeiten sichergestellt werden. Die entsprechende Parametrisierung von Fehlermodellen erfolgt zu einem späteren Zeitpunkt. Um die Struktur und einfache Nutzbarkeit des Simulators zu gewährleisten soll die genaue Parametrisierung für einen spezifischen Fall auch über das Python-Script erfolgen können. Während für eine reguläre Simulation im Python-Script Informationen bezüglich der Position des Sensors sowie weitere Informationen wie z.B. das FOV des Sensor an den Server übermittelt werden. So sollen auch Parameter für den Fehler übermittelt werden können. Daher wird eine Anpassung der LibCarla für einzelne Fehlermodelle ebenfalls notwendig.
Hierbei kann sich an den bereits implementierten Einstellungsmöglichkeiten orientiert werden, damit auch projektfremde Personen einen einfachen Einstieg in das Projekt erhalten. Durch die Anpassung der Datengenerierung während der Simulation muss grundlegend erstmals keine Anpassung an der Informationsübertragung von Server zu Client vorgenommen werden. Allerdings wurde im vorherigen Kapitel bereits ein Vorteil dieser Simulatoren, das synthetische Labeling, aufgelistet. Damit die Information über ein Label auch verarbeitet werden kann, muss diese Information aus dem Server zum Client überführt werden. Dies bedeutet, dass alle zusätzlichen Informationen die gesammelt werden sollen und nicht bereits vom CARLA-Simulator standardmäßig übertragen werden, in die Pipeline eingepflegt werden müssen und erfordern somit eine entsprechende Anpassung der LibCarla.

2.4.2 ScenarioRunner

Da Straßenverkehr besonders komplex ist und dort sehr viele verschiedene Situationen entstehen können, müssen diese für Simulationen kontrollierbar gemacht werden. Um dies zu erreichen werden sogenannte Szenarien erstellt. Diese Szenarien beinhalten die Informationen über die Straßenführung der Umgebung, das Fahrverhalten von anderen Verkehrsteilnehmern etc. Um dies einheitlich und einfach austauschbar zu machen gibt es hier das sogenannte ASAM OpenSCENARIO Format[15].
Viele Szenarien werden dabei auch in der Realität getestet. Allerdings können manche Szenarien aus unterschiedlichen Gründen nicht in der Realität getestet werden. Üblicherweise werden Gefahrenszenarien wie Unfälle, die ein hohes Risiko für Verkehrsteilneh-

[15] https://www.asam.net/standards/detail/openscenario/ Abgerufen am 03.06.2024.

mer bedeuten würden und zu Sachbeschädigungen mit hohen Schadenwert führen würde, ausschließlich simulativ getestet. Allerdings fangen die einzelnen Szenarien schon deutlich kleiner an. So sind bereits Fahrten auf einer geraden Straße ohne autonome Fahrfunktionen erfolgreich getestet werden. So übernehmen aktuelle Level-3-Systeme wie ALKSs die Steuerung der Quer- und Längsbewegungen des Fahrzeuges für eine definierte Fahrsituation (ODD). Während sich Fahrzeug innerhalb dieser ODD befindet kann der Fahrer des Fahrzeuges sich vom Straßenverkehr abwenden. Innerhalb einer ODD sind Umgebungsbedingungen definiert die den Rahmen einer Situation abstecken. Die Situation ist dabei definiert über verschiedene Bedingungen wie den Zustand der Straße, Tageszeit oder andere Charakteristiken durch des Fahrzeuges, des Verkehrs oder der Straße. Für Simulationen gibt es hier das ASAM OpenODD Format[16].
Damit ein ALKS kein Risiko für andere Verkehrsteilnehmer darstellt muss dieses ein gewisses Niveau erreichen können.
Hier greift eine Vorschrift der United Nations Economic Commission for Europe (UNECE), die sogenannte „UN Regulation No. 157 - Automated Lane Keeping Systems (ALKS)“ [16]. Dort wird definiert welche Szenarien ein ALKS erfolgreich meistern muss damit es für den Straßenverkehr zugelassen werden kann. Um zu erkennen welche Szenarien ein ALKS besteht oder nicht werden dort verschiedene Key Performance Indicator (KPI) genannt. Eine dieser KPI ist beispielsweise die Time-To-Collision (TTC). Diese beschreibt die Zeit die verstreichen müsste damit zwei Verkehrsteilnehmer kollidieren, wenn beide keine Geschwindigkeitsveränderung erfahren. Die Time Headway (THW) hingegen definiert die Zeit die eine Fahrzeug benötigt bis es an der Position eines vorausfahrenden Fahrzeuges ankommt. Für jedes Szenario sind nun Grenzwerte definiert die die einzelnen KPI nicht unterschreiten dürfen.
Da diese Szenarien eine hohe Wichtigkeit im Bereich der Simulation von autonomen Fahrzeugen einnehmen gibt es für den CARLA-Simulator den ScenarioRunner. Dies bedeutet, dass er andere Verkehrsteilnehmer in der Umgebung initialisiert und diese entsprechend dem Szenario durch die Welt steuert. Dort können nun verschiedene bereits vorgefertigte Szenarien verwendet werden. Diese überschneiden sich zu großen Teilen auch bereits mit den Szenarien die von der UNECE für eine Zulassung gefordert werden. Zusätzlich unterstützt der ScenarioRunner aber auch das OpenSCENARIO Format, worüber selbst erstellte Szenarien geladen und simuliert werden können. Diese Verbindung aus dem ScenarioRunner und dem CARLA-Simulator wird so z.B. bereits in der Praxis genutzt um KPIs in der Simulation zu testen. Ein Beispiel dafür ist das von TraceTronic[17] entwickelte Programm ecu.test[18].
Da der ScenarioRunner aktuell nur das OpenSCENARIO Format unterstützt und es keine Unterstützung für das ASAM OpenODD Format gibt ist es notwendig diese Aspekte bei der Untersuchung von ALKS auf anderem Wege zu betrachten. Der ScenarioRunner

[16] https://www.asam.net/standards/detail/openodd/ Abgerufen am 03.06.2024.
[17] https://www.tracetronic.de Abgerufen am 30.04.2024
[18] https://www.tracetronic.de/produkte/ecu-test/ Abgerufen am 30.04.2024

und CARLA besitzen allerdings einige Möglichkeiten die Bedingungen einer ODD in der Simulation zu betrachten. So gibt z.B. Abbiegeszenarien auf verschiedenen Straßen mit unterschiedlichem Belag und die Szenarien sind mit unterschiedlichen Tageszeiten versehen. Somit ist eine Berücksichtigung einiger Umgebungsbedingungen aus der ODD möglich.

2.4.3 Visualisierung und Speichern der Daten

Radar- und LiDAR Daten können über einen von CARLA bereitgestellten Weg visualisiert werden. Dies bietet die Möglichkeit die Daten live während der Simulation zu betrachten und somit nicht auf ein externes Programm angewiesen zu sein. Diese Abbildungswege sind bereits in den von CARLA bereitgestellten beispielhaften Python-Scripts eingebunden. Nachfolgend in den Abbildung 2.4, Abbildung 2.5 und Abbildung 2.6 können visualisierte Radardaten Daten erkannt werden. Radardaten werden dabei in die Umgebung gerendert und mit einer zwischen blauen, weißen und roten Schwankenden Farbe versehen.
Die Position des Punktes in den nachfolgenden Abbildungen stimmt dabei mit dem Punkt der Detektion in der Simulationswelt überein. Aufgezeigt wird durch die Skalierung der Farbstärke die relative Geschwindigkeit des Messpunktes zum Radar-Sensor ausgedrückt. Je stärker die Intensität der Farbe, desto größer ist die Differenz der Geschwindigkeiten.

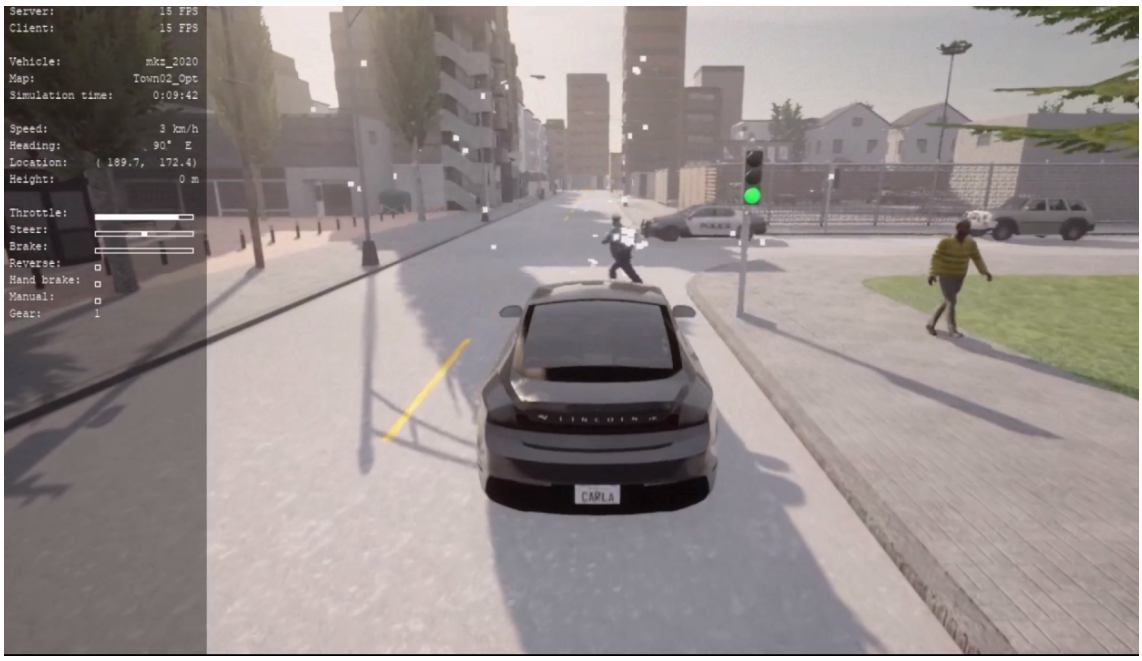


Abbildung 2.4 – Visualisierung von Radardaten bei identischer Geschwindigkeit

Abbildung 2.4 zeigt die Detektion eines Fußgänger der orthogonal die Fahrtrichtung kreuzt. Somit hat der Fußgänger keine Geschwindigkeit entlang der Fahrtrichtung und besitzt so für den Radar-Sensor die gleiche Geschwindigkeit wie das stehende Fahrzeug.

Des weiteren wird in Abbildung 2.5 ein dem Sensor entgegenkommendes Fahrzeug sowie Gebäudewände, die sich aus der Perspektive des Sensors dem Fahrzeug nähern und so eine geringere Geschwindigkeit als der Sensor besitzen, gezeigt.

Abbildung 2.5 – Visualisierung von Radardaten bei relativer niedrigerer Geschwindigkeit

Sich hier annähernde Objekte werden mit den dort zu erkennenden roten Punkten versehen. In dieser Darstellungsform werden auch statische Objekte wie die Gebäudewände mit einer Geschwindigkeit versehen, da die Geschwindigkeit relativ zum Radar-Sensor betrachtet wird und sich so statische Objekte auf den Sensor bewegen. Blaue Punkte hingegen verdeutlichen, dass sich ein Objekt schneller als der Sensor bewegt. So ist in Abbildung 2.6 ein beschleunigendes Fahrzeug zu sehen. Da das Fahrzeug zum Zeitpunkt der Aufnahme nur eine geringe Beschleunigung, relativ zum Radar-Sensor, besitzt sind die Punkte hier auch nur leicht blau gefärbt.
Hingegen werden LiDAR-Daten, gezeigt in Abbildung 2.7, in einer Top-Down-Ansicht geplottet. Dabei wird der LiDAR-Sensor zentral in der Mitte der gerenderten Umgebung angenommen. Die einzelnen Detektionspunkte werden nun abhängig von dieser Position gerendert. Da LiDAR-Daten keine Geschwindigkeitsinformationen beinhalten, werden die Punkte hier einheitlich weiß dargestellt und es findet keine farbliche Differenzierung statt.
Auch muss hier berücksichtigt werden, dass dreidimensionale Daten in einer zweidimensionalen Ansicht dargestellt werden. Somit kann keine Information über die Z-Achse aus dieser Art der Darstellung entnommen werden.
Für das Speichern der Daten wurde sich an in der Praxis gängigen Datenformaten orientiert, da CARLA nativ keine Möglichkeit zum Speichern der Punktwolken bietet. Für das Training von autonomen Fahrzeugen ist es sehr nützlich, wenn reale Daten neben

Abbildung 2.6 – Visualisierung von Radardaten bei relativer höherer Geschwindigkeit

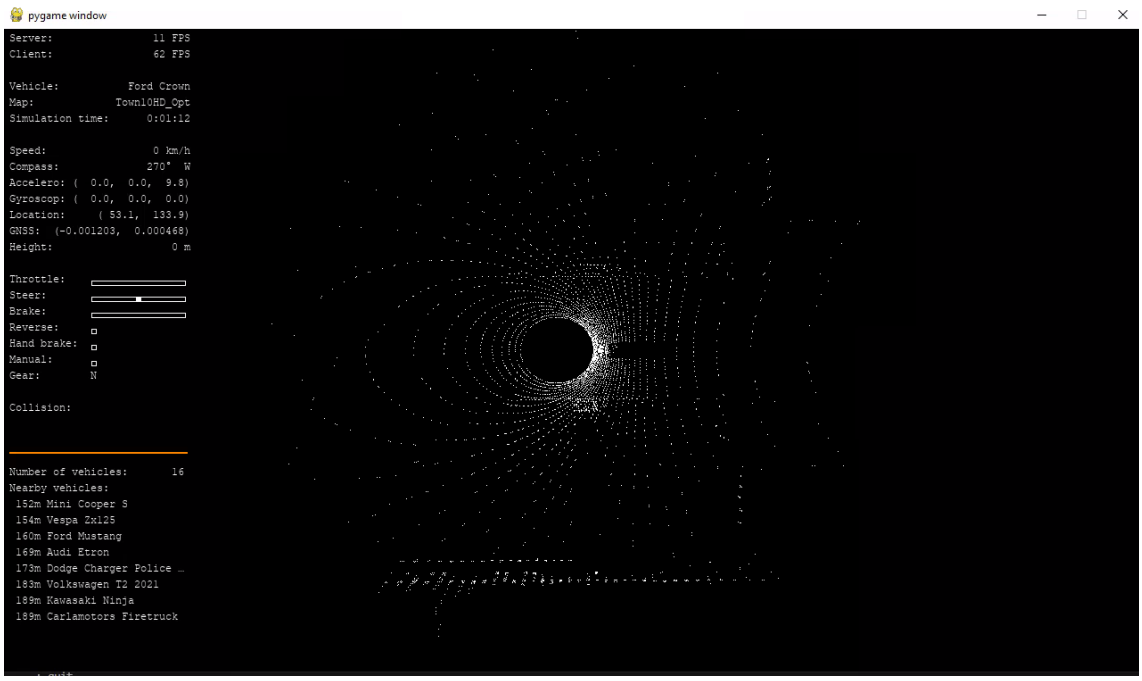


Abbildung 2.7 – Visualisierung von Radardaten bei relativer höherer Geschwindigkeit

den vorgestellten Simulatoren verwendet werden können. So veröffentlichte RadarScenes [17] einen Datensatz in dem aufgezeichnete Radardaten aufbereitet und händisch mit Labeln versehen worden sind. Dabei wurden Videoaufnahmen der einzelnen Fahrten mit den Radardaten verbunden und die Radardaten wurden manuell mit entsprechenden Labeln versehen.
Der bei dieser Veröffentlichung erstellte Datensatz wurde dabei im Hierarchical Data Format Version 5 (HDF5) abgespeichert. Für die Datenanalyse entwickelten die Autoren

einen eigenen RadarScenes-Viewer der die einzelnen Radardaten frameweise in ein XY-Koordinatensystem darstellen kann. Eine beispielhafte Darstellung eines solchen Datensatzes kann der nachfolgenden Abbildung 2.8 entnommen werden.

Abbildung 2.8 – Aufbau des RadarScenes Tools

In dieser ist zu erkennen, dass sämtliche Messpunkte des Radars relativ zum Fahrzeug positioniert werden. Auch sind die Geschwindigkeiten der Punkte relativ zum Fahrzeug durch kleine Striche an den Punkten dargestellt. Je länger ein solcher Strich ist, desto schneller bewegt sich der Punkt relativ zum Fahrzeug. Zusätzlich werden am rechten Rand des RadarScenes-Viewers weitere Informationen zu einzelnen Punkten dargestellt, die im Koordinatensystem ausgewählt werden. Das Ganze wird erweitert durch den Frame einer Videoaufnahme, die mit den Radardaten in Verbindung gebracht werden kann. Zusätzlich werden die einzelnen Radarpunkte in verschiedenen Farben, den Labeln zugeordnet, dargestellt, um direkt zu sehen um was für ein Objekt es sich an dieser Position handelt. Die verfügbaren Objekttypen und die zugehörigen Farben können der Tabelle 2.3 entnommen werden.
Für die Verwendung des RadarScenes-Viewers wurde daher ein Skript zum speichern der Daten im vorgestellten HDF5-Format erstellt

Label	Farbe
PKW	Violet
Großes Fahrzeug	Orange
LKW	Grün
Bus	Hellgrün
Zug	Braun
Fahrrad	Türkis
Motorrad	Hellblau
Fußgänger (einzeln)	Gelb
Fußgänger (Gruppe)	Pink
Tier	Blau
Anderes	Indigo
Statisch	Grau

Tabelle 2.3 – Gegenüberstellung der einzelnen Labels von RadarScenes sowie deren Farbe

3 Umgebungsbetrachtung

Das Ziel dieses Kapitels ist es einzelne Fehlerursachen und das daraus folgende Fehlverhalten der zuvor vorgestellten Sensoren genauer zu betrachten. Aufbauend auf diesem Ansatz werden Fehler mit ähnlichen Folgen später gruppiert und modelliert. Dabei werden die betrachteten Fehlerursachen in die nachfolgenden Kategorien, die bereits in Kapitel 2.2 vorgestellt worden sind, unterteilt.
Diese Kategorien werden zunächst für Radar- und LiDAR-Sensoren gemeinsam betrachtet, da einige Aspekte sich entsprechend überschneiden. Unterschiedliches fehlerhaftes Verhalten oder Fehlerursachen, die nur einen der beiden Sensoren betreffen, werden anschließend in getrennten, sensorspezifischen Kapiteln erläutert.

3.1 Interne Safety

Wie bereits in Kapitel 2.2.1 erwähnt kann dabei ein Ausfall auf Hardware- als auch auf Software/Firmware-Ebene erfolgen. Aufgrund des unvorhersehbaren Verhaltens von Software- und Firmwarefehlern werden diese im weiteren Verlauf nur bedingt betrachtet. Bei Softwarefehlern kann es sich z.B. um Fehler innerhalb eines Algorithmus im Sensor handeln, welcher somit fehlerhafte Daten generiert. Auch werden Designfehler die während der Hardwaresynthese oder der Softwareimplementierung entstehen nicht weiter betrachtet.
Der Schwerpunkt des Kapitels liegt daher auf Alterungs- und Abnutzungsprozessen am Beispiel „Wackelkontakt“, welche einen Einfluss auf den Sensor und somit auch auf dessen Funktionsweise haben. So kann es im Laufe der Zeit zu einer Korrosion der elektronischen Kontakte innerhalb des Sensors kommen. Durch solche Korrosionen können unter anderem Wackelkontakte entstehen, die entsprechende Auswirkungen auf die Funktionsfähigkeit des Sensors haben. Dieser Alterungsprozess wird unter anderem durch eine hohe Luftfeuchtigkeit verstärkt. Daher gilt es bei der Betrachtung von Alterungsprozessen immer auch die Nutzungsumgebung eines Sensors zu berücksichtigen. Hartung [18] definiert diese Umwelteinflüsse als Klimakomponenten und fasst deren Auswirkungen und typische Schäden wie in Tabelle 3.1 gelistet zusammen.
Die in Kapitel 3.1 vorgestellten Auswirkungen und Schäden sind dabei nicht immer vollumfänglich in jedem einzelnen Bereich ausgeprägt. Einzelne Alterungsprozesse wirken sich je nach Bauteil und dessen Umgebung anders aus. Auch ist es wichtig zu betrachten in welcher Umgebung ein Sensor verwendet wird, da unterschiedliche Alterungsprozesse von unterschiedlichen klimatischen Bedingungen verstärkt werden. In [19] wird aufgezeigt wie sich Alterungsprozesse auf Bauteile auswirken, die in einer klimatisch gesteuerten Umgebung[1] gelagert werden.
Generell gilt, dass klimatische Einflüsse sich im Feld deutlich stärker auswirken als in

[1] Umgebungen in denen klimatische Bedingungen, wie Temperatur u.ä. bekannt und steuerbar sind.

Klimakomponente	Hauptauswirkung und Typische Schäden		
Temperatur	Hoch	Oxidation, Rissbildung, Ausdehnung	Versagen der Isolationsfähigkeit, Mechanisches Versagen
	Niedrig	Versprödung, Schrumpfen	Verstärkter Verschleiß aufgrund thermischer Ausdehung
Luftfeuchtigkeit	Hoch	Ad- und, Adsorption, von Feuchtigkeit Quellungen, Korrosion, Elektrolyse	Mechanisches Versagen, Risse
	Niedrig	Austrockung, Versprödung	Mechanisches Versagen, Risse
Temperaturwechsel	Temperaturschock		Mechanisches Versagen, Rissbildung, Schädigung von Abdichtungen
Sonnenstrahlung	Chemische, physikalische Reakitonen, Versprödung, Erwärmung		Versagen von Isolation
Sand und Staub	Abrieb, Festfressen, Vermindern der Wärmeleitfähigkeit		Erhöhter mechanischer und elektrischer Verschleiß
Korrosive Atmosphären	Minderung von Isolierfähigkeit, Leitfähigkeitserhöhung		Erhöhter mechanischer und elektrischer Verschleiß
Mechanische Bewegung	Mechanischer Ermüdung, Resonanzanregung		Mechanisches Versagen, verstärkter Verschleiß

Tabelle 3.1 – Auswirkungen und Schäden von Klimatischen Einflüssen auf ein Bauteil [18]

der Lagerung, da hier kein Einfluss auf die Umgebungsbedingungen genommen werden kann. Wie in der Tabelle 3.1 aufgezeigt, können Alterungseffekte gegenläufig sein. Angenommen zwei gleiche Bauteile werden in unterschiedlichen Umgebungen verbaut: Bauteil A in einer warmen Umgebung und Bauteil B in einer kälteren Umgebung. Diese unterschiedlichen Belastungen durch die Temperatur begünstigen unterschiedliche Alterungseffekte. So kann z.B. der bereits erwähnte Wackelkontakt durch solche Alterungsprozesse entstehen und in bestimmten Regionen mit der selben Ursachenverkettung häufiger auftreten, als in anderen Gebieten.
Aber auch andere Effekte wie eine Materialermüdung an Leitungsverbindungsstellen kann langfristig ursächlich für einen Wackelkontakt sein. Aus technischer Sicht spiegelt der Wackelkontakt wieder, dass die Verbindung zwischen Sensor und nachfolgenden Systemen unterbrochen ist. Im schlimmsten Fall bedeutet dies, dass zeitweilig oder vollständig Daten des Sensors nicht an nachfolgende Systeme übertragen werden. Aus funk-

tionaler Sicht besteht dann ein sogenannter Datenübertragungsfehler. Allerdings kann ein solcher Fehler nicht nur durch entsprechende Alterungsprozesse im System hervorgerufen werden, auch weitere Faktoren wie eine mindere Materialqualität oder z.B. ein fehlerhafter Einbau, können einen solchen Wackelkontakt und damit verbundene Datenübertragungsfehler verursachen.
Wie bereits zuvor vorgestellt werden Softwarefehler aufgrund ihrer komplexen Ursachenverkettungen nicht vertiefend in dieser Arbeit betrachtet. Da allerdings Datenübertragungsfehler einen großen Teil der vorgestellten Fehler darstellen und diese sich auch anders Auswirken können, wird an dieser Stelle kurz auf weitere Möglichkeiten eingegangen. Wichtig ist dabei zu erwähnen, dass der nachfolgend betrachtete Datenübertragungsfehler nicht nur durch einen defekten Sensors auftreten kann. Auch ist es möglich, dass der Fehler bei einem funktionierenden Sensor auftritt, da der Fehler innerhalb der Software des Sensors liegt werden diese Fehler auch in diese Kategorie eingeteilt. Bei dieser Ausprägung eines Datenübertragungsfehlers geht es hier um die zeitliche Verzögerung, oftmals als *Lag*(Übertragungsverzögerung) bezeichnet. In [20] wird diesbezüglich folgende Aussage getätigt: „... However, the impact of sensor failures or network attacks in the channel may lead to data transmission time-delay and random packet drops“ [20]. Die dabei angesprochenen *random packet drops* können dabei mit dem gedanklichen Modell des bereits vorgestellten Wackelkontaktes abgedeckt werden. Der angesprochene *time-delay* muss allerdings getrennt davon berücksichtigt werden. Die genaue Betrachtung dafür erfolgt in Kapitel 4 sowie den entsprechenden Unterkapiteln.

3.2 Externe Safety

Für die Betrachtung der externen Safety eines Sensors werden, die bereits in Kapitel 2.2.2 erwähnten Einflüsse genauer untersucht. Es wird dabei davon ausgegangen, dass der Sensor selbst zu jedem Zeitpunkt fehlerfrei funktioniert(seiner Definition entsprechend). Bei dieser Analyse werden verschiedene Fehlerursachen untersucht. Ein möglicher Einfluss auf die generierten Daten eines Sensors ist der Installationsprozess des Sensors. In vielen Fällen findet eine Installation noch zu großen Teilen nicht automatisiert statt. Da in dieser Arbeit nur auf einen Teil möglicher Ursachen eingegangen wird, werden die abgeleiteten Modelle abstrahiert und es daher auch nicht nötig sämtliche Fehlerquellen zu finden. Ein praktisches Beispiel für einen Fehler innerhalb des Installationsprozesses eines Sensors ist die Rückrufaktion des Toyota Yaris in den Jahren 2020 und 2021. Dort wurden Radar-Sensoren innerhalb des Fahrzeugs verbaut welche nicht aktiv geschallten waren und dadurch nicht funktionsfähig waren. Dabei kann die Auswirkung des zugrundeliegenden Fehlers mit der eines Datenübertragungsfehlers verglichen werden.
Diesbezüglich schreibt der Nachrichtendienst Focus „Das PCS wäre also bei einer Inaktivität nicht in der Lage, ein Hindernis zu erkennen “ [21]. Für nachfolgende Systeme

bedeutet ein solcher Fehler dieselben Auswirkungen wie der bereits vorgestellte Datenübertragungsfehler. Während bei diesem Fehler reguläre Radar-Daten generiert, aber nicht weitergeleitet werden, gilt hier, dass die Daten gar nicht erst erzeugt werden. In beiden Fällen werden nachfolgende Systeme nicht mit den benötigten Daten versorgt.
Ein weiterer Fehler der schwerwiegende Folgen haben kann, ist die nicht sachgemäße Befestigung des Sensors. Durch eine inkorrekte Befestigung in Verbindung mit äußeren Effekten, kann es zu einer Verschiebung des FOV kommen, da der Sensor sich innerhalb seiner Befestigung bewegen kann. Die Wichtigkeit über die Positionierung und Ausrichtung eines Sensors wurde bereits in Kapitel 2.1.2 erläutert. Kurz zusammengefasst, kann es hier zu einem deutlich verzögerten oder sogar zum Nichterkennen von relevanten Objekten kommen. Dabei wird hier keine genaue Ursache für einen solchen Fehler festgelegt, da ein solcher Fehler oftmals durch die Verkettung vieler einzelner Umstände auftritt. Fehlerursachen wie zum Beispiel unzureichendes Anziehen der Schrauben, Verwendung defekter Schrauben aber auch designtechnische Fehlerursachen sind hier denkbar.
Ein solches Fehlerbild kann sich aber auch im Laufe der Zeit abbilden. Wie zuvor vorgestellt, können Alterungsprozesse einen Einfluss auf die Funktionsfähigkeit eines Systems haben. In dem gerade betrachteten Szenario einer Verschiebung des Sensors ist es weiterhin denkbar, dass die Korrosion verschiedener Bauteile, wie z.B. Schrauben, langfristig einen ähnlichen Effekt hervorrufen kann. Eine Verschiebung des Sensors kann dazuführen, dass sich das FOV des Sensors mit der Karosserie des Fahrzeuges überschneidet. Dies ist bei Radar-Sensoren ein größeres Problem, da LiDAR-Sensoren aktuell noch schwerpunktmäßig auf dem Fahrzeugdach montierte 360°-Sensoren sind. Wie in Kapitel 2.3.3 gezeigt, widmet sich momentane LiDAR Forschung schwerpunktmäßig der Verkleinerung der Sensoren mit dem Ziel diese kostengünstiger und besser integrierbar (ähnlich wie Radar Sensoren) zu bekommen. Durch die darauffolgende Anpassung ihres Einsatzortes, ist es wahrscheinlich, dass dieses Problem auch in Zukunft bei LiDAR-Sensoren an Relevanz gewinnen wird.
In dem bereits vorgestellten Fall eines nicht korrekt befestigten Sensors, kann es auch zu einer Blockade dessen kommen. Dabei müssen zwei verschiedene Arten von Blockierungen unterschieden werden. In der Englischen Literatur werden verschiedene Blockadearten wie nachfolgend beschrieben:

1. Blockage
   Die vollständige Blockierung eines Sensors, verbunden mit einem vollen Datenverlust im Bereich der Blockade. Beispielhaft wäre hier eine starke Verschmutzung des Sensors durch z.B. Schlamm.

2. Obscuration
   Die Verschleierung/Verdeckung von diversen Aspekten durch blockierende Objekte, die aber teilweise noch von den ausgesendeten Wellen durchdrungen werden können. Wie z.B. kleinere Verschmutzungen vor dem Sensor.

Im weiteren Verlauf wird in beiden Fällen dennoch von einer Blockade gesprochen, da in der Simulation später nur nicht durchdringbare Objekte verwendet werden. Die genaue Einteilung erfolgt dabei immer durch die entsprechenden Auswirkungen und sind in den meisten Fällen der „Obscuration“ zuzuordnen. Ab einem gewissen Grad der Obscuration verändert sich der entsprechende Zustand in eine „Blockage“, da nun eine vollständige Verdeckung des LiDAR- oder Radar-Sensors vorliegt die nicht mehr von den Wellen durchdringbar ist.
In dem erwähnten Fall der Verschiebung eines Sensors müssen verschiedene Parameter berücksichtigt werden. Darunter fällt unter anderem das Material der Karosserie. Denn, ob und mit welcher Intensität die Wellen durch das blockierende Objekt dringen können, ist abhängig von seinem Material.
Somit finden in einem solchen Fall vorrangig Änderung in der effektiven Reichweite, oder aber auch in der Möglichkeit kleinere Objekte zu erkennen, statt (siehe Murad [9]). Bei Materialien die nicht von den Wellen durchdrungen werden können, ist dann von einem Totalausfall der Sensorfunktion auszugehen. Dabei kann es dazu kommen, dass der Sensor die Blockade selbst als ein nahestehendes, festes Objekt erkennt. Dementsprechend werden viele nahe Punkte detektiert, die relativ gesehen zum Sensor keine Geschwindigkeitsveränderung erfahren und sich somit mit derselben Fahrgeschwindigkeit bewegen wie der Sensor. Allerdings kann eine solche Blockade nicht nur durch die Verschiebung des Sensors auftreten. Wie bereits früher vorgestellt, können auch andere Aspekte einen indirekten Einfluss auf den Sensor nehmen. Für die Betrachtung von Blockadezuständen eines LiDAR- und Radar-Sensors müssen dabei die Wetterbedingungen berücksichtigt werden. Dies bedeutet, dass Wetter- und Terrainbedingungen wie z.B. Regen, Schnee, Eis, Schlamm und Sandstaub einen Einfluss auf die Sensoren nehmen. Dabei ist anzunehmen, dass die entsprechenden Wetterzustände unterschiedliche stark ausgeprägte Schichten innerhalb des FOVs des Sensors bilden. Rivero [22] beschreibt dabei 4 schwerpunktmäßige Effekte die bei einem LiDAR-Sensor auftreten können:

1. Blindspots innerhalb des FOV,
2. Reduzierung der effektiven Reichweite,
3. Veränderungen des FOV,
4. Messungenauigkeiten.

Aufgrund der sich ähnelnden physikalischen Funktionsweisen beider Sensoren, vorgestellt in Kapitel 2.3.1 und unter Betrachtung der in [9] vorgestellten Effekte bei einer Blockade durch Schlamm, kann angenommen werden, dass einzelne Effekte vollständige und andere zumindest teilweise auch für Radar-Sensoren gelten. So zeigt Yeh [23], dass auch Radar-Wellen nicht konsistent feste Objekte durchdringen können. Wie in [24], [25] und [22] beschrieben kann die Funktionsleistung eines LiDAR-Sensors durch die Art des Schmutzes unterschiedlich stark auswirken kann. Auch wird dort aufgezeigt, dass

mit einem steigenden Verschmutzungsgrad, gemessene physikalische Parameter wie die Position eines Objektes ungenauer werden. Die Reduzierung der effektiven Reichweite würde in einem solchen Fall bedeuten, dass Objekte in einer größeren Entfernung entsprechend verspätet erkannt werden. In Rivero [22] werden dafür keine expliziten Gründe genannt. Eine mögliche Erklärung könnte aber ein durch die Blockade verursachtes steigendes *Noise-Levels* sein. Im nachfolgenden Kapitel wird weiter auf den Einfluss von Noise-Level auf die vorgestellten Sensoren eingegangen.

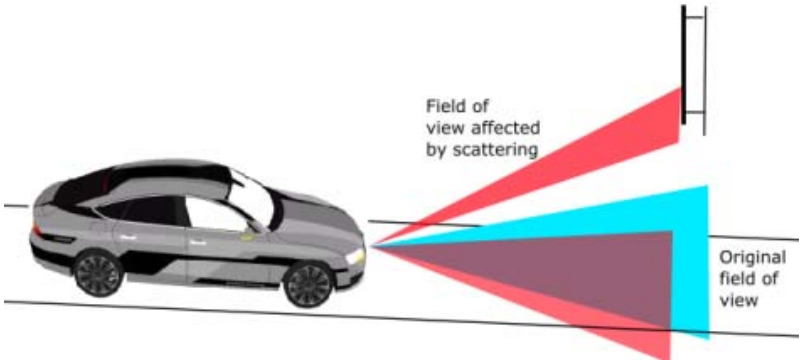


Abbildung 3.1 – Mögliche Veränderung des FOV durch eine Verschmutzung des LiDAR ([22])

Abbildung 3.1 visualisiert beispielhaft wie sich das FOV eines LiDAR-Sensors durch einen Schmutzpartikel einer Verschmutzung verändern kann. Dort wird aufgezeigt, dass sich das FOV durch die Blockade aufteilen kann und so Blindspots innerhalb des eigentlichen FOV entstehen. Durch eine solche Aufteilung des FOV kann es zu einem verspäteten Erkennen von Umgebungsobjekten kommen.
In Hasirlioglu [26] werden die Effekte von Regen auf LiDAR- und Radar-Sensoren speziell untersucht. Dabei wurden Versuche durchgeführt in denen die Regenintensität konstant gesteigert wurde. Dafür wurden zwischen den Sensoren Schichten eingefügt die mit Regen versehen werden konnten. Eine beispielhafte Skizze über den Aufbau kann der nachfolgenden Abbildung 3.2 entnommen werden.

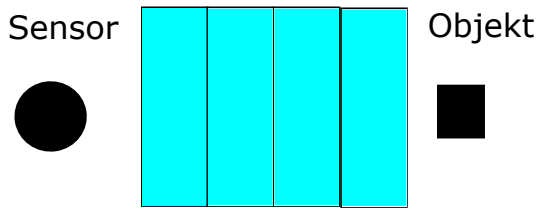


Abbildung 3.2 – Skizze über den Versuchsaufbau von Hasirlioglu [26] zur Untersuchung des Einflusses von Regen auf Radar- und LiDAR-Sensoren

Das Zeil war es die RCS-Werte eines Objektes (Viereck) von einem LiDAR-Sensor (Kreis) tu bestimmen. Zwischen beiden Komponenten befanden sich einzelne Schichten (blau) über welche Regen simuliert werden kann. Hierbei wurde feiner Sprühregen innerhalb einer Schicht durch ein Sprinklersystem erzeugt. Durch das Aktivieren mehrerer Schichten

zeitgleich kann die Degradierung der RCS-Werte im Zusammenhang mit der Regendichte genauer untersucht werden. Innerhalb dieses Versuches wurde dabei festgestellt, dass sich RCS-Werte um bis zu 67% reduzieren können. Dies sorgt laut Hasirlioglu [26] für eine deutliche Reduzierung der Leistung des Radar-Sensors, denn so ist der Radar-Sensor weniger in der Lage Objekte zu erkennen, zu verfolgen und zu klassifizieren. Auf dieser Grundlage wird für andere Blockaden ebenfalls eine Degradierung in der Messung der RCS angenommen.
Bei der Betrachtung von LiDAR-Sensoren im selben Versuchsaufbau wurden ebenfalls entsprechende Einflüsse auf weitere Messdaten festgestellt. So konnte gezeigt werden, dass mit steigender Regenintensität mehr Detektionspunkte zwischen Sensor und dem tatsächlichen Objekt wahrgenommen werden, weil Regentropfen diese Strahlen reflektieren. Dieses Resultat ist in Abbildung 3.3 zu erkennen. Auf der X-Achse sind dabei die Versuche mit steigender Regenintensität aufgetragen. Mit steigender Regenintensität sind mehr Detektionspunkte in der nähe des Sensors wahrzunehmen. Zusätzlich können weniger Detektionspunkte auf dem Ziel-Objekt am oberen Ende der einzelnen Messabbildungen festgestellt werden.

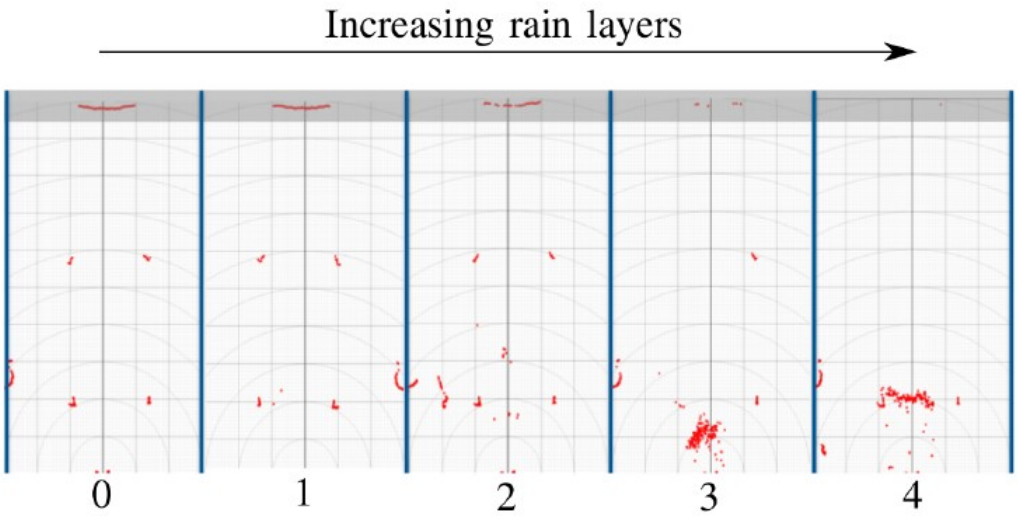


Abbildung 3.3 – Detektionspunkte eines LiDAR-Sensors in Abhängigkeit der Regenintensität ([26])

Ein weiterer Einfluss der Umwelt auf die Radar- und LiDAR-Sensoren sind Vibrationen. Wie in [27] als auch in [28] aufgezeigt, ist ein Automobilsensor verschiedenen Auslösen ausgesetzt beispielsweise:

- Bodenverhältnisse,
- Fahrverhalten des Fahrzeugführers (falls vorhanden),
- Vibrationen durch den Motorblock.

Aufgrund der ähnlichen Funktionsweise der Sensoren sind einige Auswirkungen nahezu identisch. Wie von Ma [27] und Schlager [28] aufgezeigt kann sich die Position von Mess-

punkten verschieben. Für die Untersuchung der Auswirkungen von Vibrationen wurden dort die entsprechenden Sensoren auf einer einstellbaren Vibrationsplatte befestigt. In einem definierten Abstand wurden vor dem Sensor einzelne oder mehrere Objekte aufgestellt, die mithilfe der Sensoren erkannt werden sollten. so zeigt Schlager [28] hier, dass sich die Anzahl der Detektionspunkte auf einem Objekt dadurch verändern kann und es wird draus entsprechend geschlussfolgert, dass die Anzahl der Punkte innerhalb einer Punktewolke sinkt. Die Verschiebung der Punkte innerhalb einer Messung belief sich dabei nur auf wenige Millimeter. Zusätzlich wird dort die These aufgestellt, dass die reduzierte Anzahl keine oder nur geringe Auswirkungen auf die Objekterkennung mit LiDAR-Sensoren haben sollte.
Bei Radar-Sensoren können allerdings zusätzliche Auswirkungen durch Vibrationen auftreten. So stellt Ma [27] die gemessenen Punkte auf einer Doppler-Range-Map dar. Eine solche Doppler-Range-Map ist in Abbildung 3.4 dargestellt. Dabei wird der Abstand des gemessenen Punktes über die X-Achse, die gemessene relative Geschwindigkeit des Punktes über die Y-Achse sowie die Intensität der Messung über die Z-Achse dargestellt. Zusätzlich wird die Intensität auch noch durch eine farbliche Veränderung der Fläche erweitert. Hier steht dunkelblau für keine Intensität und somit für keine gemessenen Punkte. Je rötlicher dabei der Bereich wird, desto mehr Punkte wurden an dieser Position, bzw. in diesem Abstand gemessen. Die nachfolgende Abbildung 3.4 zeigt dabei die Messergebnisse für einen nicht vibrierenden Sensor mit einem statischen Objekt. Wie bereits aufgezeigt spiegelt die blaue Fläche den leeren Raum des Experimentes wieder. Der einzelne Ausschlag in der Intensität lässt sich damit auf das Objekt zurückzuführen welches mit dem Radar gemessen werden sollte. Auch ist in der Abbildung zu erkennen, dass der Ausschlag eine zweidimensionale Fläche exakt am Nullpunkt der Y-Achse ist. Da in dem Versuch ein statisches Objekt gemessen worden ist und auch der Sensor sich nicht in einer fahrenden Bewegung befunden hat, ist die relative Geschwindigkeit des gemessenen Objektes null.

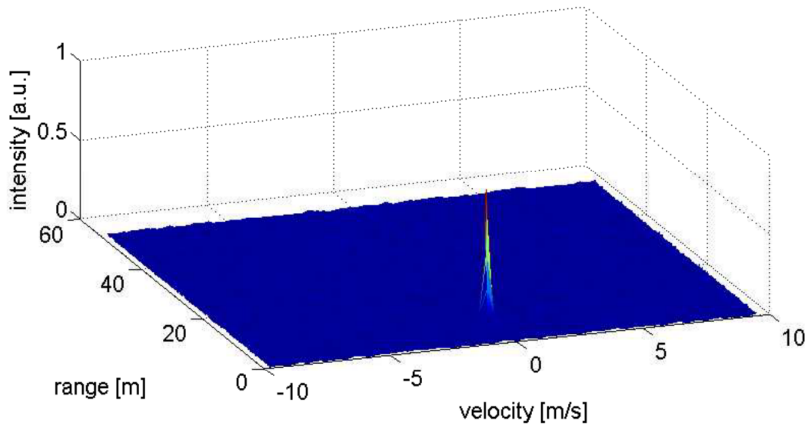


Abbildung 3.4 – Range-DopplerMap für ein nicht vibrierenden Radar-Sensor und einem statischen Ziel ([27]).

Vergleichend zu Abbildung 3.4 wird in Abbildung 3.5 der Versuch mit einem vibrierenden Sensoren dargestellt.

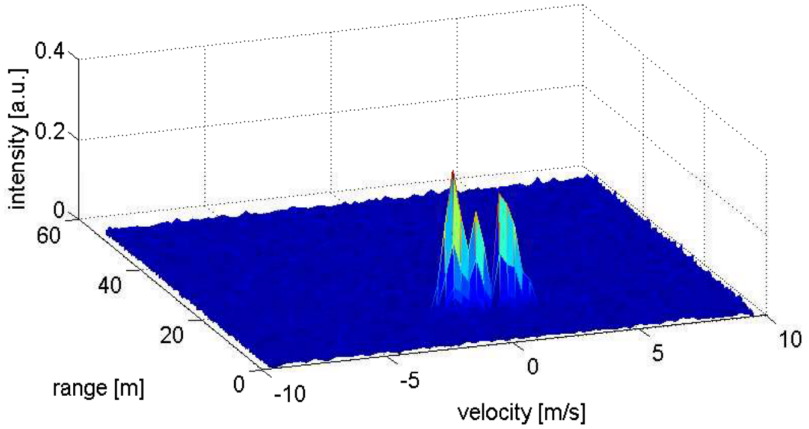


Abbildung 3.5 – Range-DopplerMap für einen vibrierenden Radar-Sensor und einem statischen Ziel ([27]).

Dort kann festgestellt werden, dass die zweidimensionale Fläche für die Geschwindigkeit in ein dreidimensionales Konstrukt übergeht. Zusätzlich wird die Fläche entlang der X-Achse größer. Auch ist erkennbar, dass sich drei Intensitätsspitzen innerhalb der Messung abbilden. In [27] werden diese Ergebnisse entsprechend interpretiert und es erfolgt die Aussage, dass aufgrund der geringen Frequenz der Vibration zwar eine Verschiebung in der Distanzerkennung vorhanden ist, diese aber nur minimal ist. Somit kann davon ausgegangen werden, dass in diesem Experiment die Distanzerkennung weiterhin funktionieren würde. Für entsprechend andere Vibrationen muss dies nicht der Fall sein. Ein größeres Problem wird dabei jedoch durch die drei Intensitätsspitzen beschrieben. Durch die Verschiebung der Intensitätsspitzen wird die Messung der relativen Geschwindigkeit, dargestellt auf der Y-Achse, verzerrt. Dies bedeutet für diesen expliziten Fall, dass ein eigentlich statisches Objekt als ein fahrendes Objekt erkannt wird. Dabei ist auch zu erkennen, dass es keine feste Richtung innerhalb der Verschiebung der Geschwindigkeit des Objektes gibt. So ist es möglich, dass das Objekt langsamer aber auch schneller detektiert wird, als es in der Realität der Fall wäre. Dieses Experiment zeigt dabei deutlich auf, dass die Geschwindigkeitsmessung eines Radar-Sensors durch Vibrationen stark beeinflusst werden kann. Diese Aussage wird durch einen identischen Versuchsaufbau und ähnliche Experimente von Hau [29] untermauert.

3.3 Cybersecurity eines Sensors

Wie bereits in Kapitel 2.2.3 vorgestellt unterteilen sich die betrachteten Angriffsarten in Jamming und Spoofing. Da LiDAR- und Radar-Sensoren auf ähnlichen physikalischen Grundlagen basieren, unterscheiden sich die Angriffsmöglichkeiten nur marginal. Für beide Angriffswege ist es grundlegend wichtig zu wissen in welcher physikalischen Wellenlänge der Ziel-Sensor arbeitet. Dafür werden die gesendeten Signale des Ziel-Sensors, oder auch *Victim Sensor* genannt, abgefangen. Über diese Signale kann nun die vom Victim Sensor verwendete Wellenlänge ermittelt werden.

Jamming Beim Jamming werden nun Wellen mit derselben Wellenlänge ausgesendet. Das Ziel dabei ist es das Noise-Level des Sensors zu erhöhen. Im Grunde genommen werden also wie bei herkömmlichen Jamming-Angriffen (bspw. bei Kommunikationsfunk) Störsignale erzeugt, mit dem Ziel tatsächliche Signale zu überlagern. Im Sensor führt dies zu Ungenauigkeiten der empfangenen Daten, bis hin zum Verlust von einzelnen Punkten (Detektionen), welche im Normalfall empfangen worden wären, jedoch durch zu viele Störsignale bis zur Unkenntlichkeit verändert worden sind [30], [31].
Zusätzlich zeigt Yeh [23], dass durch das Jammen von Radar-Sensoren die Gefahr von Kollisionen im Straßenverkehr besteht und die aktuelle Technik damit große Probleme hat. Allerdings wird dort auch beschrieben, dass es mit aktueller Jammertechnologie noch hinreichend schwierig ist erfolgreiche Angriffe auf auf ein gezieltes Fahrzeug durchzuführen. Dies kann sich allerdings mit technologischem Fortschritt in der Zukunft verändern. Entsprechende Auswirkungen werden für LiDAR-Sensoren unter dem Begriff *Saturation*(Übersättigung) in [32] beschrieben. Durch das Übersättigen des Empfängers können einzelne Punkte oder im schlimmsten Fall ganze Teile des FOVs unbrauchbar werden. Somit besteht für Radar- und LiDAR-Sensoren eine reale Gefahr durch Jamming. Zusätzlich muss hier aber noch betrachtet werden, dass eine solche Gefahr nicht nur durch gezielte Angriffe besteht.
Yeh [23] als auch Alland [30] beschreiben ein viel diskutiertes Problem, bei welchem die bereits beschriebenen Jamming-Auswirkungen auch innerhalb des normalen Straßenverkehrs auftreten können. Im regulären Straßenverkehr besteht die Möglichkeit, dass Sensor-Signale von Sensoren anderer Verkehrsteilnehmer von eigenen Sensoren aufgenommen werden. Somit können Interferenzen auch unbeabsichtigt zu den selben Auswirkungen wie ein gezielter Jamming-Angriff führen. Da die Auswirkungen in diesem Falle identisch sind zu denen eines gezielten Angriffes wird dieser Fehler der Kategorie der Cybersecurity zugeordnet obwohl, der Fehler durch die Umgebung generiert wird und somit auch der Kategoriebeschreibung von 3.2 entspricht.

Spoofing Spoofing-Angriffe auf Radar- und LiDAR-Sensoren laufen dabei ähnlich zu den Jamming-Angriffen ab. Allerdings unterschieden sich diese in ihren Auswirkungen massiv. Während es bei Jamming-Angriffen vor allem darum geht die Sensoren un-

brauchbar zu machen, zielen Spoofing-Angriffe darauf ab dem Sensor gezielt Fehlinformationen zu übermitteln. Tanis [31] beschreibt dabei eine Möglichkeit eines Spoofing-Angriffes als *Deceptive Jamming*. Aufgrund der in Kapitel 2.2.3 genutzten Definitionen wird diese Angriffsart hier den Spoofing-Angriffen zugeordnet. Dort wird aufgezeigt, dass durch einen zeitlichen Versatz der zurückgesendeten Radar-Welle Messpunkte im Victim Radar erzeugt werden können, die in jenem als realistische Messpunkte gewertet werden.
In [32] wird gezeigt, dass solche Angriffe auch bei LiDAR-Sensoren im Rahmen des Möglichen liegen. Auch wird dort veranschaulicht, dass es technisch möglich ist Detektionspunkte zu generieren, die näher am Victim-Sensor sind als der Angriffs-Sensor selbst. Spoofing-Angriffe nutzen dabei aus, dass generierte fehlerhafte Daten vom Sensor nicht als solche erkannt werden können. Der Sensor selbst erkennt die Daten als korrekt an und leitet somit falsche Informationen an die folgenden Systeme weiter. Eine große Besonderheit der aktuellen LiDAR-Sensoren entsteht durch die Verwendung der aktuellen 360°-LiDAR-Systeme. Aufgrund des gebogenen Glases des Sensors ist es möglich, dass Angriffe nicht nur innerhalb des FOVs durchgeführt werden müssen. In den bisher betrachteten Szenarien wurde immer davon ausgegangen, dass es eine direkte visuelle Verbindung zwischen Angreifer und Victim-LiDAR gibt.
Da 360°-LiDAR-Systeme allerdings eine gebogene Glaswandungen aufweisen, besteht die Möglichkeit, dass gesendete Lichtwellen durch das Glas gebrochen und in den Sensor weitergeleitet werden. In [32] wird gezeigt, dass es möglich ist fehlerhafte Punkte zu detektieren, wenn eine angreifende Lichtquelle nicht in direktem visuellen Kontakt zum Opfer steht. Abbildung 3.6 zeigt dabei die systematische Skizze darüber, wie sich ein

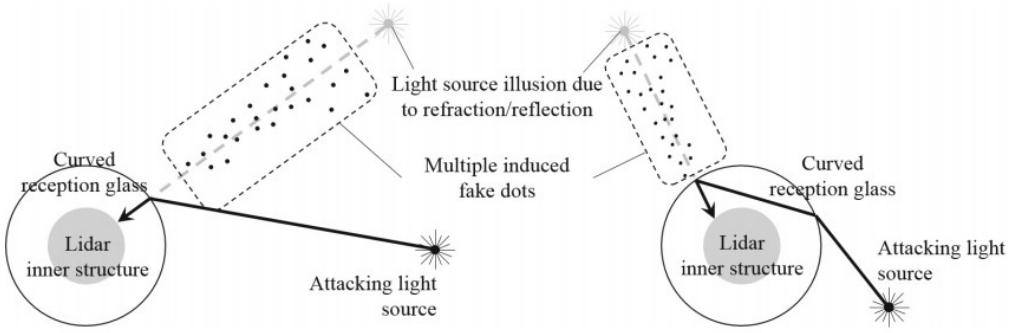


Abbildung 3.6 – Systematische Skizze eines Angriffes über eine gebogene Glaswandung auf ein LiDAR ([32])

theoretischer Lichtstrahl verhalten könnte, welcher von einem Angreifer von außerhalb des FOV ausgesendet wird. Der genaue Hergang solcher Angriffe wird im weiteren Verlauf der Arbeit nicht abgedeckt, jedoch betont es die Vulnerabilität von LiDAR-Sensoren und die damit verbundene Relevanz der Betrachtung von Security Angriffen auf diese.
Wie bereits gezeigt, ist es möglich fehlerhafte Detektionspunkte in den entsprechenden Sensoren hervorzurufen. Wenn viele solcher Punkte in einem System vorhanden sind,

können sogenannte *Ghost Targets* entstehen. Während es beim Spoofing ein definiertes Ziel sein kann solche fehlerhaften Umgebungsobjekte zu generieren, ist es theoretisch möglich, dass Ghost-Targets durch Jamming oder Interferenzen hervorgerufen werden. Die Autoren von [30] beschreiben die Auftrittswahrscheinlichkeit allerdings als sehr gering. Wahrscheinlicher ist es hingegen, dass einzelne Punkte beim Jamming erkannt werden, diese aber nicht als entsprechendes Objekt weitergeleitet werden.

4 Modelldefinition

Im vorherigen Kapitel wurde gezeigt wie Radar und LiDAR Sensoren auf unterschiedlichste Weise Fehler in den produzierten Daten aufweisen können. Hierbei konnte gezeigt werden, dass trotz völlig unterschiedlicher Ursachen, ähnliche oder gleiche Effekte auftreten können. Im Nachfolgenden werden diese Effekte daher in Fehlermodelle zusammengefasst. Diese Modelle liefern die Grundlage für die spätere Implementierung im CARLA-Simulator. Dabei soll auch sichergestellt werden, dass mehrere Modelle kompatibel sind und so auch eine Verknüpfung von Fehlern in der Implementierung ermöglicht wird. Dafür werden die einzelnen Modelle anhand der während der Entwicklung ermittelten Parameter vorgestellt.

4.1 Datenübertragungsfehler

Wie bereits zuvor vorgestellt kann es innerhalb eines autonomen Fahrzeuges zu Datenübertragungsfehlern kommen. Dort wird bereits zwischen einem Wackelkontakt und einem zeitlich verzögerten Datenübertragungsfehler unterschieden. Obwohl es sich hier bei beiden Fehlern um einen Datenübertragungsfehler handelt werden diese in den nachfolgenden Unterkapiteln getrennt betrachtet da ihre Auswirkungen sich gravierend unterscheiden.
Dabei werden Modelle unterteilt in paketverlustbehaftete Modelle und zeitverzögerte Übertragungsmodelle. Das paketverlustbehaftete Modell hingegen spiegelt sämtliche Fehler wieder, die für einen kompletten Informationsverlust stehen. Das zeitverzögerte Modell spiegelt dabei die Fehler wieder bei denen keine Daten verloren werden, diese aber nicht zum Detektionszeitpunkt an weitere Systeme weitergeleitet werden. Beide nachfolgend vorgestellten Modelle gelten für Radar- und LiDAR-Sensoren.

4.1.1 Paketverlustbehaftete Übertragungsfehler

Für paketverlustbehaftete Datenübertragungsfehler wurden die nachfolgend aufgelisteten Parameter ausgewählt:

- Startzeitpunkt,
- Auftrittsdauer,
- Intervall,
- Degradierung.

Diese Parameter bieten die Möglichkeit die Paketverluste genauer zu beschreiben. Dabei spiegeln der Startzeitpunkt und die Auftrittsdauer wieder, ab welchem Zeitpunkt ein Fehler in der Simulation auftreten kann und wie lange dieser anhält. Damit mehr als nur

ein einzelnes Fehlerauftreten simuliert werden kann wird das Modell mit dem Intervallparameter ergänzt. Dieser beschreibt in welchen Intervallen, beispielsweise alle 10 Sekunden, der Fehler wieder auftritt. In der Umsetzung muss allerdings bedacht werden, dass dennoch weiterhin auch die Simulation nur eines einzelnen Fehlerauftretens möglich sein muss.
Da bereits in den Grundlagen aufgezeigt wurde, dass sich Fehler im Laufe der Zeit verschlimmern können wird das Modell ebenfalls mit einem Degradierungsparameter ausgestattet. Dieser soll beschreiben wie sich das Modell im Laufe der Simulation verschlechtert. Somit ist es denkbar, dass der Fehler z.B. in schnelleren Intervallen wieder auftritt. Wie diese Veränderungen exakt aussehen und was sie beeinflussen ist an dieser Stelle noch nicht festgelegt. Für die Umsetzung des Degradierungsparameters ist es nötig zu wissen wie der Sensor innerhalb des Simulators funktioniert und wie der Fehler in die Simulation eingebunden wird. Eine Spezifizierung des Degradierungsparameters erfolgt daher erst in Kapitel 5.
Mittels diesem Fehlermodells können diverse reale Szenarien abgebildet werden, wie beispielsweise:

- Verlust weniger Pakete in hohen Intervallen.
  Dieses Szenario soll dabei einen kurzen aber sehr häufig auftretenden Wackelkontakt am Sensor simulieren.
- Verlust weniger Pakete in langsamen Intervallen zu Beginn und steigenden Intervallen während des Voranschreitens der Simulation.
  Dadurch soll sein ein Wackelkontakt simuliert werden, der im Laufe der sich im Laufe der Zeit verschlechtert und so zu einem häufigeren Auftreten führt.
- Totalausfall des Sensors.
  So kann ein kompletter Abbruch der Verbindung zwischen Sensor und folgenden Systemen simuliert werden.

4.1.2 Verzögerte Übertragung

Damit Datenübertragungsfehler aufgrund von Verzögerungen im System beschrieben werden können wird ein Modell mit den nachfolgenden Parametern definiert:

- Startzeitpunkt,
- Grad des Verlustes,
- Intervall,
- Degradierung.

Dieses Modell wird fast vollständig durch dieselben Parameter beschrieben wie das zuvor vorgestellte Modell (siehe Abschnitt 4.1.1). Beide Modelle benötigen Informationen

darüber, wann und in welchen Intervallen Fehler auftreten. Da auch dieser Fehler sich im Laufe der Zeit sich verstärken kann wird ein Degragierungsparameter benötigt. Verschiedene Ansätze sind denkbar, wie eine Umsetzung über eine zeitliche Verschiebung oder eine Verschiebung über die Anzahl der zu verzögerten Packete denkbar. Die genaue Definition erfolgt erst in Kapitel 5.4.
Mit diesem Modell sollen folgende Szenarien simulierbar sein:

- Verzögerung weniger Pakete in schnellem Intervall.
  Dies soll dabei eine häufig auftretende Verzögerung von Paketen durch z.B. eine konstant hohe Last im internen Netzwerk mit kurzen Überlastungen darstellen.
- Verzögerung vieler Pakete in steigendem Intervall.
  Dieses Szenario spiegelt im Gegensatz eine steigende, langanhaltende Last im Netzwerk wieder, die im Laufe der Zeit häufiger auftritt.

4.2 Datenveränderungsfehler

Fehler die eine Veränderung der Daten hervorrufen werden im nachfolgenden anhand ihrer Effekte getrennt betrachtet. Da es Unterschiede zwischen den Effekten bei den Sensoren gibt kann nicht jedes Modell auf den Radar- sowie den LiDAR-Sensor angewendet werden. Da zum Beispiel: LiDAR-Sensoren keine Geschwindigkeitsmessungen vornehmen kann hier auch keine Verschiebung der Messung erfolgen.

4.2.1 Verschiebung von Positionsmessungen

Für die Beschreibung des Modells wurden die nachfolgenden Parameter ausgewählt:

- Auftrittszeitpunkt,
- Intervall,
- Dauer,
- Randomisierung,
- Art der Verschiebung,
- Degradierung.

Wie bereits bei den vorgestellten Modellen ist es nötig grundlegende Informationen wie den Auftrittszeitpunkt, das Intervall sowie einen Parameter für die Degradierung in das Modell einzubinden. Wie bereits in Kapitel 4.1.1 aufgezeigt muss auch die Fehlerdauer berücksichtigt werden, weshalb das Modell einen Parameter für diese besitzt. Die bisher vorgestellten Parameter stecken allerdings nur den Rahmen für die Simulation ab. Es ist daher noch nötig das Modell um einen Verschiebungsparameter zu erweitern.

Dieser beschreibt inwiefern die detektierten Punkte verschoben werden sollen. Genauer gesagt, was die maximale Verschiebung eines Punktes entlang einer gewählten Achse ist. Da die Verschiebung allerdings zufällig sein soll, muss hier eine Randomisierung der Verschiebung in jede definierte Richtung gewählt werden. Damit eine solche Randomisierung erfolgen kann, wird das Modell mit einem entsprechenden Parameter erweitert. Dieser beschreibt welche Verteilungsart genutzt werden soll, um eine zufällige Verschiebung auszuwählen. Dabei soll es Möglich sein zwischen einer linearen Verteilung und einer Weibullverteilung auszuwählen. Durch diese umfangreiche Parametrisierung des Modells sollen unter anderem folgende Szenarien simulierbar sein:

- Kleine Verschiebung der detektierten Punkte auf der Oberfläche.
- Verschiebung der erkannten Distanz eines Objektes.
- Konstant häufiger auftretende Verschiebungen in alle Richtungen.

Alle drei theoretischen Szenarien spiegeln dabei verschiedene Auswirkungen aufgrund von Vibrationen auf einen Radar oder LiDAR-Sensor wieder.

4.2.2 Fehlerhafte Geschwindigkeitsmessungen

Das Modell der fehlerhaften Geschwindigkeitsmessung wird durch dieselben Parameter beschrieben wie das zuvor vorgestellte Modell:

- Auftrittszeitpunkt,
- Intervall,
- Dauer,
- Randomisierung,
- Art der Verschiebung,
- Degradierung.

Der einzige Unterschied zu dem zuvor vorgestellten Modell ist eine Veränderung innerhalb der Beschreibung für die Art der Verschiebung. Für dieses Modell kann nur eine Verschiebung der Geschwindigkeit erfolgen. Somit muss der Parameter hier nur eine Verschiebung und keine drei einzelnen abbilden. Dieses Fehlermodell ist lediglich dazu gedacht andere Fehlermodelle, wie z.B. die Verschiebung der Messpunkte, zu erweitern. Bei Radar-Sensoren tritt allerdings eine Verschiebung der Positionsmessung in den meisten Fällen auch mit der Verschiebung der Geschwindigkeitsmessung auf. Damit hier eine realitätsnahe Simulation für beide Sensoren erfolgen kann, ist es nötig die beiden Fehlermodelle für den Radar-Sensor zu kombinieren.

4.2.3 Reduktion effektiver Reichweite

Für die Beschreibung des Modells für die Reduktion der effektiven Reichweite eines Sensors werden die nachfolgenden Parameter benötigt.

- Auftrittszeitpunkt,
- Intervall,
- Dauer,
- Degradierung,
- Reduktion.

Wie in den vorherigen Modellen ist es wichtig zu beschreiben wie der Fehler auftreten kann, in welchen Wiederholungsintervallen er auftritt und wie lange der Fehler andauert. Zusätzlich erhält auch dieses Modell einen Degradierungsparameter wodurch eine Verschlechterung des Fehlers im Laufe der Zeit simuliert werden kann. Für die Beschreibung der Auswirkung des Fehlers erhält das Modell einen Reduktionsparameter. Dieser beschreibt wie sich die effektive Reichweite des Fehlers verändert. Diese Parameterkombination ermöglicht Simulationen wie z.B.:

- Häufige kurze Intervalle mit hoher reduzierter Messreichweite.
- Seltene hohe Intervalle mit niedriger reduzierter Messreichweite.
- Konstant häufiger auftretende Reduktion der Reichweite.

Die vorgestellten, theoretisch möglichen Simulationen spiegeln dabei einen Angriff auf den Sensor wieder. Durch die Erhöhung des Noise-Levels sinkt die effektive Reichweite des Sensors. Dabei hat der Angreifer Einfluss auf die Auswirkung und kann je nach Anpassung seines Angriffs die möglichen Szenarien abbilden. Aber auch Vibrationen können einen Effekt wie im letzten beschrieben Szenario erzeugen.

4.2.4 Detektion nicht existenter Messpunkte

Das Modell für die Detektion nicht existenter Messpunkte kann durch die nachfolgenden Parameter beschreiben werden:

- Auftrittszeitpunkt,
- Intervall,
- Dauer,
- Degradierung,
- Detektionsmenge,

- Detektionsort.

Auch dieses Modell besitzt die bereits vorgestellten grundlegenden Fehlerparameter. Neben diesen wird das Modell wird das Modell um Parameter über die Detektionsmenge und den Detektionsort erweitert. Dabei beschreibt der Detektionsort-Parameter wo innerhalb des FOV des Sensors nicht existierende Punkte detektiert werden. In welcher Form diese Bereiche angegeben werden können wird im Hinblick der technischen Umsetzung beschrieben. Neben dem Parameter für die örtliche Detektion der Messpunkte ist es nötig zu wissen, wie viele Punkte fehlerhaft detektiert werden. Diese Information wird über den Parameter der Detektionsmenge eingestellt.
Durch die Verknüpfung der genannten Parameter werden somit Szenarien wie:

- Häufige Intervalle mir kurzer Dauer und mit wenigen fehlerhaften Messpunkten im gesamten FOV.
- Seltene Hohe Intervalle mit vielen fehlerhaften Messpunkten in einem definierten Teil des FOV.
- Konstant steigendes Auftreten von fehlerhaften Messdetektionen in teilen des FOV.

denkbar.
Wie bereits im vorherigen Kapitel stellen diese Szenarien mögliche Effekte von Angriffen auf LiDAR oder Radar Sensoren dar. Die Ausprägung des Fehlers wird dadurch durch den Angreifer festgelegt und kann unter anderem zu den beschrieben Szenarien führen.

4.3 Verschiebung und Rotation von Sensoren

Um die Veränderung der Orientierung des Sensors beschreiben zu können erhält das Modell folgende grundlegende Parameter:

- Ereignis,
- Verschiebungs- und Rotationsbeschreibung,
- Veränderungsart

sowie die optionalen Parameter

- Auftrittszeitpunkt,
- Intervall,
- Dauer,
- Degradierung.

Im Vergleich zu den bereits vorgestellten Modellen gibt es hier einen sogenannten Ereignisparameter. Dieser beschreibt ob eine Verschiebung auf zeitlicher Basis erfolgt oder ob eine Verschiebung durch ein externes Ereignis ausgelöst werden soll. Ein denkbares Ereignis für eine Verschiebung wäre dabei die Kollision mit einem anderen Objekt in der Simulation. Für zeitbasierte Veränderungen der Orientierung des Sensors erhält das Modell die bereits in den anderen Modellen vorgestellten Parameter bezüglich des Auftretens, der Wiederholung des Fehlers (Intervall), dem Parameter zur Beschreibung der Dauer des Fehlers sowie den dazugehören Degradierungsparameter.
Für andere Ereignisse, welche nicht über einen zeitlichen Ablauf ausgelöst werden, wie zum Beispiel: kollisionsbedingte Verschiebungen, werden die genannten Parameter ignoriert und sind daher als optional definiert. Zusätzlich gibt es den Veränderungsparameter. Dieser beschreibt die Verschiebung und Rotation des Sensors beim Auftreten des Events und wird zusätzlich mit einem Parameter gekoppelt welcher die Art der Veränderung beschreibt. Mithilfe dessen wird beschrieben, ob eine Veränderung in der Verschiebung schlagartig erfolgt oder ob sie über den zeitlichen Ablauf des Fehlers regelmäßig aufritt. Dadurch ermöglichen sich folgende Szenarien:

- Starke schlagartige Veränderung des Sensors FOV
  Dieses Szenario kann so genutzt werden um eine Kollision des Fahrzeuges zu simulieren.
- Konstante Rotation des Sensors während eines zeitlichen Intervalls.
  Dabei sollen die Auswirkungen von unbefestigten Straßen auf einen nicht korrekt befestigten Sensor simuliert werden.

4.4 Blockade des Field of View

Damit die Blockade eines Sensors beschrieben werden kann wird das Modell mit folgenden Parametern ausgestattet:

- Auftrittszeitpunkt,
- Intervall,
- Degradierung,
- Anzahl,
- Ort,
- Lebenszeit,
- Fallgeschwindigkeit.

Auch dieses Modell besitzt wie die vorherigen einige Basisparameter, wie den Startparameter. Das Intervall beschreibt hierbei nun nicht ein Auftreten des Fehlers sondern das Auftauchen einer weiteren Verschmutzung innerhalb des FOV des Sensors. Dies ist mit dem Degradierungsparameter gekoppelt und ermöglicht das Abbilden von stetig steigenden Blockaden. Beispielsweise könnte das Fahren auf einer schlammigen Straße bei starkem Regen abbilden, wobei sich die Qualität der Straße durch den Regen konstant verschlechtert und sich somit häufiger Schlamm vor dem Sensor ansammelt so abgebildet werden. Zusätzlich benötigt das Modell einen Anzahlparameter, der beschreibt wie viele Blockadeobjekte beim Auftreten des Fehlers innerhalb des FOV generiert werden sollen. Wo diese Blockade entsteht wird durch den Ortsparameter beschrieben, der angibt in welchem Bereich des FOV eine Blockade entstehen soll und ob eine Blockade nahe des Sensors entsteht oder ob sich diese über die gesamte Reichweite des Sensors erstreckt. Damit weitere Blockadearten beschrieben werden können werden die Blockadeobjekte mit einem sogenannten Lebenszeitparameter ausgestattet. Dieser beschreibt wie lange ein einzelnes Objekt in der Simulation den Sensor blockieren kann. Nach Ablauf der Zeit findet durch das Objekt keine Blockade des Sensors mehr statt.
Damit dynamischere Blockaden simuliert werden können gibt es noch einen Fallgeschwindigkeitsparameter. Dieser beschreibt wie schnell sich Objekte in Richtung Boden bewegen und sich somit den Bereich in dem sie eine Blockade hervorrufen verändern. Durch die Kombination dieser Parameter werden Szenarien wie zum Beispiel:

- Konstant steigende Blockade direkt vor dem Sensor durch z.B. Schlamm und Fahren auf schlammigen Straßen.
- Konstante Blockade durch sich bewegende Blockadeteilchen im ganzen FOV.
  Dies kann unter anderem eine Blockade durch tauenden Schnee beschreiben der sich langsam entlang des FOVs bewegt.
- Fallende, blockierende Objekte im gesamten FOV zur Simulation von Regenproblemen bei LiDAR-Sensoren.

5 Implementierung der Modelle

Nachfolgend wird vorgestellt wie die Modelle in der Simulation umgesetzt wurden. Da viele Modelle überschneidende Parameter besitzen, werden diese einmalig zu Beginn erläutert und einzelne Modelle die von diesen Parameter abweichen werden mit den entsprechenden Informationen erweitert. Damit auch eine volle Funktionsweise der regulären Sensorik sichergestellt werden kann, werden neue Sensoren mit dem Hintergrund der Vererbung in den Server integriert. Somit können sämtliche Einstellungsoptionen der regulären Sensoren einfach und schnell auf einen entsprechend Fehlerhaften Sensor erweitert werden.

5.1 Generelle Informationen

Das Ziel der Implementierung ist es die bereits vorgestellten Modelle in den CARLA-Simulator einzubinden. Neben den einzelnen Modellen und deren Implementierung ist es nötig einige grundlegenden Abläufe und generelle Informationen zum Simulator und dessen Funktionsweise zusammenzutragen. Der Ablauf einer Simulation innerhalb der CARLA-Pipeline (Abbildung 2.3) welcher für die Umsetzung genutzt wird, kann hier wie nachfolgen beschrieben werden.

1. Start des Simulators.
2. Start eines Clients.
3. Client sendet Informationen an den Server.
   Dabei wird dem Simulators die Information darüber mitgeteilt welche anderen Verkehrsteilnehmer teil des simulierten Szenarios sind..
4. Client sendet Informationen über das simulierte Fahrzeug.
   Hier sendet der Client die Information über das ausgewählte Fahrzeugmodell, die zu simulierenden Sensoren und die zu verwendenden Fehlermodelle.
5. Simulationsschleife startet.
   Der Simulator beginnt das simulierte Fahrzeug zu bewegen, Sensordaten zu generieren und simuliert die einzelnen Open Source nacheinander.
6. Beenden des Clients.
   Dabei werden alle Verkehrsteilnehmer einschließlich des simulierten Fahrzeuges entfernt und der Simulator kann somit ohne Neustart für eine neue Simulation genutzt werden.

Die Umsetzung einer Simulation soll so ablaufen, dass bereits vorhandene Simulationsstrukturen erweitert werden können. Dies bedeutet, die Implementierung erfolgt immer auf

dem Weg über die Pyhton-Scripts der möglichen Clients. Diese Scripts beinhalten sämtliche Informationen zu den zu simulierenden Szenarien, eingeschlossen aller Parameter für die Fehlersimulation. Diese Informationen werden nun während der Simulationsschleife abgeprüft und führen bei Bedarf zu Änderungen in der Simulation. Damit der Simulator darüber informiert ist welche Fehlermodelle in der aktuellen Simulation für einen spezifischen Sensor genutzt werden, wurde der Sensor mit einem Szenarioparameter ausgestattet.
Diese „ScenarioID“ speichert in einzelnen Bits welche verschiedenen Szenarien gelten und es wird für jeden simulierten Frame überprüft ob das Szenario aktiv ist. Sollte hierbei ein Szenario als aktiv geschalten sein, wird für jeden Frame die Logik dieses Fehlermodelles durchlaufen. Neben Parametern die in fast allen Modellen vorhanden sind gibt es implementierungsspezifische Parameter, welche die zuvor vorgestellten Fehlermodelle genauer spezifizieren und für die Logik der Fehlermodelle benötigt werden. Zusätzlich soll sichergestellt werden, dass der implementierte fehlerhafte Sensor ohne aktives Fehlerszenario funktioniert, genau wie ein regulärer CARLA-Sensor.

5.2 Implementierung von geteilten Parametern

Zu den geteilten Parametern zählen die Folgenden:

- Startzeitpunkt,
- Auftrittsdauer,
- Intervall,
- Degradierung.

Für Parameter die einen zeitlichen Aspekt abbilden ist es hier wichtig zu wissen, dass der CARLA-Simulator den zeitlichen Ablauf der Simulation als eine Gleitkommazahl (float) wiedergibt und diese ab Start des Simulationsservers über die tatsächlich verstrichene Zeit angepasst wird. Eine für uns relevante Simulation startet nicht unbedingt ab Sekunde null. Dies liegt daran, dass der Simulationsserver bereits vor der eigentlichen Simulation gestartet wird und somit bis zu ihrem Beginn bereits Zeit vergeht. Daher wird der Startzeitpunkt, also ein float-Wert, innerhalb der Simulation gewählt anstatt der mit dem zeitlichen Versatz zwischen Serverstart und tatsächlichem Beginn der für uns relevanten Simulation erweitert. Die Auftrittsdauer wird auch als float-Wert beschrieben und wird für eine einfache Berechnung genutzt, die es ermöglicht, in den einzelnen Zeitschritten abzuprüfen, ob der Fehler während dieses zeitlichen Bereichs aktiv ist. Das Intervall beschreibt nun einen float-Wert der genutzt wird um den Startzeitpunkt während der Simulation anzupassen. Nachdem ein einzelnes Fehlerauftreten simuliert wurde, wird der Startzeitpunkt um den Wert des Intervalls erweitert und so der Startzeitpunkt für das nächste Auftreten eines Fehlers festgelegt.

Da bisher immer nur von dem Degradierungsparameter gesprochen wurde, ist es nötig diesen für die Implementierung genauer zu beschreiben. Hier wird der Degradierungsparameter in zwei Unterparameter aufgeteilt. Einen Dauerdegradierungs- sowie einen Intervalldegradierungsparameter. So wird der generelle Fehlerdauerparameter als auch der Fehlerintervallparameter nach jeder Fehlersimulation mit ihrem zugehörigen Degradierungsparameter verrechnet. So kann die Dauer des Fehlers sowie das Intervall nach jedem Auftritt steigen oder sinken. Zusätzlich gibt es aufgrund der Client-Server-Struktur des CARLA-Simulators die Möglichkeit die beiden Parameter zu jedem Zeitpunkt der Simulation anzupassen und so ein dynamisches Fehlverhalten zu erzeugen.

5.3 Paketverlustbehaftete Übertragungsfehler

Die Implementierung von paketverlustbehafteten Übertragungsfehlern nutzt lediglich die bereits vorgestellten grundlegenden Parameter. Über die Überprüfung der Grundparameter kann festgestellt werden, ob der Fehler in dem aktuell simulierten Frame aktiv geschaltet ist. In dem Fall, dass der Fehler aktiv ist wird das Aussenden der generierten Sensordaten unterbunden. In Abbildung 5.1 ist dieser Ablauf skizziert.

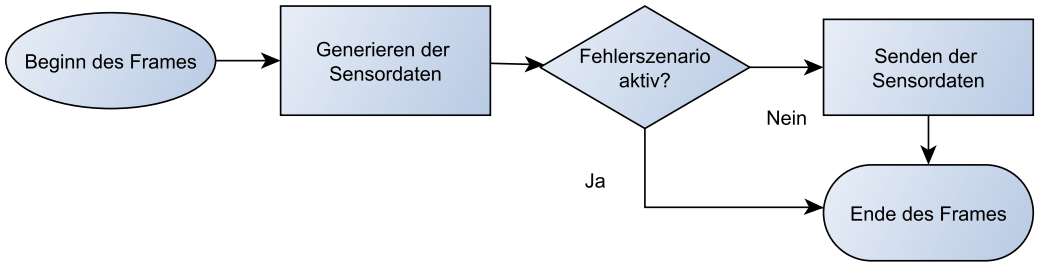


Abbildung 5.1 – Flussdiagramm des skizzierten Ablaufs des Paketverlustbehafteten Übertragungsfehlers.

5.4 Verzögerte Übertragung

Da die Sensordaten während eines einzelnen Open Source erstellt werden, müssen diese Daten für eine Verzögerung längerfristig gespeichert werden. Dafür würde ein sogenannter *Ringbuffer* genutzt.
Dies bedeutet, dass sämtliche generierten Daten in dem Ringbuffer gespeichert werden. Abhängig davon ob eine Verzögerung stattfindet werden nun Daten aus dem Ringbuffer weitergeleitet.
Für den Ringbuffer ist es nötig zwei verschiedene Offsets zu überwachen. Es muss überwacht werden wo neue generierte Pakete im Ringbuffer gespeichert werden müssen. Zusätzlich muss bekannt sein welche Pakete bereits gesendet wurden. Da es aufgrund dieser Art der Implementierung zwei verschiedene Offsets gibt, besteht die Möglichkeit,

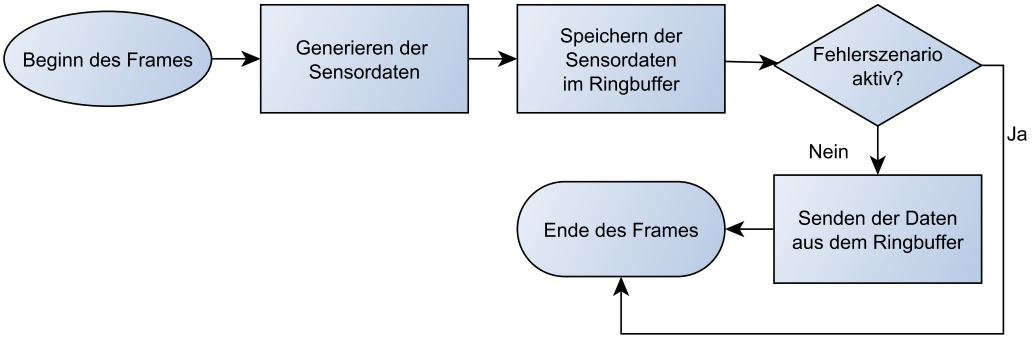


Abbildung 5.2 – Flussdiagramm des skizzierten Ablauf für eine Verzögerte Übertragung.

dass bei einer großen Verzögerung sich die Offsets wieder annähern und so eine tatsächliche Verzögerung verkleinert wird. Dies ist damit verbunden, dass verzögerte noch nicht gesendete Pakete durch andere verzögerte Pakete überschrieben werden. Somit besteht die Möglichkeit, dass einzelne Datenpakete auch verloren gehen können. Dieser Verlauf wird skizzenhaft in der Abbildung 5.2 verdeutlicht.
Damit dieser Ablauf möglich ist können die nachfolgenden Parameter gesetzt werden.

1. Anzahl der verlorenen Pakete.
   Dieser Parameter beschreibt wie viele Pakete beim Auftreten des Fehlers verzögert werden. Während eine Verzögerung somit aktiv ist, werden somit die Pakete zurückgehalten. Dies führt zu einer kurzen Phase ohne die Übertragung von Daten.

2. Genutzte Größe des Ringbuffers
   Wie bereits beschrieben, ist die Größe des Ringbuffers ausschlaggebend dafür, wie früh und wie häufig Pakete verloren gehen. Je kleiner der Ringbuffer, desto früher und häufiger treten Paketverluste auf.

5.5 Verschiebungen von Messungen

Nachfolgend werden die Parameter die für die Verschiebung von Messpunkten benötigt werden für Radar und LiDAR Sensoren getrennt definiert. Innerhalb des CARLA-Simulators werden Radardaten in ein horizontales Koordinatensystem übertragen.
Dies bedeutet, dass der Messpunkt über den Azimuth- und Altidute-Winkel sowie eine Tiefe beschrieben wird. Zusätzlich wird bei Radardaten eine eine Geschwindigkeit des detektierten Punktes übermittelt. Das LiDAR hingegen speichert seine Daten in einem regulären dreidimensionalen Koordinatensystem mit den entsprechenden X, Y und Z-Achsen.
Für die Implementierung ist es daher notwendig alle Parameter verändern zu können. Da eine Verschiebung des Messpunktes für LiDAR und Radar möglich ist wurde die Geschwindigkeitsmessung sowie die Verschiebung des Messpunktes getrennt implementiert. Dies bedeutet, dass beide Implementierungen unabhängig voneinander einen

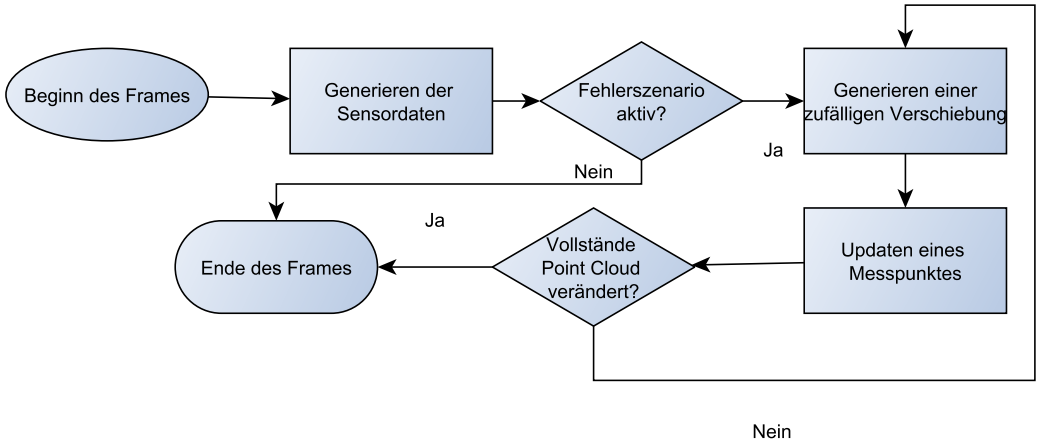


Abbildung 5.3 – Flussdiagramm für den skizzierten Ablauf von verschobenen Messwerten.

Start-, Intervall-, Dauer- und entsprechende Degradierungsparameter besitzen. Damit eine Verschiebung erfolgen kann muss diese genauer definiert werden. Hierfür wurde zunächst festgelegt, dass die Verteilung der Messpunkte gleichverteilt oder weibullartig erfolgen kann.
Für die Weibullverteilung wurden dabei die Parameter fest mit einem Skalenparameter von 1 und einem Formparameter von 3.602 für eine Annäherung an eine Normalverteilung hinterlegt. Zusätzlich wurde ein Mersenne-Twister-Zufallszahlengenerator mit einem festen Seed eingebunden. Somit wird sichergestellt, dass die Simulationen wiederholbar sind, da Zufallszahlengeneratoren feste Algorithmen besitzen und somit unter Verwendung des gleichen Seeds immer eine konstante Abfolge an zufälligen Zahlen innerhalb einer Simulation generieren. Die Kombination aus dem Zufallszahlengenerator und der durch den Nutzer festgelegten Verteilung wird verwendet, um zufällige Zahlen zwischen 0 und 1 zu generieren. Zusätzlich zu der Verteilung wird durch den Nutzer eine maximale Abweichung für die einzelnen festgelegten Werte angegeben. Dies bedeutet, der Nutzer gibt für das Radar eine maximale Abweichung für die folgenden Parameter an:

1. Azimuth,
2. Altitude,
3. Abstand,
4. Geschwindigkeit.

Für jeden der Werte wird nun eine zufällige Zahl zwischen null und eins ermittelt und diese beschreibt den Anteil der Verschiebung des Wertes. Diese Verschiebung wird nun mit dem tatsächlich gemessenen Wert verrechnet und so der neue verschobene Messwert generiert. Die Einbindung des Ablaufs innerhalb eines Open Source kann der Abbildung 5.3 entnommen werden.

Dieselbe Logik kann für das LiDAR verwendet werden. Allerdings findet dort eine Verrechnung mit den nachfolgenden Parametern statt:

1. Maximale Abweichung in der X-Achse,
2. Maximale Abweichung in der Y-Achse,
3. Maximale Abweichung in der Z-Achse.

Dieser Ablauf kann dabei der Abbildung 5.3 entnommen werden.
Wie dort zu erkennen ist, wird für jeden Detektionspunkt innerhalb des simulierten Open Source eine Verschiebung generiert und diese Verschiebung wird auf den einzelnen Punkt angewendet. Dies wird in einer Schleife wiederholt, bis sämtliche generierten Messungen verändert sind. So wird sichergestellt, dass eine Verschiebung der gesamten Punktwolke erfolgt.
Wichtig ist dabei, dass die detektierten Punkte innerhalb der Simulation immer über die X-, Y- und Z-Achse beschrieben werde. Für das LiDAR werden diese Werte direkt verändert. Für das Radar werden diese Werte mithilfe vorgefertigter Funktionen der Unreal Engine in Azimuth, Altitude und Abstand umgerechnet. Anschließend wird auf die neu berechneten Werte die Verschiebung angewandt.

5.6 Veränderte Reichweite

Die Implementierung von LiDAR- und Radar-Sensoren in CARLA bietet bereits die Möglichkeit die Reichweite eines Sensors zu bestimmen. Da die fehlerhaften LiDAR- und Radar-Sensoren aufgrund der Vererbung diesen Parameter ebenfalls besitzen kann hier eine Implementierung auf einfachem Wege erfolgen. Damit das Fehlermodell allerdings aktiv geschaltet werden kann muss die neue Reichweite an das Modell übergeben werden. Der Ablauf wird in Abbildung **??** für ein besseres Verständnis skizziert.
Für den Fehler wird überprüft ob ein Frame der erste Frame mit aktivem Fehler oder der erste Frame ohne aktiven Fehler ist. Sollte diese Prüfung erfolgreich sein findet eine Veränderung der genutzten Reichweite des Sensors statt. Im Falle des ersten Open Source mit reduzierter Reichweite, wird der reguläre Wert für die Reichweite gespeichert und mit dem reduzierten Wert überschrieben. Für den anderen Fall findet dies umgekehrt statt und die Reichweite des Sensors wird auf den regulären Wert zurück gesetzt. Beides erfolgt vor der Generierung der einzelnen Messpunkte. So wird sichergestellt, dass die Generierung der Messpunkte mit der für diesen Zeitpunkt korrekten Reichweite erfolgt.

5.7 Detektion nicht existenter Punkte

Damit die Detektion von nicht existenten Punkten erfolgen kann werden die bereits vorgestellten grundlegenden Parameter mit folgenden Parametern erweitert:

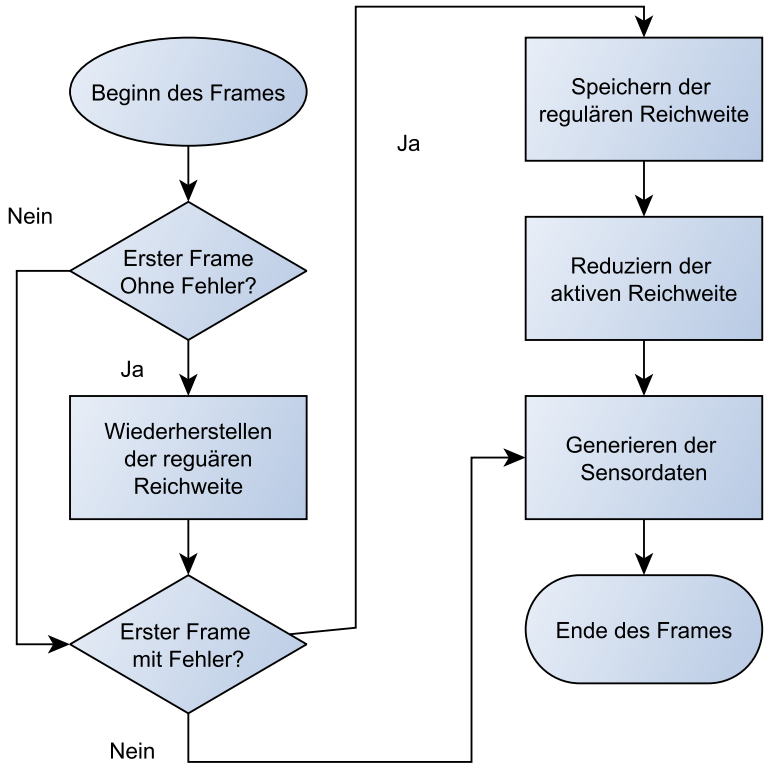


Abbildung 5.4 – Flussdiagramm des skizzierten Ablaufes einer reduzierten Sensorreichweite.

1. Anzahl der fehlerhaften Detektionen,
2. Flag-Variable für den vertikalen Bereich der Detektionen,
3. Flag-Variable für den horizontalen Bereich der Detektionen.

Wie bereits in den zuvor vorgestellten Fehlermodellen wird in jedem Frame geprüft ob das Fehlermodell aktiv ist.
Sollte dies der Fall sein, wird nach der Generierung der Sensordaten eine einfache Schleife durchlaufen, die basierend auf der Anzahl der zu generierenden fehlerhaften Detektionen zufällige Punkte innerhalb des FOV generiert.
Diese Generierung prüft dabei die beiden Flag-Variablen die vom Nutzer gesetzt werden können. Dabei können über diese folgende Einstellungen getroffen werden:

1. Vertikal:
   a) Gesamtes vertikales FOV,
   b) Oberer Teil des FOV,
   c) Unterer Teil des FOV.
2. Horizontal:

a) Gesamtes horizontales FOV,

b) Rechter Teil des FOV,

c) Linker Teil des FOV.

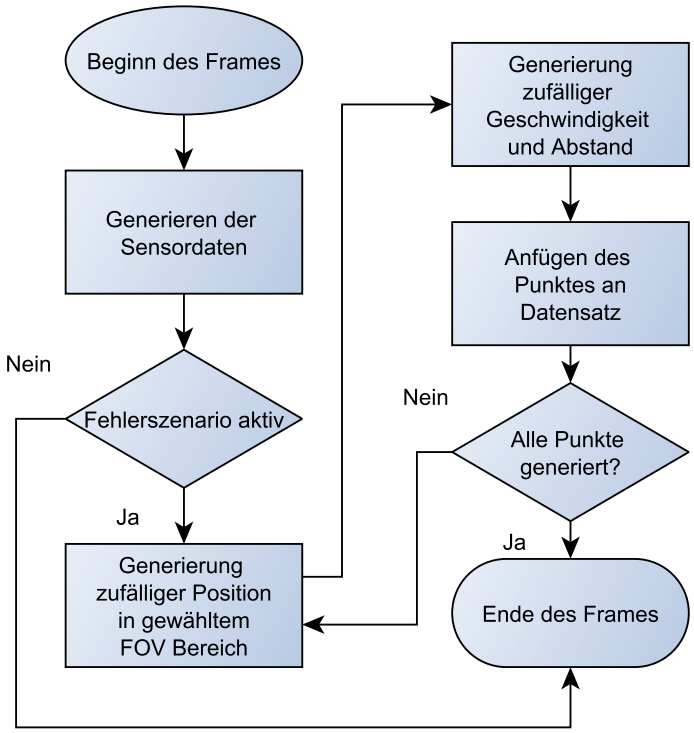


Abbildung 5.5 – Flussdiagramm des skizzierten Ablaufs der Detektion von nicht existierenden Punkten.

Diese Bereiche werden in der Generierung der Position der fehlerhaften Detektion berücksichtigt. Zusätzlich wird ein zufälliger Abstand des Punktes zum Sensor, sowie eine zufällige relative Geschwindigkeit des Punktes generiert.
Für die Generierung der Position des Messpunktes innerhalb des FOV wird Abhängig von der Sensorart ein anderer Algorithmus verwendet. Da bei den LiDAR-Sensoren ein 360° FOV vorhanden ist werden über eine Gleichverteilung zufällige Werte entlang der X,Y und Z-Achse generiert generiert die innerhalb der Messreichweite des Sensors liegen. Dafür wird die Messreichweite mit der zufälligen Zahl zwischen 0 und 1 multipliziert. Um den definierten Bereich des FOV sicher zu stellen können die einzelnen Werte innerhalb des FOV transferiert werden. Anschließend werden die generierten Werte mit der Position des Sensors verrechnet wodurch eine neue Position innerhalb des FOV entsteht.
Die Berechnung für den Radar-Sensor nutzt dabei einen in CARLA bereits vorhandenen Algorithmus. Für das Ray-Casting berechnet der CARLA-Simulator einen zufälligen Punkt mit dem Abstand der Messreichweite. Dieser Algorithmus wurde angepasst um eine zufällige Reichweite zwischen 0 und der Messreichweite des Sensors zu verwenden. Durch diese Anpassung generiert der Algorithmus einen zufälligen Punkt innerhalb

des FOV des Sensors. Wie beim LiDAR-Sensor kann an dieser Stelle der Punkt innerhalb den vier festgelegten Bereichen transformiert werden um sicher zu stellen, dass der Punkt im vorgegeben Bereich des FOV liegt. Dies erfolgt durch eine Überprüfung in welchem Bereich des FOV der generierte Punkt liegt. Hierbei kann ein Transfer des Punkte vom linken in den rechten Bereich sowie vom oberen in den unteren Bereich über einen Austausch des mathematischen Vorzeichens erfolgen.
Die dabei durchlaufene Struktur kann der Abbildung 5.5 entnommen werden.

5.8 Verschiebung des Sensors

Für die Verschiebung des Sensors werden die grundlegenden Parameter umfangreich erweitert. Die nachfolgenden Parameter werden für die Implementierung umgesetzt:

1. Rotation des Sensors:
   a) Yaw-Winkel,
   b) Pitch-Winkel,
   c) Roll-Winkel,
2. Flag-Variable für den Auslöser der Verschiebung,
3. Flag-Variable für die Art der Verschiebung.

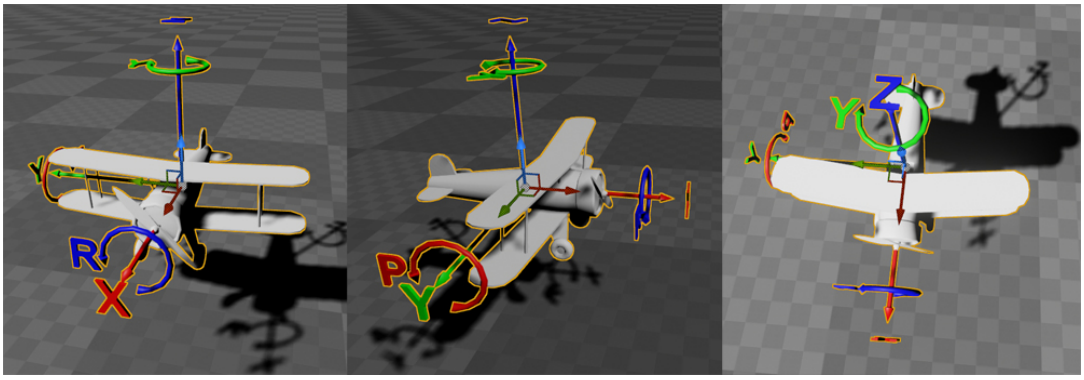


Abbildung 5.6 – Koordinatensystem und entsprechende Winkelrotationen der Unreal Engine [1]

Die Rotation des Sensors wird dabei über drei Unterparameter definiert. Diese geben an wie der Sensor über den Yaw/Pitch und Roll-Winkel geneigt wird. Eine Verdeutlichung der Winkel kann der Abbildung 5.6 entnommen werden. Dort wird das Koordinatensystem der hinter dem CARLA-Simulator stehenden Engine in Verbindung mit den Rotationswinkeln gezeigt.
Zusätzlich wird der Auslöser der Verschiebung über eine einzelne Flag-Variable festgelegt. Wie in Kapitel 4.3 beschrieben kann hier ein externes Ereignis oder wie in anderen

[1] https://carla.readthedocs.io/en/latest/python_api Abgerufen am 03.04.2024

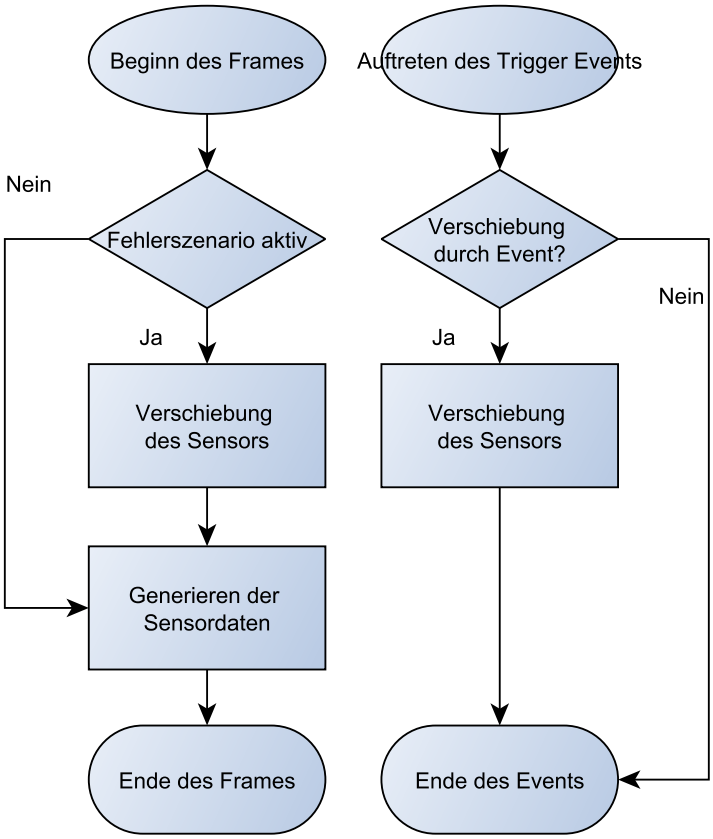


Abbildung 5.7 – Flussdiagramm des skizzierten Ablauf für einen Verschobenen Sensors abhängig vom Auslösenden Ereignisses.

Modellen ein zeitlicher Ablauf der Auslöser sein. Zusätzlich gibt es eine Flag-Variable welche die Art der Verschiebung beschreibt. Hier wird geprüft ob der Sensor lediglich zu Beginn des Fehlers einmalig verschoben wird oder ob eine Verschiebung in jedem simulierten Frame erfolgt.
Dabei kann der Ablauf der Abbildung 5.7 entnommen werden. Dort auf der linken Seite wird der Ablauf eines einzelnen simulierten Open Source bei einer zeit-basierten Verschiebung aufgezeigt. Auf der rechten Seite ist der Ablauf für eine ereignisbasierte Verschiebung aufgezeigt.

5.9 Sensor Blockade

Wie bereits im Modell beschrieben nutzt die Sensor Blockade nur folgende Grundparameter:

- Startzeitpunkt,
- Intervall,
- Degradierung.

Um eine Blockade zu simulieren wurde sich dazu entschieden tatsächliche Blockade-Objekte innerhalb des FOV zu generieren. Dies bedeutet, dass eine Anzahl an Objekten festgelegt werden kann. Zusätzlich wird dies erweitert mit der Entfernung innerhalb des FOV. Während Blockaden wie Schnee- und Schlammschichten nah am Sensors sind, tauchen störende Blockierungen wie z.B. Regen in der gesamten Reichweite des FOVs auf.
Wie bereits in Kapitel 5.7 vorgestellt wird auch das FOV durch zwei weitere Parameter, horizontal und vertikal, unterteilt. Somit kann der Spawnbereich der blockierenden Objekte genauer definiert werden, dafür werden die dort vorgestellten Algorithmen für zufällige Punkte innerhalb des FOV betrachtet. Für die vorgestellte Lebenszeit werden zwei verschiedene Möglichkeiten gewählt. Einerseits kann eine statische Lebensdauer für ein Objekt gewählt werden. Über diese kann somit auch eine permanente Blockade des Sensors erstellt werden. Alternativ kann eine zufällige Dauer über eine lineare Verteilung unter Zuhilfename einer gegebenen maximalen Lebensdauer genutzt werden. Aufgrund des Aufbaus der hinter dem Simulator stehenden Unreal Engine können die blockierenden Objekte ihre Lebenszeit selbständig überprüfen und bei Ablauf dieser Zeit entfernen sie sich selbstständig aus der Simulationsumgebung. Zusätzlich prüft dieses interne Frame-Event der blockierenden Objekte ob sich ein Objekt bewegen muss. Wie bereits im Modell vorgestellt kann der Blockade eine Fallgeschwindigkeit zugeordnet werden. Diese wird von den einzelnen blockierenden Objekten genutzt und das Objekt bewegt sich bei jedem Frame des Simulators um den gegebenen Wert.
Dieser Ablauf innerhalb eines Open Source kann der Abbildung 5.8 entnommen werden. Dort ist links der Ablauf innerhalb des Sensors und rechts der Ablauf in einem einzelnen blockierenden Objekt skizziert. Dabei wird der rechte Block für jedes Blockierende Objekt in jedem Frame durchgeführt.

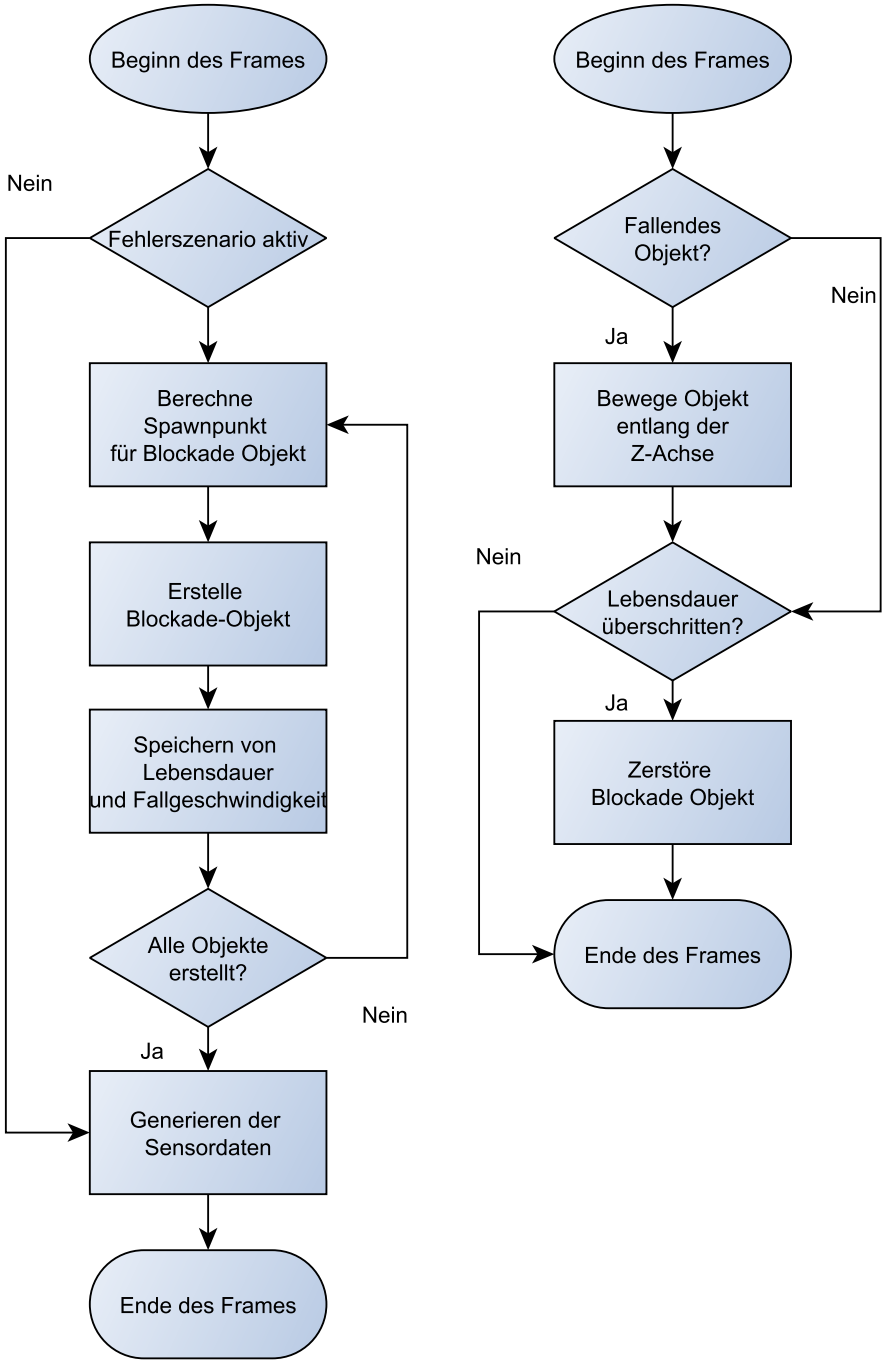


Abbildung 5.8 – Links: Flussdiagramm des Frameablaufs im Sensor. Rechts: Frame Ablauf in einem Blockade Objektes.

6 Praktische Anwendung des Simulators

Die Verwendung von Simulatoren bietet einen großen Mehrwert in der Praxis. Simulatoren können verwendet werden für das Training und die Validierung von KI-Software hinzu zum Testen von Sensorsetups und Modellen. Allerdings arbeiten bisherige Simulatoren nach dem Funktionsparadigma. Sie betrachten dabei nur eine Ideale Welt - perfekte Straßen und Verkehrsschilder als auch Sensoren die ideale Ausgaben liefern. Mittels der Erweiterung um Fehlermodelle wird es nun zusätzlich möglich ein Fehlerparadigma zu betrachten. Damit ergeben sich weitere Anwendungsfälle wie die Validierung eines ALKS mit fehlerhaften Sensoren, Training von KI zur frühzeitigen Erkennung von Fehlerbildern, und weiteres, welches die Robustheit in unbekannten Situationen enorm fördern kann.
Nachfolgend werden die Auswirkungen ausgewählter Fehler auf die generierten Punktwolken aufgezeigt. Aufbauend drauf werden praktische Use-Cases betrachtet, die Aufzeigen sollen welche Nutzen die Daten in der praktischen Anwendung besitzen. Dabei soll aufgezeigt werden wie fehlerhafte Daten in bestehende und standardisierte Strukturen eingebunden werden können. Durch diese Anbindung an standardisierte Strukturen können alle Systeme, die Radar- oder LiDAR-Daten oder daraus resultierenden Daten verwenden, analysiert, trainiert oder verifiziert werden.

6.1 Auswirkung spezifischer Fehlermodelle

Für die generelle Anwendung des Simulators gibt es ein breites Spektrum an Möglichkeiten. Während der Entwicklung wurden die vom Simulator generierten Punktwolken genauer untersucht um die Fehlermodelle zu validieren und sie zu demonstrieren. In diesem Kapitel sollen die Auswirkungen der Fehlermodelle auf die Punktwolken vorgestellt werden. Um Doppelungen zu vermeiden werden nicht alle Unterfehlermodelle beleuchtet und es werden weiterhin nur die Fehlermodelle vorgestellt die sich gut visualisieren lassen. Fehlermodelle wie z.B. ein Ausfall der Übertragung sind dabei in der Visualisierung trivial, da dort einfach ein Frame ohne Detektionen angezeigt werden würde. Zur Visualisierung werden in diesem Kapitel schwerpunktmäßig die von CARLA bereitgestellten Möglichkeiten sowie die im HDF5-Format gespeicherten Punktwolken und dem damit verbunden RadarScenes-Viewer genutzt. Aber auch andere in der Praxis gängige Datenformate, wie Rosbags[1] oder das im nachfolgenden Kapitel 6.2.1 genauer erläuterte OSI-Format sind für das Speichern der Daten eine Möglichkeit und beinhalten so die veränderten Punktwolken.
Für den Einfluss von Fehlermodellen auf die generierten Daten werden daher Frames mit und ohne Fehler visuell verglichen.

[1] https://ros.org Abgerufen am 31.05.2024

Verschiebung des Sensors In Abbildung 6.1 wird die Verschiebung des FOV eines Sensors dargestellt. Dabei stellt Abbildung 6.1a ein Frame vor der Verschiebung des Sensors dar. Dort ist gut zu erkennen wie die meisten Detektionspunkte zentral vor dem Fahrzeug in der Umgebung vorhanden sind.
So sind auf Abbildung 6.1 Detektionen auf den Häusern und Fahrzeugen links, sowie auf den Bäumen rechts zu erkennen. Die untere Abbildung 6.1b zeigt hier eine deutliche Verschiebung des FOV und die damit verbundene Verschiebung der detektierten Messpunkte. Dies wird dadurch deutlich, dass sich auf den Bäumen in der rechten Bildhälfte mehr Detektionspunkte befinden und lediglich ein einzelner Detektionspunkt am rechten Rand der Gebäude zu sehen ist.

Blockade des Sensors Ebenfalls soll der Einfluss eines Blockadefehlers betrachtet werden. Die entstandenen Visualisierungen sind Abbildung 6.2 zu entnehmen.
Dort in der Unterabbildung 6.2a ist ein Frame zu erkennen bei dem noch keine Blockade des Sensors vorhanden ist. Sämtliche Detektionen sind über das gesamte FOV des Sensors verteilt. Abbildung 6.2b zeigt hingegen einen späteren Frame desselben Simulationsdurchlaufs in dem bereits eine Blockade vorhanden ist. Diese ist daran zu erkennen, dass sich eine große und deutliche Ansammlung an Detektionen zentral in geringem Abstand vor dem Fahrzeug befindet. Da auf diesem Weg viele Radarwellen durch die Blockade abgefangen werden, sinkt die Anzahl der Detektionspunkte auf tatsächlichen Objekten deutlich ab. Im Vergleich mit der Abbildung 6.2b ist guz zuerkennen, dass dort deutlich weniger Detektionspunkte auf den umliegenden Bäumen und Häusern sowie der Straße zu erkennen sind.
Dieser Effekt kann dabei auch in dem von RadarScenes bereit gestellten RadarScenes-Viewer betrachtet werden. Dies wird in Abbildung 6.4 dargestellt.
In Abbildung 6.4a ist zu erkennen, dass der Radar-Sensor viele Detektionspunkte innerhalb des FOV vorhanden sind. Abbildung 6.4b hingegen zeigt deutlich weniger Punkte im Bereich des FOV. Zusätzlich werden dort im Ursprung des Koordinatensystems Detektionen angezeigt, die auf die blockierenden Objekte zurückzuführen sind. Die Darstellung am Ende des Fahrzeugs obliegt der Tatsache, dass das Fahrzeug lediglich als Referenz für den Ursprung des Koordinatensystems steht und nicht in direktem Bezug zur Position des Sensors dargestellt ist.
An dieser Stelle zeigt sich nochmal der Vorteil der gewählten Implementierungsart: Blockierende Objekte weisen keinen Einfluss auf die Kameradaten auf. Die hier genutzten Bilder sind gezielt mit der von CARLA gegebenen Visualisierung für Radar-Daten erfolgt, weshalb die einzelnen Detektionspunkte in der Umgebung aktiv dargestellt werden. Ist diese Visualisierung in einer Simulation deaktiviert, kann keine Beeinflussung der Kamera festgestellt werden, da die simulierten Objekte zu klein sind um von der Kamera erfasst zu werden.
Erkennbar ist dies in der Abbildung 6.3, die eine Simulation mit ähnlichem Blockadegrad darstellt, allerdings ohne aktive Visualisierung der Detektionen entnommen.

(a) Keine Verschiebung

(b) Verschiebung

Abbildung 6.1 – Frames aus der Simulation eines Radars mit und ohne Verschiebung des FOV

Dies stellt für die Praxis auch sicher, dass die eingebundenen Fehlermodelle andere Sensoren in der Simulation nicht oder nur minimal beeinflussen.

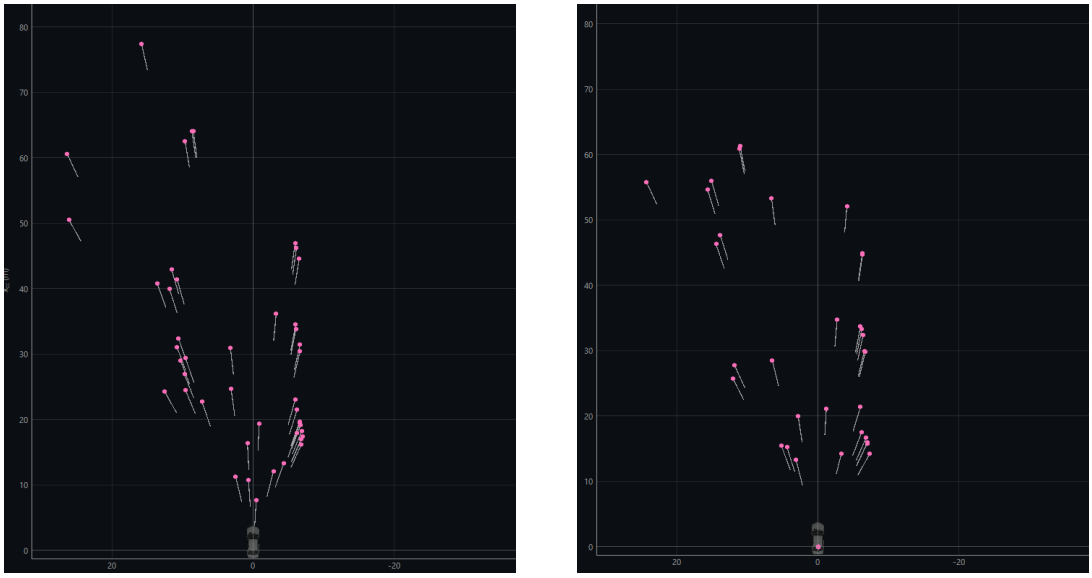
(a) Keine Blockade **(b)** Blockade

Abbildung 6.2 – Frames aus der Simulation eines Radars mit und ohne Blockade

Abbildung 6.3 – Simulation eines blockierten Radar-Sensors ohne Visualisierung der Detektionspunkte.

Die in Abbildung 6.5 visualisierten Detektionen spiegeln dabei eine Simulation mit einem ähnlichen Blockierungsgrad wie bisher betrachtete Blockadesimulationen wieder. Auf den gezeigten Bildern ist dabei ein 360°-LiDAR simuliert worden. Dort sind zentral vor dem Fahrzeug über die Kamera keine blockierenden Objekte zu erkennen. Die Abbildung 6.5b zeigt auch eine deutliche Reduzierung der Messpunkte im direkten Vergleich zu Abbildung 6.5a. Die Reduzierung der Messpunkte um das gesamte LiDAR kann damit erklärt werden, dass eine Blockade im gesamten FOV des Sensors simuliert wurde und

Blockade noch über Radar-Scenes visualisieren (Um aufzuzei-

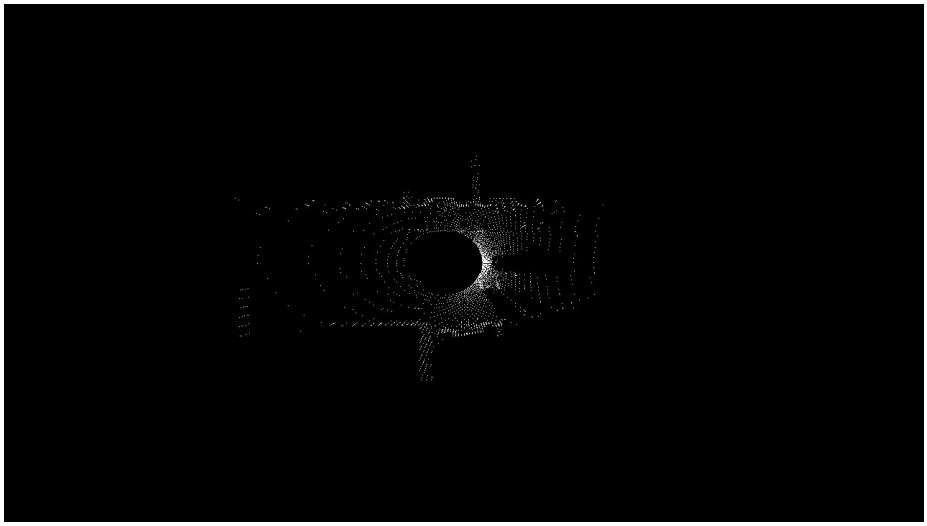


(a) Keine Blockade in externem RadarScenes-Viewer **(b)** Blockade in externem RadarScenes-Viewer

Abbildung 6.4 – Frames aus der Simulation eines Radars mit und ohne Blockade betrachtet in externem RadarScenes-Viewer

das LiDAR vollständig von der Blockade umschlossen ist.

Detektion nicht existierender Punkte Damit der Einfluss von fehlerhaft generierten Detektionspunkten untersucht werden kann, wird das erstellte Skript zum Speichern der Daten und der damit verbundene RadarScenes-RadarScenes-Viewer genutzt. Da es das Ziel dieses Fehlermodells ist, Detektionspunkte innerhalb eines gegebenen Bereiches des FOV zu generieren ist, es relevant für die Überprüfung diese Punkte erkennen zu können. Die von CARLA bereit gestellten Visualisierungswege sind dafür nicht geeignet, da sich generierte Messpunkte visuell nicht von den tatsächlichen Messpunkten zu unterscheiden sind. Da die Punkte für den RadarScenes-RadarScenes-Viewer mit Labeln versehen werden können, wurde sich für die Visualisierung dazu entschieden, generierte Messpunkte mit dem Label *Other* zu versehen.
Für den Test wurden 15 Detektionspunkte in der linken Hälfte des FOV generiert. Ein Frame aus der Simulation kann dabei in Abbildung 6.6 betrachtet werden.
Dort ist gut zu erkennen, dass generierte Messpunkte lediglich innerhalb der linken Hälfte des FOV zu finden sind. Ebenfalls sind diese über den gesamten Bereich des FOV zufällig verteilt. Dabei ist in Abbildung 6.6 oben links eine Ansammlung an Punkten zu erkennen, welches links und rechts durch einige einzelne fehlerhafte Messpunkte erweitert wird. Hier kann nun spekuliert werden, dass dies dazu führen könnte, dass Objekterkennungsalgorithmen das dortige Objekt größer wahrnehmen als es tatsächlich ist. Um eine solche These zu stützen könnten die Daten mit und ohne fehlerhafte Punkte von einem Objekterkennungsalgorithmus analysiert und die Ergebnisse verglichen werden.



(a) Keine Blockade

(b) Blockade

Abbildung 6.5 – Frames aus der Simulation eines LiDAR mit und ohne Blockade

Fehlerhafte Geschwindigkeitsmessungen Zusätzlich zu den bereits aufgezeigten Auswirkungen von Fehlern ist nachgestellt in Abbildung 6.7 die Simulation von fehlerhaften Geschwindigkeitsmessungen zu erkennen. Für diese Simulation wurden die Daten im HDF5-Format gespeichert, um anschließend über den in Kapitel 2.4.3 vorgestellten RadarScenes-Viewer betrachtet zu werden.
In Abbildung 6.7b sind hier zentral auf dem vorausfahrenden Fahrzeug rote Detektionspunkte zu erkennen. Die anderen Detektionen weisen eine geringere Geschwindigkeit auf, erkennbar durch die Darstellung in weiß. Dies spiegelt sich auch in den Radardaten in Abbildung 6.7a wieder. Dort sind einzelne Striche deutlich länger und zeigen so eine höhere Geschwindigkeit an, als die direkt anliegenden Detektionen.
Dieser hier beispielhaft erstelle Datensatz im HDF5-Format könnte nun in der Praxis angewendet werden um ein ALKS zu trainieren. Aber auch die Anwendung für das Training

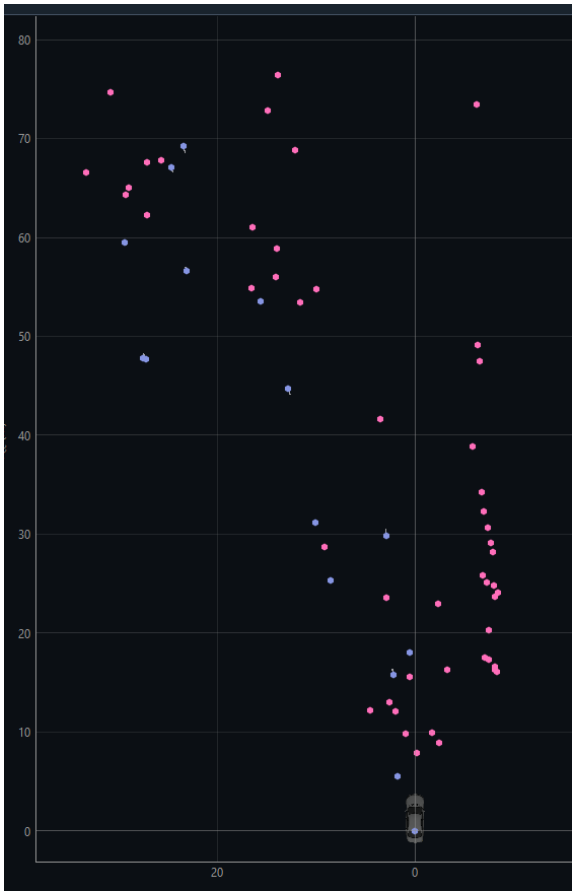
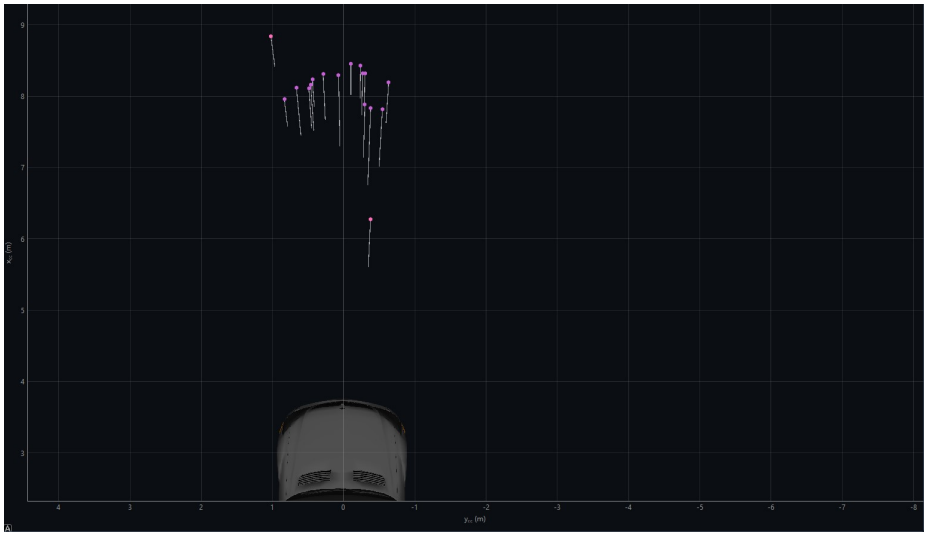


Abbildung 6.6 – Simulation von Fehlerhaft generierten Messpunkten.

einer eigenständigen Software für die Erkennung von fehlerhaften Geschwindigkeiten wäre denkbar. Weiterhin lassen sich diese Daten auch nutzen um Zuverlässigkeitsprozesse und Qualitätsstandards der Sensoren zu verifizieren. Wie solche Use-Cases aussehen können wird in nachfolgenden Kapitel erläutert.

6.2 Praktische Use-Cases

In diesem Kapitel wird vorgestellt wie der veränderte CARLA-Simulator in bestehende in der Praxis verwendete Strukturen eingebunden werden kann. Diese beiden Use-Cases zeigen zwei verschiedene Möglichkeiten auf wie der Simulator für die Verifikation von ALKS, KI-Systemen oder von Sensormodellen bzw. zur Überprüfung der Leistung dieser unter Einfluss von fehlerhaften Punktwolken genutzt werden kann.
Für diese Use-Cases werden extern vorhandene System, wie ein ALKS oder eine Functional Mock-up Unit (FMU) mit den weiterentwickelten Simulator verbunden. Durch die möglichkeit veränderte Punktwolken in bereits existirende Systeme einzubinden kann deren Anwendungsbereich verfeiernt werden oder deren Leistungsfähigkeit überprüft werden. Neben der validierung dieser Systeme generell könnte auf diesem Wege auch eine ODD verfeinert werden. So wäre eine Untersuchung denkbar, ab welchem Blockadegrad



(a) Geschwindigkeitsveränderung in RadarScenes

(b) Geschwindigkeitsveränderung in CARLA

Abbildung 6.7 – Frames aus der Simulation eines LiDAR mit und ohne Blockade

die Steuerung durch ein ALKS noch möglich ist. Anhand dieser Information könnte die ODD angepasst werden um das für das ALKS definierte Szenario weiter zu verfeinern. Somit stellt der in dieser Arbeit weiterentwickelte Simulator eine Grundlage für alle Systeme und deren Definitionen die auf Basis von Sensordaten agieren.

6.2.1 Generierung von Objektlisten

In der Praxis werden Radar- und LiDAR-Sensordaten oftmals durch einzelne KI-Systeme gesendet, die aus den Punktwolken sogenannte Objektlisten generieren. In diesem Use-Case soll nun aufgezeigt werden wie der Simulator in eine bestehende Verkettung verschiedener Tools eingebunden werden kann, um den Einfluss von Fehlern in den Sensoren auf die generierten Objektlisten zu untersuchen.
Dabei wird auf einige standardisierte Formate zurückgegriffen, die eine schnelle und un-

komplizierte Verbindung von verschiedenen System ermöglichen. Einer dieser Standards ist das sogenannte OSI[2]. Bei diesem Interface handelt es sich um die Standarisierung von Daten für den Austausch zwischen Sensoren und anderen in autonomen Fahrzeugen befindlichen Systemen. Dieses Interface kann mit dem Functional Mock-up Interface (FMI) verbunden werden. Das FMI stellt eine Schnittstelle dar um Simulationsmodelle einfach und unkompliziert verwendbar zu gestalten. So kann ein einzelnes Sensormodell simulatorunabhängig über das FMI verbunden werden, indem es in eine sogenannte FMU verpackt wird. Da es in der Praxis gängig ist, dass Sensoren bereits ein bestimmtes Preprocessing der Daten erledigen, gilt es dieses auch in Simulationen zu berücksichtigen. Da die Simulatoren allerdings nur die reinen Punktwolken bereitstellen, muss dieses Preprocessing nachgestellt werden. Dies erfolgt über Sensormodelle welche in Form von einer FMU in die Simulationskette eingebunden werden können.
Es können diverse Sensormodelle mit der FMI verknüpft werden, die dort die Punktwolken aus dem Simulator erhalten. Vorteilhaft an einer FMU ist dabei, dass die Modelle Simulatorunabhängig sind und je nach Sensormodell kann dann aus der eingehenden Punktwolke eine Objektliste generiert werden, oder es werden andere Algorithmen zur Datenverarbeitung eingebunden. Mit den Fehlermodellen ist es möglich zu untersuchen wie sich einzelne Fehler auf das Preprocessing auswirken.
Zusätzlich kann die Wirkkette weiter verfolgt und die Objektlisten mit einem ALKS oder anderer KI Software verbunden werden. So lässt sich untersuchen wie diese mit den Fehlern umgeht.
In diesem Versuchsaufbau werden die im OSI-Format dargestellten Punktwolken werden nun in die sogenannte Open Source Toolchain for Automotive Research (OSTAR)-Tool-Kette des Deutsches Zentrum für Luft- und Raumfahrt (DLR) mit einer oder mehreren FMUs eingespeist [33]. Diese Tool-Kette ist dabei darauf ausgelegt das CARLA eigene Datenformat der Sensor-Daten in OSI konforme Datenstrukturen zu überführen. Zusätzlich bietet diese Tool-Kette die Möglichkeit die Daten im OSI-Format zu speichern und macht sie somit für andere Systeme, die diese als Input benötigen verwendbar. Für diesen Versuchsaufbau wird allerdings die Möglichkeit des Ladens von FMUs genutzt, da so die Daten direkt im Zusammenhang mit dieser untersucht werden könnten. Dieser Ablauf wird schematisch in Abbildung 6.8 dargestellt. Dort werden auch die einzelnen verwendeten Dienste benannt. Genauere Informationen sowie der Quellcode bezüglich der OSTAR-Tool-Kette können den Github-Repositories [34] und [33] entnommen werden.
Für den Versuchsaufbau ist es allerdings relevant auch die FMU genauer zu betrachten. Dabei wird die von Holder et. al. entwickelte FMU verbunden [35]. Diese FMU ist in der Lage die von der Tool-Kette bereitgestellten veränderten Punktwolken in Objektlisten zu überführen. Genauere Informationen zu der FMU und ihrer Funktionsweise können der zugehörigen wissenschaftlichen Arbeit [35] entnommen werden.
Die so aufgebaute Verknüpfung des veränderten CARLA-Simulators mit den soeben vor-

[2] https://www.asam.net/standards/detail/osi/ Abgerufen am 29.04.2024

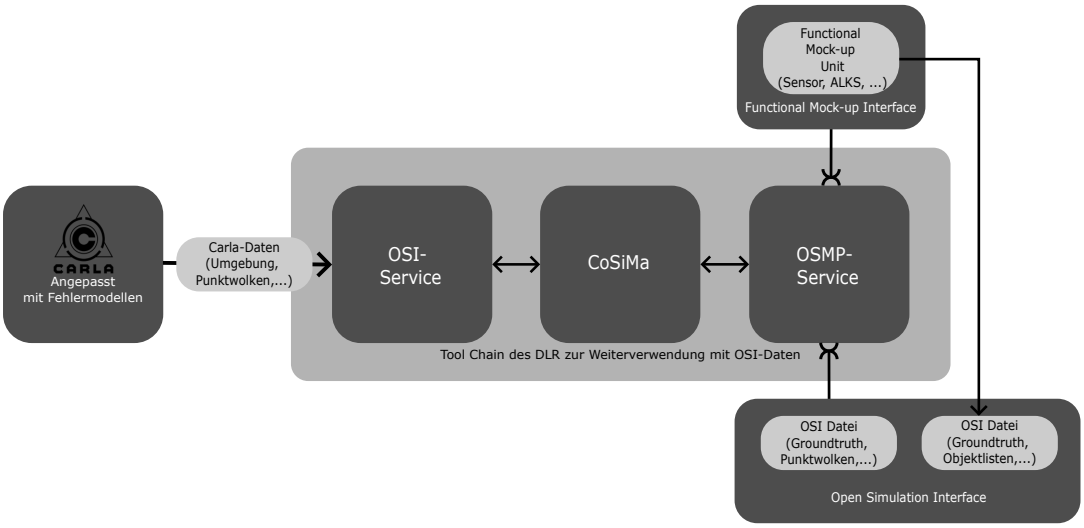


Abbildung 6.8 – Schematischer Aufbau der Tool-Kette des DLR und dem angepassten CARLA-Simulators.

gestellten Systemen kann nun genutzt werden, um die Auswirkung von Fehlermodellen abhängig von deren Ausprägungsstärke zu untersuchen. So kann nun über die Parametrisierung der Fehler, als auch die Auswahl des Szenarios der Einfluss auf die Objektlisten untersucht werden. Eine genaue Analyse würde hier mehrere von einander unabhängige Simulationen desselben Szenarios vergleichen. Eine Simulation wäre dabei als Referenz ohne die Einwirkung eines Fehlers, alle anderen Simulationen würden dabei verschiedene Fehler in unterschiedlichen Schweregraden simulieren. Aufgrund der sehr hohen Anforderungen an die Hardware des Simulators und der Verkettung der Tools konnte hier lediglich der Proof-of-Concept aufgestellt werden. Eine Generierung von Daten und die Auswertung dieser war daher nicht möglich.
Dieser theoretische Versuchsaufbau zeigt allerdings, dass die generierten Fehlermodelle mit in der Praxis angewandten Standards schnell und unkompliziert für weitere relevante Analysen verwendet werden können.

6.2.2 ecu.test

Damit ein ALKS für den Straßenverkehr zugelassen wird, muss dessen Performanz validiert werden. Dies erfolgt wie bereits vorgestellt über verschiedene KPIs.
Dabei können diese in der Praxis auf verschiedenem Wege bestimmt werden. Eine Möglichkeit ist das von TraceTronic entwickelte Programm ecu.test. Dieses ist in der Lage sich mit verschiedenen Simulatoren und ALKS Implementierungen zu verbinden und während einer Simulation die KPIs zu bestimmen.
Für diesen Use-Case findet nun die Verknüpfung von einem ALKS von Autoware[3] sowie ecu.test[4] statt. Dabei nutzt das ALKS Kamera- sowie LiDAR-Sensorik. In dieser Ver-

[3] https://autoware.org Abgerufen am 08.05.2024
[4] https://www.tracetronic.de/produkte/ecu-test/ Abgerufen am 27.05.2024

suchsaufbau, wird nun ein LiDAR-Sensor verwendet welcher entsprechend Fehlerhafte Daten generiert.
Diese Verknüpfung kann der Abbildung 6.9 entnommen werden.

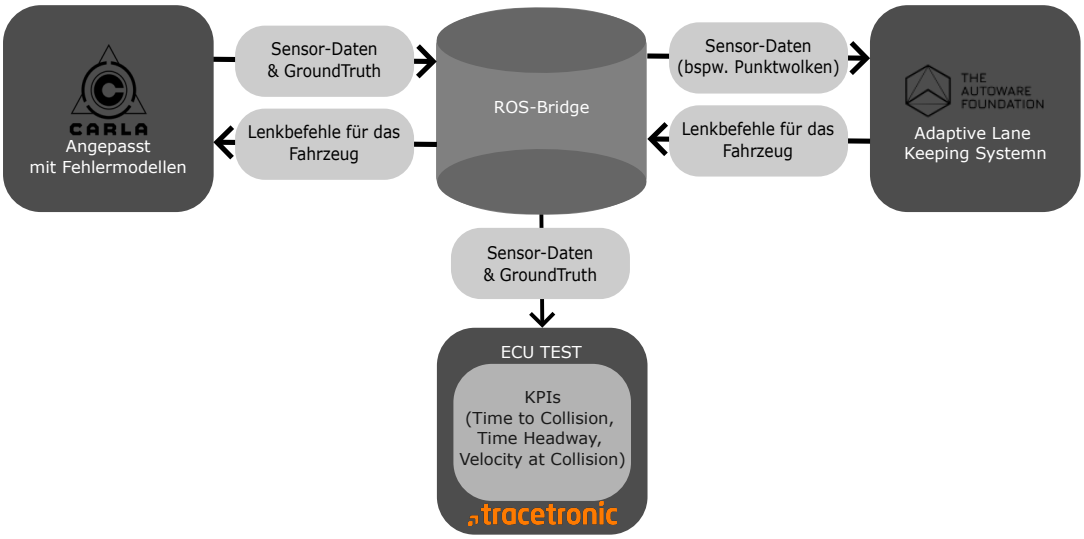


Abbildung 6.9 – Skizzierter Versuchsaufbau aus dem angepassten Simulator, dem Autoware-ALKS und ecu-test.

Dabei ist zu sehen, dass ecu.test hier mit der Übersicht auf die gesamte Simulation diese KPIs bestimmt. So werden von dem CARLA-Simulator Informationen in Form einer Ground truth bereitgestellt. Der bereits vorgestellte OSI-Standard definiert, dass die Ground truth alle Informationen über alle simulierten Objekte beinhaltet [36]. Darunter fallen auch die durch den Fehler veränderten Sensordaten. Für die Abbildung wurden diese getrennt aufgelistet, da die gesamte Ground truth dem ALKS nicht bereit gestellt wird, sondern dieses nur die Sensor-Daten erhält.
Auf Basis dieser Verknüpfungen ist es nun möglich die KPIs für ein ALKS zu bestimmen. Dabei können die für Zertifizierungsverfahren relevanten Szenarien simuliert werden. Dies bietet die Möglichkeit, dass der direkte Einfluss von Fehlern auf ein ALKS visuell im Fahrverhalten des ALKS erkennbar wird, aber auch in Form der KPIs betrachtet werden kann. Es wäre zusätzlich möglich auf diesem Weg zu analysieren, wie gravierend ein Fehler sein muss, damit die Funktionsfähigkeit des ALKS eingeschränkt ist. Dort könnten unter anderem KPIs wie die TTC betrachtet werden. Da es hier direkte vorgaben gibt, welche Werte eingehalten werden müssen und diese so in eine Korrelation mit dem Schweregrad des Fehlers gebracht werden können.
Für ein ALKS wäre der Best-Case, dass ein Nachweis darüber erfolgen kann, dass entweder dieses nicht durch den Fehler betroffen ist, oder Schutzmechanismen den Fehler bereits vor einem kritischen Zeitpunkt erkennen können.
Leider konnten über die Entwicklungszeit hinweg, aufgrund hoher Hardware Anforderungen, keine umfangreichen Analysedaten generiert werden. Ein Proof-of-Concept konnte aber in Zusammenarbeit mit TraceTronic erbracht werden.

Proof-of-Concept Für den Proof-of-Concept wurde ein ALKS von Autoware mit der von TraceTronic bereitgestellten Anwendung ecu.test verbunden. Es wurden vier verschiedene Simulationen durchgeführt. Dabei wurden drei Simulationen mit einem fehlerhaften LiDAR-Sensor durchgeführt, sowie eine Kontrollsimulation mit einem unbeeinträchtigten Sensor. Für den Test wurde ein Datenübertragungsfehler in unterschiedlichen Ausprägungen simuliert. Diese Ausprägungen können der Tabelle 6.1 entnommen werden.

SimulationsID	Fehlerdauer	Fehlerintervall	Ausfallzeit in Prozent
0	-	-	0%
1	1s	2s	50%
2	1.5s	3s	50%
3	1.5s	2s	75%

Tabelle 6.1 – Gegenüberstellung der einzelnen Simulationsparameter

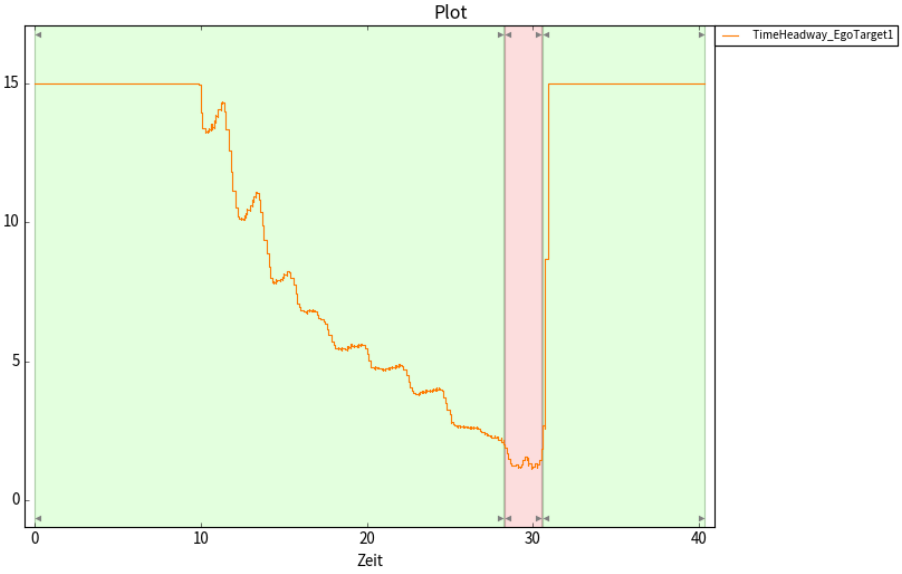
Da die Simulationen vor allem dafür gedacht waren zu überprüfen ob ein Einfluss auf das ALKS vorhanden ist, wurden die Parameter hier deutlich verstärkt. Für die Simulation wurde dabei ein Szenario gewählt bei dem eine Kollision an einer Kreuzung vermieden werden sollte. Daher sind für diese Simulationen Parameter gewählt worden, die deutlich über realistischen Werten liegen. Dabei spiegelt die Simulation mit der ID 0 den funktionierenden Sensor wieder. Die Simulationen mit den IDs 1 und 2 sind hier so parametrisiert, dass der Sensor während der Hälfte der Simulation keine Daten generiert. Dabei wurden allerdings unterschiedliche Ausprägungen gewählt. Dies führt dazu, dass der Sensor zu unterschiedlichen Zeitpunkten innerhalb der Simulation funktionsfähig ist, so ist der Ausfall des Sensors zum Zeitpunkt einer Kurve kritischer als beim Befahren einer geraden Straße. Für die letzte Simulation wurde ein deutlich massiveres Fehlerbild mit einem nahezu vollständigen Ausfall des Sensors gewählt. Hier wird auch der größte Einfluss auf die betrachteten KPIs erwartet. In der Praxis sollten bei diesen massiven Ausfallraten bereits Schutzmechanismen und ähnliches greifen.
Die relevanten Ergebnisse der Simulation können der Tabelle 6.2 entnommen werden.

SimulationsID	TTC	THW
0	0.8s	2.2s - Test Failed
1	0.67s	3.1s - Test Failed
2	0.51s	2.8s - Test Failed
3	0.04s	4.2s - Test Failed

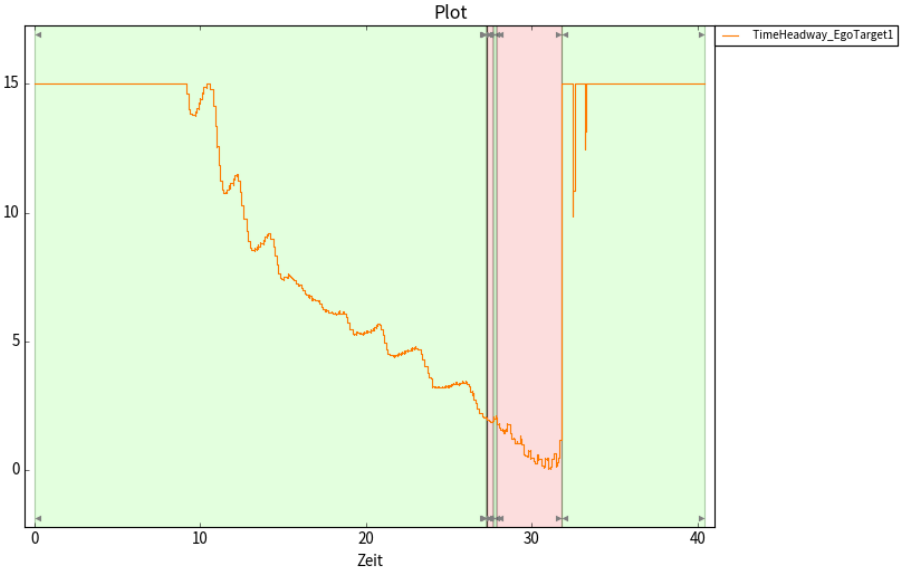
Tabelle 6.2 – Gegenüberstellung der einzelnen Simulationsparameter

Als Vergleichswert für die fehlerbehafteten Simulationen wurde so die TTC von 0.8 sec, sowie die THW von 2.2s ermittelt. Dabei ist zu beachten, dass für die THW der Test als Failed angesehen wird, da der definierte Minimalwert unterschritten wurde. Dieser Minimalwert ist dabei für alle Simulationen identisch. Relevant ist dabei vorallem die Zeit wie lange dieser Minimalwert unterschritten ist. Bei der Vergleichssimulation mit dem funktionierenden Sensor findet hier eine Unterschreitung des Wertes über eine Dauer

von 2.2s statt.
In der Betrachtungen der Simulationen mit den IDs 1 und 2 fällt auf, dass die TTC und THW beide in einem ähnlichen Größenverhältnis stehen. Die unterschiedlichen Abweichungen können dabei verschiedene Gründe aufweisen. Einerseits sind die Intervalle unterschiedlich, so kann es passieren, dass für das ALKS kritischere Zeitpunkte bei einer Simulation entfallen, während sie bei der anderen durch andere Intervalle vorhanden sind. Andererseits muss auch klar hervorgehoben werden, dass es bei Simulationen alleine immer zu einer Schwankung der Werte kommen kann. Dies obliegt der Art von Simulationen und den dahinterliegenden Ansätzen. Ein genauer Grund für die Unterschiede der Werte kann daher nicht genannt werden und es bleibt lediglich die Spekulation über die Ursachen der Abweichungen. Da es sich bei dem ALKS auch um ein bereits trainiertes System handelt, ist dessen exakte Funktionsweise nicht bekannt. Dennoch zeigen die beiden Simulationen mit den IDs 1 und 2 eine deutliche Degradierung innerhalb der betrachteten KPI.
Die größte Veränderung der KPI erfolgt dabei, wie erwartet, in der Simulation 4 mit dem am stärksten ausgeprägten Fehler. Die dortige TTC von 0.04s lässt darauf schließen, dass es innerhalb der Simulation beinahe zu einer Kollision kam. Ein so massiver Fehler zeigt somit eine deutliche Auswirkung auf die Leistungsfähigkeit des verwendeten ALKS. Auch die THW unterstreicht diese Aussage deutlich. Die Unterschreitung des gewählten Minimalwertes für die THW ist fast doppelt so lange, wie bei einem regulären Sensor.
In Abbildung 6.10 sind die beiden zeitlichen Verläufe für die THW der Simulationen 0 und 1 zu erkennen. Dabei ist auf der X-Achse der zeitliche Verlauf der Simulation zu erkennen. Auf der Y-Achse hingegen ist die THW eingetragen. Die Hintergrundfarbe grün gibt dabei an, dass die THW über dem festgelegten Minimalwert von zwei liegt.
Bei der Roten Hintergrundfarbe ist zu erkennen, dass dieser Minimalwert unterschritten wurde. So lässt sich in Abbildung 6.10a erkennen, dass die THW bei der Annäherung an die Kreuzung gegen Ende der Simulation unterschritten wird. Abbildung 6.10b hingegen zeigt, dass der Grenzwert bereits früher unterschritten wird und diese auch länger andauert. So kann angenommen werden, dass in Simulation 3 das Fahrzeug mit einer höheren Geschwindigkeit an die Kreuzung heranfuhr und so die THW früher unterschritten wurde und es so auch länger dauert bis die THW den Minimalwert wieder überschreitet.
MitHilfe dieser Tests wird aufgezeigt, dass eine Degradierung des ALKS durch Fehler auftritt. Damit eine genauere Aussage über den Umfang der Degradierung getroffen werden kann, müssen weitere Tests durchgeführt werden. So sollte das ALKS mit weiteren Fehlerausprägungen und auch in weiteren Szenarien getestet werden. Es wäre auch Vorteilhaft mehrere Simulationen mit denselben Parametern durchzuführen, um Ausreißer in den Ergebnissen zu erkennen.



(a) THW der Simulation 0



(b) THW der Simulation 3

Abbildung 6.10 – Vergleich der THW von Simulation 0 und 3

7 Fazit

Das Ziel dieser Arbeit war es einen ersten Schritt in Richtung der Fehlerparadigmen in Simulatoren zu gehen und die Wichtigkeit dieses Schrittes zu verdeutlichen. Hier wurde zu Beginn eine umfangreiche Literaturrecherche zu verschiedensten Fehlern von Radar und LiDAR-Sensoren und auf die Auswirkungen in den generierten Daten betrieben. Auf Basis dieser Literaturrecherche wurden die Effekte kategorisiert und Fehlermodelle entwickelt welche die Effekte auf die Daten beschreiben. Diese Modelle wurden anschließend in den Open Source CARLA-Simulator eingebunden, wodurch synthetische Daten mit verschiedenen Fehlereinflüssen generiert werden können. Zusätzlich wurde aufgezeigt, dass der Simulator an gängige Systeme der Praxis angebunden werden kann und so auch das Standardinterface OSI unterstützten werden kann. In Verbindung mit dem Unternehmen TraceTronic konnte unter der Verwendung der Software ecu.test der Einfluss von fehlerinjizierten Sensor-Daten auf ein ALKS nachgewiesen werden. Diese ersten Ergebnisse zeigen die Degradierung von einem Open Source ALKS im Fehlerfalle der Perzeptionssensorik. In dieser Arbeit wurde so ein Grundstein gelegt, anhand dessen weitere Forschung in diesem Bereich betrieben werden kann. Zusätzlich wurde aufgezeigt, wie diese Forschung in der Praxis Anwendung finden kann und bereits heute für die Verbesserung der Systeme genutzt werden kann.
Trotz der umfangreichen Simulationsmöglichkeiten die sich durch die implementierten Fehlermodelle und deren Parametrisierung ergeben, zeigten sich bereits während der Entwicklung Limitationen in der Umsetzung und den aktuellen Möglichkeiten Fehler abzubilden. Hier war die hohe Hardwareanforderung ein früh erkanntes Problem. Durch die Art der Implementierung der Fehlermodelle, in Verbindung mit leistungsarmer Hardware und der damit reduzierten Anzahl von Frames, können sich Fehler anders in den generierten Daten auswirken als zuvor angenommen. So kann es dazu kommen, dass für Fehler eine gewisse Fehlerdauer angenommen wird, der Fehler final aber nur wenige fehlerhafte Daten generiert. Hier wäre es zu empfehlen in zukünftigen Projekten zusätzlich zu festen Fehlerdauern dies stattdessen an die Anzahl von generierten Frames zu knüpfen. Dies würde dafür sorgen, dass generierte Daten mehr an die Realität angenähert werden.
Zusätzlich sollten in zukünftigen Untersuchungen auch verschiedene Fehlermodelle miteinander verschaltet werden können. So könnte der Einfluss auf Sensoren betrachtet werden, wenn diese nicht nur einem spezifischen Fehlermodell unterliegen, sondern verschiedene Fehler den Sensor beeinflussen. Während in dem hier entwickelten Simulator bereits verschiedene Fehlermodelle mit einander verknüpft werden können, besteht nicht die Möglichkeit, dass ein Sensor mehreren Fehlern mit demselben Effekt unterliegt. So wäre es in der Realität denkbar, dass ein LiDAR-Sensor mit einer Schneeschicht bedeckt ist und zeitgleich nebliges oder regnerisches Wetter vorhanden ist. Die aktuelle Implementierung lässt dabei nur einen der beiden Zustände zu und es ist keine Simulation von beiden Effekten parallel möglich. Des Weiteren könnten Fehlermodelle verfeinert

werden. Zum Beispiel könnte bei einer Kollision die genaue Verschiebung des Sensors anstatt einer zufälligen Verschiebung berechnet werden.
Auch wurde aufgezeigt, dass eine Beeinflussung der Systeme durch die veränderten Daten vorhanden ist und das dies mit in der Praxis gängigen Systeme erkennbar und messbar ist. Schwerpunktmäßig wurde hier aufgezeigt, dass kritische KPIs wie die TTC unterschritten werden und auch die THW in deutlich größerem Rahmen, als im Vergleich mit funktionierenden Sensoren, unterschritten werden. Für zukünftige Arbeiten könnten hier umfangreichere Simulationen gefahren werden um genaue Informationen über die Degradierung sowie das Zusammenspiel der Parameter herauszufinden. Auch wäre es interessant dort die Einflüsse von anderen Fehlermodellen zu betrachten. In dieser Arbeit wurden lediglich Simulationen mit einem einfachen Datenübertragungsfehler simuliert.

Abkürzungsverzeichnis

ALKS	Adpative Lane Keeping System
API	Application Interface
DLR	Deutsches Zentrum für Luft- und Raumfahrt
float	Gleitkommazahl
FMCW	Frequency-modulated continuous wave
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FOV	Field of View
HDF5	Hierarchical Data Format Version 5
KI	Künstlichen Intelligenz
KPI	Key Performance Indicator
LiDAR	Light detection and ranging
ODD	Operational Design Domain
OSI	Open Simulation Interface
OSTAR	Open Source Toolchain for Automotive Research
Radar	Radio detection and ranging
RCS	Radar Cross Section
ROS	Robot Operating System
SAE	Society of Automotive Engineers
THW	Time Headway
TTC	Time-To-Collision
UNECE	United Nations Economic Commission for Europe

Glossar

ALKS Ein System das die Steuerung eines Fahrzeugs innerhalb einer gegeben ODD übernimmt.

Blind Spots Bereiche die nicht erkannt werden durch Sensoren.

Flag-Variable Eine Variable die einen definierten Zustand beschreibt und nur einen von mehreren möglichen Zuständen haben kann.

Ghost Target Detektion eines Objektes wo kein Tatsächliches Objekt vorhanden ist

Ground truth Übersicht über die simulierte Umgebung.

ODD Beschreibt die Situation in der ein ALKS die Steuerung des Fahrzeugs übernimmt.

Preprocessing Aufbereitung der Radar-Daten durch den Radar-Sensor selbst.

RadarScenes-Viewer Ein Programm zur Darstellung von Daten.

Seed Startwert für einen Zufallszahlengenerator.

Spawnbereich Bereich in dem ein neues Objekt erscheinen kann.

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] *Warum autonomes Fahren von synthetischen Daten abhängt*, https://www.automotiveit.eu/technology/autonomes-fahren/warum-autonomes-fahren-von-synthetischen-daten-abhaengt-676.html, Access: 14.05.2024.

[2] A. Haider u. a., „A Methodology to Model the Rain and Fog Effect on the Performance of Automotive LiDAR Sensors“, *Sensors*, Jg. 23, Nr. 15, 2023, ISSN: 1424-8220. DOI: 10.3390/s23156891. Adresse: https://www.mdpi.com/1424-8220/23/15/6891.

[3] R. Rinaldo u. a., „Towards Modelling Sensor Failures in Automotive Driving Simulators“, Jan. 2023, S. 326–333. DOI: 10.3850/978-981-18-8071-1_P124-cd.

[4] J. Vargas u. a., „An Overview of Autonomous Vehicles Sensors and Their Vulnerability to Weather Conditions“, *Sensors*, Jg. 21, Nr. 16, 2021, ISSN: 1424-8220. DOI: 10.3390/s21165397. Adresse: https://www.mdpi.com/1424-8220/21/16/5397.

[5] M. Hartstern u. a., „Conceptual Design of Automotive Sensor Systems : Analyzing the impact of different sensor positions on surround-view coverage“, in *2020 IEEE SENSORS*, 2020, S. 1–4. DOI: 10.1109/SENSORS47125.2020.9278638.

[6] M. Bartock u. a., „Foundational PNT Profile: Applying the Cybersecurity Framework for the Responsible Use of Positioning, Navigation, and Timing (PNT) Services“, National Institute of Standards und Technology, Gaithersburg, Maryland, USA, Techn. Ber. NIST Internal Report 8323 Rev. 1, 2023. DOI: https://doi.org/10.6028/NIST.IR.8323r1.

[7] J. Hecht, „Lidar for Self-Driving Cars“, *Opt. Photon. News*, Jg. 29, Nr. 1, S. 26–33, Jan. 2018. DOI: 10.1364/OPN.29.1.000026. Adresse: https://www.optica-opn.org/abstract.cfm?URI=opn-29-1-26.

[8] *Anatomy of an Electromagnetic Wave*, https://science.nasa.gov/ems/02_anatomy, Access: 02.08.2023.

[9] M. Murad u. a., „Next generation short range radar (SRR) for automotive applications“, in *2012 IEEE Radar Conference*, 2012, S. 0214–0219. DOI: 10.1109/RADAR.2012.6212139.

[10] W.-J. Liao u. a., „Radar Cross Section Enhancing Structures for Automotive Radars“, *IEEE Antennas and Wireless Propagation Letters*, Jg. 17, Nr. 3, S. 418–421, 2018. DOI: 10.1109/LAWP.2018.2793307.

[11] R. Wills, „Calculation of radar cross section for an active array“, in *IEE Colloquium on Antenna Radar Cross-Section*, 1991, S. 8/1–8/4.

[12] W. Wiesbeck u. a., „RADAR 2020: THE FUTURE OF RADAR SYSTEMS“, Juli 2015.

[13] H. Groll und J. Detlefsen, „History of automotive anticollision radars and final experimental results of a mm-wave car radar developed on the Technical University of Munich“, in *Proceedings of International Radar Conference*, 1996, S. 13–17. DOI: 10.1109/ICR.1996.573762.

[14] L. Feng u. a., „Performance Improvement for Single-Photon LiDAR with Dead Time Selection“, *International Journal of Aerospace Engineering*, Jg. 2022, S. 6 847 331, Juli 2022, ISSN: 1687-5966. DOI: 10.1155/2022/6847331. Adresse: https://doi.org/10.1155/2022/6847331.

[15] A. Ngo, „A Methodology for Validation of a Radar Simulation for Virtual Testing of Autonomous Driving“, Jan. 2023.

[16] *UN Regulation No. 157 - Automated Lane Keeping Systems (ALKS)*, https://unece.org/transport/documents/2021/03/standards/un-regulation-no-157-automated-lane-keeping-systems-alks, Access: 03.05.2024.

[17] O. Schumann u. a., „RadarScenes: A Real-World Radar Point Cloud Data Set for Automotive Applications“, *CoRR*, Jg. abs/2104.02493, 2021. arXiv: 2104.02493. Adresse: https://arxiv.org/abs/2104.02493.

[18] M. Hartung, „Verfahren zur Erhöhung der Zuverlässigkeit und Verlängerung der Lebensdauer elektronischer Baugruppen durch dynamische Belastungsreduzierung während des Betriebs“, 2017. DOI: https://doi.org/10.18453/rosdok_id00002308.

[19] S.-J. Kimmerle und H.-D. Ließ, „Zuverlässigkeit elektrischer Komponenten im Bordnetz: Jedes Bauteil zählt“, Jg. 19, S. 42–45, Sep. 2019.

[20] X. Du u. a., „Event-Triggered Robust Fusion Estimation for Multi-Sensor Time-Delay Systems with Packet Drops“, *Applied Sciences*, Jg. 13, Nr. 15, 2023, ISSN: 2076-3417. DOI: 10.3390/app13158778. Adresse: https://www.mdpi.com/2076-3417/13/15/8778.

[21] *Toyota ruft in Europa 319.000 Autos zurück*, 2022. Adresse: https://www.focus.de/auto/news/c-hr-und-yaris-toyota-ruft-in-europa-319-000-autos-zurueck_id_68958592.html (besucht am 4. Dez. 2023).

[22] J. R. V. Rivero u. a., „Characterization and simulation of the effect of road dirt on the performance of a laser scanner“, in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, S. 1–6. DOI: 10.1109/ITSC.2017.8317784.

[23] E. Yeh u. a., „Security in automotive radar and vehicular networks“, *Microwave Journal*, Jg. 60, S. 148–164, Mai 2017.

[24] M. Trierweiler u. a., „Influence of sensor blockage on automotive LiDAR systems“, in *2019 IEEE SENSORS*, 2019, S. 1–4. DOI: 10.1109/SENSORS43011.2019.8956792.

[25] B. Schlager u. a., „Contaminations on Lidar Sensor Covers: Performance Degradation Including Fault Detection and Modeling as Potential Applications“, *IEEE Open Journal of Intelligent Transportation Systems*, Jg. 3, S. 738–747, 2022. DOI: 10.1109/OJITS.2022.3214094.

[26] S. Hasirlioglu u. a., „Test methodology for rain influence on automotive surround sensors“, in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2016, S. 2242–2247. DOI: 10.1109/ITSC.201,6.7795918.

[27] F. Hau u. a., „Influence of vibrations on the signals of automotive integrated radar sensors“, in *2017 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*, 2017, S. 159–162. DOI: 10.1109/ICMIM.2017.7918881.

[28] B. Schlager u. a., „Automotive Lidar and Vibration: Resonance, Inertial Measurement Unit, and Effects on the Point Cloud“, *IEEE Open Journal of Intelligent Transportation Systems*, Jg. 3, S. 426–434, 2022. DOI: 10.1109/OJITS.2022.3176471.

[29] F. Hau u. a., „The degradation of automotive radar sensor signals caused by vehicle vibrations and other nonlinear movements“, *Sensors*, Jg. 20, Nr. 21, S. 6195, 2020. DOI: 10.3390/s20216195.

[30] S. Alland u. a., „Interference in Automotive Radar Systems: Characteristics, Mitigation Techniques, and Current and Future Research“, *IEEE Signal Processing Magazine*, Jg. 36, Nr. 5, S. 45–59, 2019. DOI: 10.1109/MSP.2019.2908214.

[31] S. Tanis, „Automotive Radar Sensors and Congested Radio Spectrum: An Urban Electronic Battlefield?“, *ADI Analoge Dialogue*, Jg. 52, Juli 2018.

[32] H. Shin u. a., „Illusion and Dazzle: Adversarial Optical Channel Exploits Against Lidars for Automotive Applications“, in *Cryptographic Hardware and Embedded Systems – CHES 2017*, W. Fischer und N. Homma, Hrsg., Cham: Springer International Publishing, 2017, S. 445–467, ISBN: 978-3-319-66787-4.

[33] *The Open Source Toolchain for Automotive Research (OSTAR) as a Closed-Loop System Simulation Framework for Integration Test and Validation*, https://elib.dlr.de/195865/, Access: 01.05.2024.

[34] *Quickstart for OSTAR*, https://github.com/DLR-TS/OSTAR-Quickstart, Access: 01.05.2024.

[35] M. Holder u. a., „The Fourier Tracing Approach for Modeling Automotive Radar Sensors“, in *2019 20th International Radar Symposium (IRS)*, 2019. DOI: 10.23919/IRS.2019.8768113.

[36] *Ground truth*, https://opensimulationinterface.github.io/osi-antora-generator/asamosi/latest/interface/architecture/ground_truth.html, Access: 06.05.2024.