Schlussbericht
(gem. Anlage 2 zu Nr. 8.2 NKBF 98)

Vorhabensbezeichnung	RDV – Real Driving Validation: Erweiterung der Überprüfbarkeit von Continuous SW Integration in Kommunikation mit Fahrzeugen im Feld
Akronym	RDV
Förderkennzeichen	19A21051
Laufzeit	01.01.2022 bis 30.06.2025
Zuwendungsempfänger	Institut für Qualitäts- und Zuverlässigkeitsmanagement GmbH (kurz: IQZ GmbH)
Berichtszeitraum	Schlussbericht, 01.01.2022 – 30.06.2025
Koordinator	Dr.-Ing Fabian Plinke, [plinke@iqz-wuppertal.de]
Ort, Datum	Hamburg, den 29.12.2025

Inhaltsverzeichnis

I. Kurzdarstellung 3

1.1. Aufgabenstellung 3

1.2. Voraussetzungen unter denen das Vorhaben durchgeführt wurde 3

1.3. Planung und Ablauf des Vorhabens 3

1.4. Wissenschaftlich-technischer Stand an den angeknüpft wurde 4

1.5. Zusammenarbeit mit anderen Stellen 5

II. Eingehende Darstellung 6

2.1. Verwendung der Zuwendung und des erzielten Ergebnisses im Einzelnen 6

2.2. Wichtigste Positionen des zahlenmäßigen Nachweises 24

2.3. Notwendigkeit und Angemessenheit der geleisteten Arbeit 24

2.4. Voraussichtlicher Nutzen, Verwertbarkeit der Ergebnisse im Sinne des fortgeschriebenen Verwertungsplan 25

2.5. Fortschritte an anderen Stellen 25

2.6. Veröffentlichungen der Ergebnisse 25

I. Kurzdarstellung

1.1. Aufgabenstellung

Gesamtprojekt

Das Projekt Real Driving Validation (RDV) adressierte die Herausforderung, eine kontinuierliche und belastbare Qualitätsaussage für automatisierte Fahrfunktionen von Level 3 bis 5 auf Gesamtfahrzeugebene („Grey Box“) zu ermöglichen. Auf Basis realer Fahrdaten aus dem Feld und Hauptuntersuchungen wurde eine effiziente Methodik zur Validierung KI-basierter Fahrfunktionen entwickelt, die mithilfe stochastischer Modelle, Ausfallwahrscheinlichkeiten sowie Markov-Modellen kritische Szenarien gezielt identifiziert, simuliert und bewertet. Dadurch können Entwicklungs-, Update- und Zulassungsprozesse beschleunigt und in digitale Ökosysteme sowie Normungs- und Standardisierungsprozesse überführt werden.

Teilvorhaben IQZ

Im Rahmen von AP2 lag der Fokus der Aufgabenstellung des IQZ auf der Entwicklung einer zuverlässigkeits-, verfügbarkeits- und sicherheitstechnischen Modellierung des Gesamtsystems (hier in Form eines „Autonomous Driving Stacks“ (kurz AD-Stack).

Ziel war die Entwicklung eines Simulationsmodells zur Bewertung der Sicherheitskritikalität von Verkehrssituationen unter Berücksichtigung von Fahrzeug, Umgebung und weiteren Verkehrsteilnehmern. Dafür sollte ein Gesamtzustandsraum des Fahrzeugs einschließlich Betriebs- und Fehlerzuständen modelliert werden der insbesondere Aspekte moderner, resilienter Fahrzeugarchitekturen, wie ein dynamisches Fehlermanagement (FDIR), beinhaltet. Die Ergebnisse dienen als Grundlage für zukünftige AD-Referenzsysteme (SAE Level 4–5) und die Weiterentwicklung standardisierter, datengetriebener Modellierungsansätze.

1.2. Voraussetzungen unter denen das Vorhaben durchgeführt wurde

Das Vorhaben wurde als geschlossenes Konsortialprojekt (ohne Zugehörigkeit zu einer Projektfamilie) durchgeführt. Das Konsortium setzte sich aus 6 Unternehmen, davon 4 kleinen und mittleren Unternehmen (KMU) sowie 2 wissenschaftlichen Einrichtungen und 2 Prüforganisationen zusammen.

Das IQZ bietet eine Vielzahl hoch spezialisierter Analysemethoden im Bereich der Sicherheits- und Zuverlässigkeitstechnik an und verfügt über breites Expertenwissen für simulationsgestützte Systemmodellierung mittels Monte-Carlo-Methode. Die zuverlässigkeitstechnische Systemmodellierung ist ein wesentlicher Baustein im Rahmen der Zuverlässigkeits- und Sicherheitsanalysen des Entwicklungsprozesses von automatisierten und autonomen Fahrsystemen.

Die Zusammensetzung des Konsortiums ermöglichte die Abdeckung unterschiedlicher Simulationslevel im Rahmen der Produktentwicklung. Während EMI und IQZ die Model-In-The-Loop (MIL)-Level abdecken, konnten durch dSPACE, Liangdao und TTTech Auto Germany, Software- und Hardware-In-The-Loop (SIL/MIL)-Betrachtungen beigesteuert werden. Somit waren die Voraussetzungen für eine umfassende simulationsgestützte Produktvalidierung und -verifizierung von AD-Stacks durch das Konsortium gegeben.

1.3. Planung und Ablauf des Vorhabens

Das Vorhaben wurde in einer Mischstruktur aus 11 Arbeitspaketen und 5 Use-Cases durchgeführt.

Jeder Konsortialpartner hatte die Leitung über ein Arbeitspaket (hier: AP2 - Definition Gesamtzustandsraum (Leitung IQZ) und war zusätzlich in Use-Cases beteiligt, in denen die direkten Anwendungen der in den Arbeitspaketen entwickelten Projektergebnisse gemeinsam mit anderen Konsortialpartnern umgesetzt wurden.

Die Umsetzung der Use-Cases begann, nach dem Ausscheiden des ursprünglichen Konsortialführers Horiba, durch die Übernahme der Konsortialführung durch dSPACE. Nachfolgend wird nur die letztendliche Use-Case-Struktur erwähnt:

Abbildung 1: Umstrukturierung in Use Case 1-5

Das IQZ war an zwei Use-Cases maßgeblich beteiligt:

<u>Use-Case 1: Perzeptionsalgorithmus</u>
In UC1 wurde ein „Data Replay Testing"-Verfahren von Perzeptionsalgorithmen entwickelt, mit dem LiDAR-Daten mit unterschiedlichen Manipulationsmechanismen verändert und in einem Data Replay verwendet werden können. Diese Manipulation dient insbesondere dazu, die Robustheit der Perzeptionsalgorithmen zu überprüfen und mögliche Schwachstellen bzw. Verbesserungspotential in diesen zu identifizieren.

<u>Use-Case 2: Ausfallraten</u>
In UC2 wurden die Schnittstellen von „RT-CARS" (siehe Abschnitt 2.1) zu anderen Modellen weiterentwickelt. Dies geschah durch den Umbau des Tools von einer Matlab Implementierung mit JSON (Text) basierter Eingabe zur Systembeschreibung, zu einem Stand-Alone Programm inklusive einer grafischen Benutzeroberfläche.

1.4. Wissenschaftlich-technischer Stand an den angeknüpft wurde

Im Rahmen der Gesamtzustandsmodellierung von AD-Stacks gibt es einige Herausforderungen die zum Zeitpunkt des Projektstarts modellierungstechnisch nicht gelöst werden konnten.

AD-Stacks bestehen aus vernetzter Hard- und Software verschiedener Level. Als Hardware werden – im Kern – diverse Sensoren (Kameras, unterschiedliche Radare, Lidare) sowie hochperformante Rechenhardware benötigt. Diese muss in der Lage sein die komplexen Informationen der Sensorik im Rahmen der Objekterkennung, der Intention und Prädiktion der erkannten Objekte und der Trajektorieplanung in Echtzeit zu verarbeiten. Die reine Fahrfunktionen (Quer- und Längsführung) ist eine Low-Level-Aufgabe und wird nicht dem AD-Stack zugeordnet.

Durch die hochgerade und nicht-lineare Vernetzung (kein klassisches Input→ Prozess → Output System) innerhalb der Umsetzung der automatisierten oder autonomen Fahraufgabe, ist eine Zuverlässigkeits-, Verfügbarkeits- und Sicherheitsanalyse mit klassischen Methoden der Zuverlässigkeitstechnik nicht möglich.
Die modellierungsseitige Berücksichtigung komplexer funktionaler Abhängigkeiten der Software- und Hardware-Verknüpfungen ist mit bekannten Methoden auf Gesamtsystemlevel nicht möglich.
Aus diesem Grund wurde ein neues Modellierungslevel auf Basis eines Gesamtsystemzustands entwickelt, welcher sowohl komplexe soft- und hardwareseitige Abhängigkeiten als auch redundante funktionale Pfade berücksichtigen kann.

1.5. Zusammenarbeit mit anderen Stellen

Auf Konsortialebene gab eine Zusammenarbeit mit dem Nemobil-Projekt, welches teilweise überlappend zu RDV stattfand. Zudem gab es über die Mitgliedschaft einiger Konsortialpartner in der ASAM (Association for Standardization of Automation and Measuring Systems) eine weitere Zusammenarbeit im Bereich der Standardisierung.

Wissenschaftliche Kooperationen mit weiteren – nicht am Projekt beteiligten – Organisationen gab es im Rahmen des Teilvorhabens nicht.

II. Eingehende Darstellung

2.1. Verwendung der Zuwendung und des erzielten Ergebnisses im Einzelnen

RT-CARS

Im Rahmen von AP2 hat das IQZ eine Softwareapplikation namens „RT-CARS“ zur Sicherheits- und Zuverlässigkeitsevaluation kritischer Systeme entwickelt. Als Basis hierfür diente ein in Matlab implementiertes Simulationswerkzeug auf Basis der Monte-Carlo Methode. Diese rudimentäre Vorentwicklung beinhaltete bereits eine ausgereifte Modellbildungsebene sowie diverse Logikkonzepte in der Auswertungsebene für kritischen vernetzte Systeme, wie dem autonomen Fahrzeug. Dennoch reichte sie als Gesamtimplementierung nicht über ein prototypisches Level hinaus. So mussten Systembeschreibung und Evaluationsparameter über mehrere JSON Dateien bereitgestellt werden, welche dann nach dem Einlesen in interne Objekte umgewandelt und nach den gegebenen Konditionen simuliert worden sind. Dies erlaubte bereits eine detailreiche Modellierung doch wenig Übersicht und Bezug zum Modell und die Fehleranfälligkeit war hoch. Die Simulations- und Auswertungslogik waren ohne Effizienzsteigernde Maßnahmen (bspw. parallel-computing) implementiert. Aus diesen Gründen sollte das Simulationstool in zwei Etappen neugedacht werden; (1) eine benutzerfreundliche und zugleich fehlerregulierende Eingabemaske zur Systemmodellierung sollte entwickelt werden (2) die Eingabemaske und die Auswertungslogik sollten zu **einem** Konstrukt entwickelt werden (Stand-Alone Programm oder automatische Anbindung an Matlab backend).
Für die Erreichung des ersten Ziels ist eine grafische Benutzeroberfläche unter Verwendung der Programmiersprache C++ und dem Qt[1] Framework entwickelt worden. Mit dieser konnten alle Komponenten und Eigenschaften des JSON in einer grafischen Oberfläche modelliert werden. An geeigneten Stellen sind Falscheingaben (bspw. inkorrekte Symbole in Bezeichnungen, Fehlerraten etc.) abgefangen worden. Ein JSON Konverter hat dann das grafische System in JSON Dateien kompatibel mit dem Matlab-Backend überführt. Damit konnte ebenfalls die Integrität und Korrektheit der JSON Dateien sichergestellt werden. Für das zweite Ziel wurden zunächst IQZ-intern die Vor- und Nachteile einer automatischen Anbindung von der GUI zum Matlab-Backend und der Neuentwicklung des Backend in C++ diskutiert. Letztendlich ist sich aus mannigfaltigen Gründen für eine Neuentwicklung entschieden worden, wie u.a.:

- Steigerung von Performanz und Effizienz,
- bessere Wartbarkeit und Auslegung der neuen Codebasis dahingehend,
- Nutzen von diversen Tools zur Sicherstellung der Programmkorrektheit (code formatters, unittests etc.),
- Erstellen einer nachhaltigen Alleinlösung.

Somit entstand die im nachfolgenden beschriebene Alleinlösung „RT-CARS“.

[1] https://www.qt.io/ abgerufen am 11.11.2025

Beschreibung

Die in C++/Qt-geschriebene Applikation „RT-CARS“ (siehe Abbildung 1) ermöglicht es nun dem Benutzer ein System auf Komponentenebene zu modellieren. Dafür lassen sich komfortabel via Drag-and-drop Hardware Komponenten definieren. Es wird zwischen drei Basistypen unterschieden:

1. Sensoren,
   Ein geschlossenes Hardware-Firmware-System, welches autonom Messdaten erfasst, vorverarbeitet und ausgibt, jedoch keine Eingaben entgegennimmt[2] (bspw. RADAR- oder Luftdruck-Sensor).

2. Chips,
   Hardware Chips die eine Funktionalität bereitstellen können (Signal erzeugen) und weiterhin 0-n ($n \in \mathbb{N}$) Computing Units bereitstellen. Computing Units sind virtuelle Prozesseinheiten welche Rechenressourcen für Software Applikationen bereitstellen.

3. Switches.
   Switches leiten Signale zwischen den anderen Komponenten weiter und senden selber kein Signal, da sie keine eigene Funktionalität bereitstellen.

Alle Komponenten können mit vordefinierten Fehlermodi versehen werden. Die von einer Komponente produzierten Ausgaben werden in RT-CARS klassisch als Signale abstrahiert. Software Applikationen werden separat definiert, da sie als vielfache Applikationen auf den Computing Nodes von Chips definiert werden können, aber keine eigene feste Darstellung besitzen. Software Applikationen zeichnen sich dadurch aus, dass sie Signale konsumieren (benötigen) ein Signal senden und Ressourcen (eines Computing Nodes) verbrauchen. Software Applikationen werden ebenfalls mit Fehlermodi versehen. Datenverbindungen zwischen den Komponenten werden mittels Konnektoren (Connector) realisiert. Diese haben bisher keine weitere Semantik, außer dass Signale weitergeleitet werden. So werden sie mit dem Einsatz von Switches dazu verwendet verschiedene Kommunikationssysteme so wie Busse (z.B. CAN Bus) zu modellieren.

[2]Unabhängig von möglichen Parametrisierungen sind Eingabesignale grundsätzlich nicht notwendig und es werden keine Daten zur Weiterverarbeitung entgegengenommen.

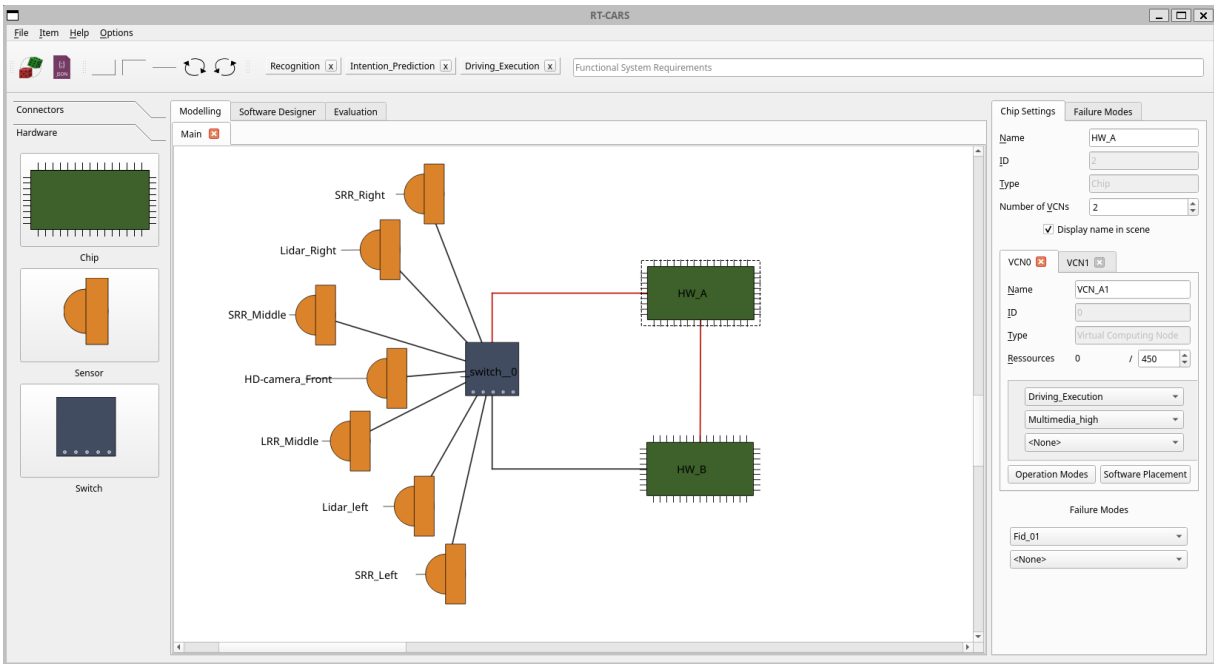


Abbildung 2: GUI (RT-CARS) Hauptfenster mit modelliertem Beispiel-System

Die Fehlermodellierung ist über ein Eingabefenster geregelt bei dem der Nutzer zunächst die Möglichkeit hat, Fehlermodi global zu definieren. Dabei werden unterschiedliche Ausfallmodelle unterstützt: Weibull-verteilte, Exponentialverteilte oder Normalverteile Ausfallraten sowie Ausfälle die zu einer fixen Rate auftreten. Zur Widerverwendung werden diese in einer lokalen Datei abgespeichert. In erster Linie sind hier klassische (FuSi) Fehler adressiert und es gibt die Möglichkeit pro Fehler eine Reparaturrate zu definieren (hier gibt es wieder die gleichen Verteilungsoptionen, die unabhängig vom Fehler sind). Über das Einstellungsfenster der Hard- und Softwarekomponenten können die angelegten Fehlermodi dann für eine Komponente gesetzt werden. Da pro Komponente beliebig viele Fehlermodi definiert werden können, und die Reparaturen unabhängig voneinander sind, besteht hier jedoch auch eine erste Möglichkeit auch transiente Fehler, wie temporäre Sensorausfälle durch Verschmutzung etc. abzubilden (SOTIF). Die Modellierung von Redundanzen erfolgt über die Definition von Redundanzgruppen. Dabei sind Software und Hardware getrennt worden, da davon ausgegangen worden ist, dass Hardware und Software Komponenten keine äquivalenten Aufgaben erfüllen können (Software benötigt immer Hardware auf der sie ausgeführt werden kann). So kann dann einer Software als notwendige Eingabe auch eine redundante Signalgruppe anstatt eines Signals gegeben werden. Modellierte Systeme lassen sich im XML-Format Speichern und Laden, sodass der Austausch des Modells und dessen Erweiterung einfach möglich ist.
Für die Auswertungslogik muss der Benutzer festlegen, welche Signale oder Signalgruppen im System mindestens vorhanden sein müssen – und damit welche Komponenten funktionsfähig sein müssen damit ein sicherer Systemzustand gegeben ist. Letztlich wird aus dieser Information das Zuverlässigkeitsziel für die nachgesetzte Simulation abgeleitet. Für die Systemauswertung wird eine Simulation nach der Monte-Carlo Methode ausgeführt.
Die Anzahl der Iterationen, avisierte Lebenszeit sowie Toleranzzeiten für diverse FDIR(O) Schritte sind vom sind vom Nutzer individuell zu bestimmen. Die Simulation läuft dabei wie folgt ab: RT-CARS überführt modellierte Komponenten in eine Simulationsdarstellung und erzeugt wesentliche interne Systemzustände indirekt (Logik zur Überprüfung des Endzustandes/totaler Systemausfall). Es wird ereignisbasiert durch die Iteration gelaufen. Zunächst wird mithilfe der hinterlegten Fehlermodi und

eines Zufallsgenerators eine Liste an Ausfällen erzeugt, die dann bei tatsächlich eingetretenem Ausfall (active->failed) um mögliche Reparaturzeitpunkte erweitert wird. Die Reparatur von Hardware Komponenten ist simpel gehandhabt (siehe Abbildung 2). Diese werden einfach zu dem kalkulierten Zeitpunkt wieder repariert (failed->active).

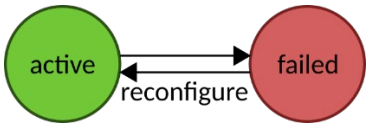


Abbildung 3: Hardware Zustände

Für Software wird ein komplexerer Prozess angestoßen. Hierfür ist das interne Zustandsmodell, wie in Abbildung 3 dargestellt, hinterlegt.

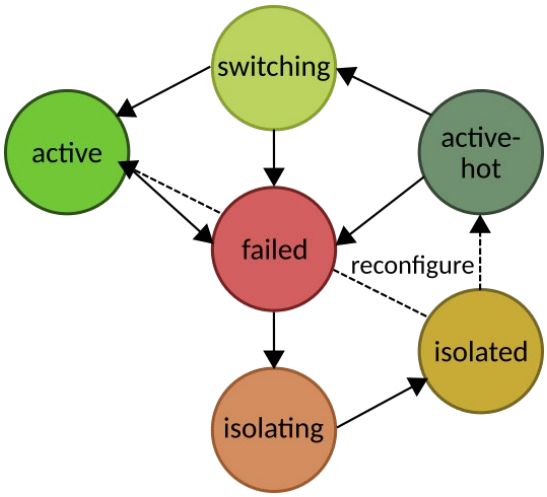


Abbildung 4: Software Zustände durch FDIR-Prozess

Der active Zustand beschreibt dabei den Zustand in dem die Applikation funktionsfähig ist und die Ausgabe der Applikation genutzt wird. In dem Zustand active-hot ist die Applikation genauso aktiv, jedoch wird ihre Ausgabe nicht genutzt. Redundante Applikationen laufen in diesem Zustand mit. Fällt eine Applikation aus, dauert es eine gewisse Zeit bis der Fehler erkannt und die Applikation isoliert worden ist. Genauso ist der Übergang von active-hot nach active (Switching) nicht unbedingt sofort (instant) möglich. Minimal und maximal Zeiten für diese Prozesse lassen sich in den Simulationseinstellungen bestimmen.

Beim Ausfall einer sicherheitskritischen Software Applikation, wird überprüft, ob eine redundante Instanz auf diesem oder einem anderen Computing Node im active-hot Zustand verfügbar ist. Ist das der Fall, wird für diese der Switching-Prozess gestartet. Für die ausgefallene Instanz wiederrum wird ein Reperaturprozess angestoßen, um nach Möglichkeit das gleiche Fehlertoleranz-Level wiederherzustellen.

Für den Fall, dass ein Chip Item mit sicherheitskritischer Software auf seinen Computing Nodes ausfällt, wird versucht redundante Instanzen (auf anderen Chips) zu aktivieren und falls dies nicht ausreicht, die benötigten Applikationen auf anderen Computing Nodes mit ausreichend Ressourcen neuzustarten. Da die Zustandswechsel gewisse Zeiten beanspruchen können, wird für den gesamten FDIR(O) Prozess eine Toleranzzeit festgelegt (Nutzereingabe). Ist ein System nicht in der Lage innerhalb dieser Toleranzzeit (analog Fault Tolerant Time Interval (FTTI) aus der ISO 26262) einen sicheren Zustand wiederherzustellen, so gilt es als komplett ausgefallen und die Iteration wird im Fehlerzustand beendet. Eine Iteration ohne Komplettausfall endet nach Erreichen der avisierten Lebenszeit. Nach

Beendigung der Simulation stehen dem Benutzer verschiedene Ergebnisauswertungen bereit. Je nach Einstellungen (Logging-Level) wird pro Iteration ein detailliertes Ereignislogs bereitgestellt. Auf Basis dieser Logs sind verschiedene Analysen zu Zuverlässigkeits-KPI bereits implementiert: Ausfallwahrscheinlichkeit, Zuverlässigkeit, Verfügbarkeit. Außerdem bietet RT-CARS die Möglichkeit zur Parameterabschätzung für verschiedene Verteilungsfunktionen sowie Detail-Auswertungen wie die Anzahl an durchgeführten Software Verschiebungen und Widerherstellungen.

Limitierungen

Der Fokus der Modellierung in RT-CARS liegt in der Sicherheit und Zuverlässigkeit des Fahrzeugs. Es wurden bereits erste Überlegungen und Tests unternommen auch Umgebungsfaktoren in die Fehlermodellierung einfließen zu lassen (Stichwort, SOTIF). So können transiente Fehler wie vorrübergehende Signalstörungen (z.B. Interferenzen) oder Bildstörungen (z.B. Verschmutzungen) als „kurzlebige Fehler“ (mit hohen Reparaturraten) beschrieben werden. Für genaue Analysen müsst es jedoch eine grundsätzliche Kapselung der Fehlerarten geben (bspw. kann in der Realität ein verdreckter Sensor auch noch funktional ausfallen). Weiterhin ist bis dato ist RT-CARS nicht explizit in der Lage Safety-Risiken durch exploits von Vulnerabilitäten abzubilden (Cybersecurity). Wenn auch in der Theorie natürlich eine „Ausfall“-rate für dieses Risiko angelegt werden kann, fehlt im aktuellen Zustandsraum eine dedizierte Unterscheidung zwischen den Konsequenzen eines Cyberangriffs (bspw. wird eine Komponente zunächst „übernommen“ funktioniert dennoch weiter wie erwartet) und denen eines Ausfalls.
Um diese jedoch in der Gesamtsystemmodellierung in AP2 nicht untergehen zu lassen, wurde die Integration in ein Markov-basierten Model-checking Ansatz (PRISM) erfolgreich geprüft. Eine Automatisierung von RT-CARS ausgehend, ist jedoch nicht weiterverfolgt worden.

Anwendung in UC2

Im Use Case 2 wurde RT-CARS in der gegenseitigen Verifikation der Modelle mit dem EMI Markov-Modell angewendet. Hierbei sind wichtige Erkenntnisse gewonnen worden, die die Weiterentwicklung von RT-CARS vorangetrieben haben. Abbildung 4 zeigt den Aufbau vom Use Case.

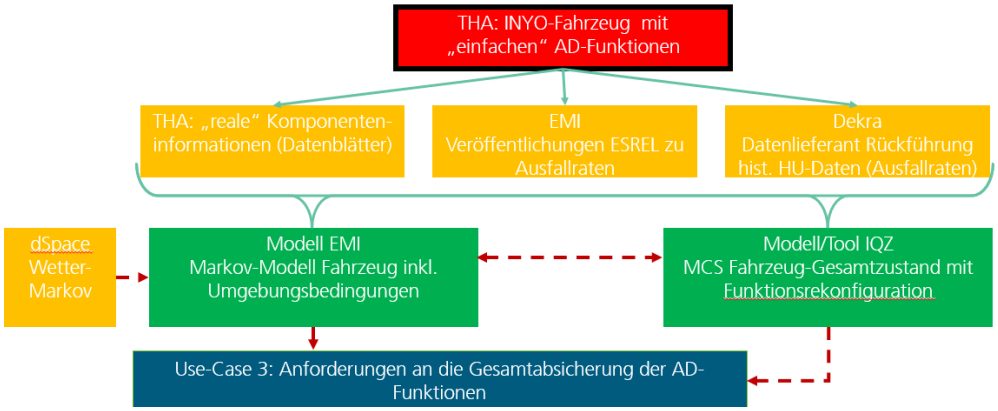


Abbildung 5: Aufbau des Use Case 2

Bei der Modellverifikation sollten zunächst beide Modelle (RT-CARS und EMI Markov-Modell), die *gleiche* Eingabearchitektur eines autonomen bzw. automatisierten Fahrzeuges modellieren und auswerten. Das Ziel war es zu überprüfen, ob beide Modelle grundsätzlich ähnliche Ergebnisse liefern. Auf Grund ihrer offensichtlichen semantischen Unterschiede sind Abweichungen der Ergebnisse

allgemein zu erwarten, jedoch sollten grundlegende Sicherheits-KPIs wie die Zuverlässigkeit oder die Verfügbarkeit des Systems gleich oder sehr ähnlich ausfallen.
Dafür wurde sich zunächst auf die Teilarchitektur eines autonomen bzw. automatisierten Fahrzeuges abstrahiert auf den Sensor-HUB mit Anbindung an einen Fahrzeug Computer geeinigt. Der Sensor-HUB besteht aus diversen Perzeptionssensoren (5 Kamerasensoren und 4 LIDAR Sensoren). Die nachstehende Abbildung 5 zeigt dieses Modell in RT-CARS.

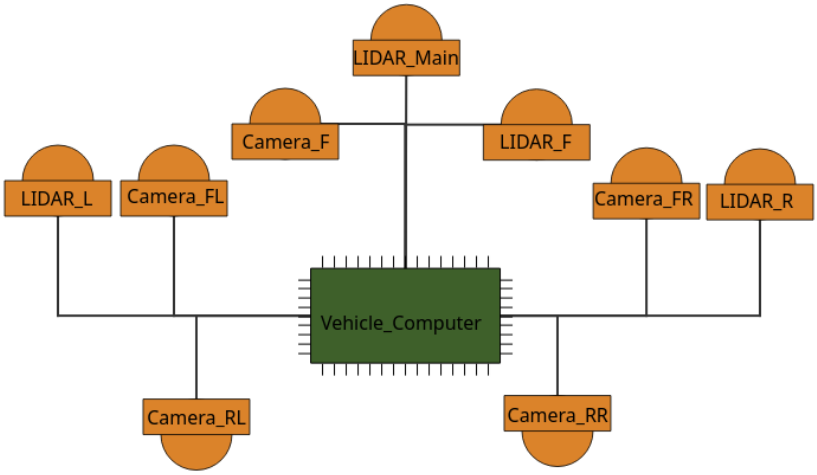


Abbildung 6: INYO Setup in RT-CARS

Durch unterschiedliche Positionen und Ausrichtungen im Fahrzeug entstehen unterschiedliche Redundanzdefinitionen der Sensoren (siehe Abbildung 6). So ist beispielsweise das Haupt-LIDAR in jeglicher Situation unverzichtbar, während das Front-Lidar rechts als redundant zu erachtende Daten zur Front-Kamera rechts liefert. Folglich kann das System mit einzelnen Sensorfehlern umgehen und bleibt, je nach dem welcher Sensor ausgefallen ist, funktionsfähig. Für jede modellierte Komponente ist eine Fehlerrate (Ausfallwahrscheinlichkeit) festgelegt worden.

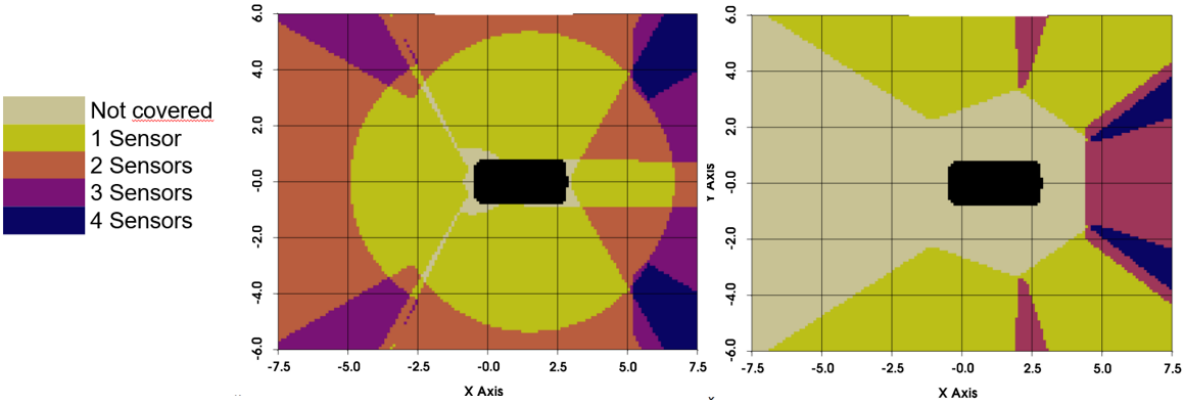


Abbildung 7: Sensor-Coverage INYO Setup. Links Lidar, Rechts Kamera.

In einem ersten Vergleichsdurchlauf konnte gezeigt werden, dass beide Modelle (RT-CARS mit Monte-Carlo Methode und Markov Modell) sehr ähnliche Daten liefern – damit wird dieser erste Vergleich als erfolgreich angesehen. Im nächsten Schritt sollte das modellierte System um die Reparatur von Sensoren erweitert werden. So könnte beispielsweise dargestellt werden, dass ein Sensorfehler nur temporär anhält. Dies ist zwar aus Sicht der funktionalen Sicherheit ohne einen Werkstattaufenthalt nicht sinnvoll anzunehmen, könnte aber ggf. genutzt werden, um transiente Fehler, wie die temporäre Blockade durch Verschmutzung, widerzuspiegeln. Da RT-CARS zunächst Komponentenreparaturen

ausschließlich mit dem auf Software spezifizierten FDIRO Prozess darstellen konnte, sind für einen weiterführenden Vergleich der Modelle Hardware-Reparaturen implementiert worden (wie oben beschrieben). Das damit um Reparaturmöglichkeiten erweiterte Beispielmodell wurde ausgewertet und mit den Ergebnissen des Markov Modells verglichen.

Hier sind zunächst Unstimmigkeiten aufgefallen, die nach eingehender Analyse auf die Qualitätsannahmen einer reparierten Komponente zurückzuführen waren: In RT-CARS wird eine Hardware-Reparatur wie eine totale Reparatur bspw. durch einen Komponentenaustausch modelliert (Annahme aus der funktionalen Sicherheit), während in der Markov Modellierung die Reparatur als Widerherstellung der Komponente implementiert ist. Folglich setzt sich in RT-CARS die Lebenszeit der Komponente, und damit auch die Wahrscheinlichkeit für nachfolgende Ausfälle, auf den Ausgangszustand zurück während in der Markov Modellierung die Lebenszeit vom Ausfallzeitpunkt wieder aufgenommen wird. Abbildung 7 zeigt die Zuverlässigkeit (Reliability) von dem RT-CARS Modell. Nach 10.000 Stunden liegt diese bei c.a. 38% .

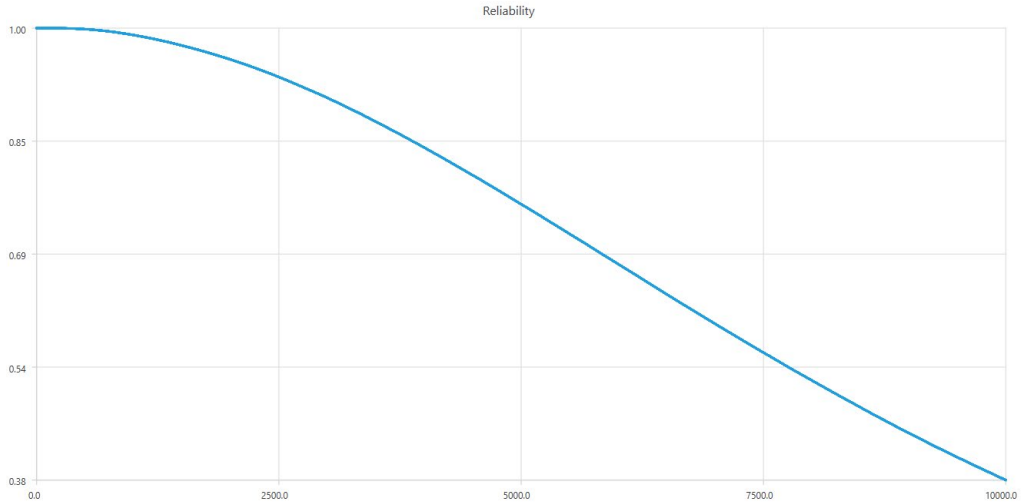


Abbildung 8:RT-CARS Reliability mit Hardware (Sensor) Reparaturen

In Abbildung 8 sind die Ergebnisse verschiedener Auswertungen des Markov Modells skizziert.

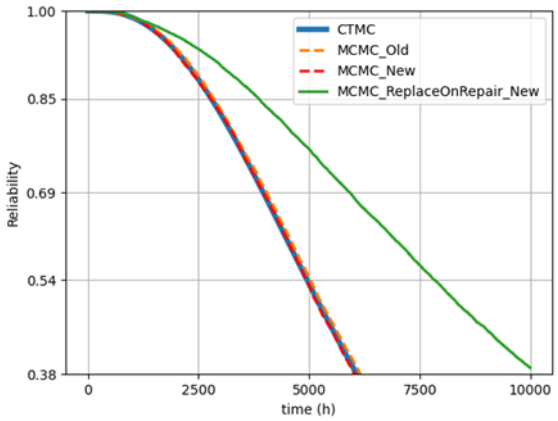


Abbildung 9: Auswertung Markov-Modell (EMI)

Hier ist erkennbar, dass die Zuverlässigkeit nach 10.000 in den ersten Ansätzen deutlich unter 38% liegt. Nach einer testweisen Anpassung der Auswertungslogik für die Reparatur („replace on repair“) konnten der gleiche Wert wie in der RT-CARS Auswertung von 38% erzielt werden. Damit konnte ein zuvor nicht bewusster struktureller Unterschied aufgedeckt werden, seine Auswirkung gezeigt und am

Ende auch die beiden Modelle mit der gegebenen Architektur erfolgreich gegeneinander geprüft werden.

Use Case 1 - Data Replay Testing von Perzeptionsalgorithmen

Das Entwickeln von automatisierten Fahrfunktionen erfordert einen hohen Umfang an Testaufwand. Dieser Testaufwand kann, insbesondere wenn es sich um das Testen mit einem Fahrzeug in realer Umgebung handelt, sehr zeitaufwendig und kostenintensiv sein. Aus diesem Grund wurde im Rahmen des Use Case 1 durch die Projektpartner dSpace, IQZ, JHP und LiangDao ein Testaufbau zur Überprüfung eines Perzeptionsalgorithmus erarbeitet, welcher unter der Verwendung von bereits aufgenommenen Sensordaten unterschiedliche Perzeptionsalgorithmen evaluieren kann. Die Aufteilung der jeweiligen Arbeitsschwerpunkte der Partner im Use Case 1 ist in Abbildung 1 dargestellt.

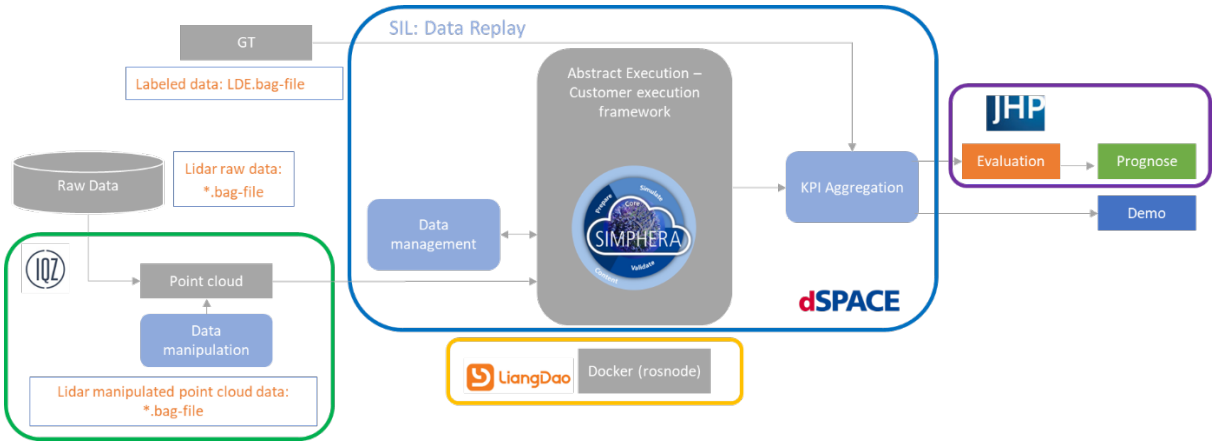


Abbildung 10: Arbeitsschwerpunkte des Use Case 1

Die Sensordaten werden von LiangDao zur Verfügung gestellt. Es handelt sich hierbei um Daten aus einem sogenannten LiDAR Sensors, welcher mit Hilfe von Laserstrahlen den Abstand zu Objekten bestimmen kann. Diese Daten wurden in im Vorhinein durchgeführten Testfahrten aufgenommen und im Anschluss wurden die detektierten Objekte gelabelt, so dass eine Ground Truth vorhanden ist, gegen welche der Perzeptionsalgorithmus im Nachhinein abgeglichen werden kann. Die Sensordaten werden in das Software-in-the-Loop Data Replay von dSpace eingespielt und dort unter Verwendung eines Perzeptionsalgorithmus weiterverarbeitet. Das Ergebnis des Algorithmus wird im Anschluss an JHP übergeben, welche diese Daten unter Verwendung der Ground Truth evaluieren und den Perzeptionsalgorithmus auf diese Weise bewerten.
Die Sensordaten, welche von LiangDao aufgenommen und zur Verfügung gestellt werden, bilden einen idealen Datensatz ab, welcher unter idealen Bedingungen aufgenommen wurde. Da dies in der Realität nicht gegeben sein muss, wurde durch das IQZ ein zusätzliches Modul entwickelt, welches die aufgenommenen Sensordaten mittels unterschiedlicher Algorithmen manipuliert und anschließend an dSpace weitergibt. Auf diese Weise kann der Perzeptionsalgorithmus auf seine Robustheit bzgl. nicht idealer Sensordaten überprüft werden. Die Wirkweise der Manipulation wird im folgenden Kapitel erläutert.

Sensordatenmanipulation

Motivation

Die von LiangDao bereitgestellten Datensätze bilden, wie im vorherigen Kapitel beschrieben, den idealen Zustand ab. Bei der späteren Verwendung von Sensordaten im realen Fahrzeug kann es jedoch aufgrund unterschiedlicher Faktoren dazu kommen, dass die Sensordaten verfälscht werden. Diese Faktoren können intrinsisch sein, z.B. durch Interferenzen oder Übertragungsfehler, oder durch externe Gegebenheiten hervorgerufen werden. Hier sei u.a. die verschlechterte Wahrnehmung der Umgebung aufgrund von Wetterbedingungen oder durch die Verschmutzung des Sensors zu nennen. Damit diese Aspekte auch schon bei der Evaluierung des Perzeptionsalgorithmus mitberücksichtigt werden können, wurde ein Manipulationsalgorithmus erarbeitet, welcher die originalen Daten einließt, diese, in Abhängigkeit zu den definierten Eingabegrößen, anpasst, und im Nachhinein wieder im ursprünglichen Format dem Software-in-the-Loop Modul bereitstellt.

Manipulationsalgorithmen

Die Bereitstellung der Sensordaten durch LiangDao erfolgt mittels dem Dateiformat rosbag[3]. Diese hat den Vorteil, dass unterschiedliche Sensordaten gesammelt können und nicht jedes Sensorsignal einzeln übergeben werden. Die im vorliegenden Projekt verwendete Dateistruktur setzt sich dabei aus den Daten zusammen:

- 'ld_msgs/ld_image'
- 'sensor_msgs/PointCloud2'
- 'ld_msgs/ld_imugps_can'
- 'ld_msgs/ld_object_list'.

Der für die Manipulation relevante Datensatz mit den Sensordaten ist dabei ‚sensor_msgs/PointCloud2'. Dieser Datensatz beinhaltet die x-, y- und z-Koordinaten von Punkten, welche mit den Laserstrahlen des LiDAR-Sensors detektiert wurden. Aufgrund der Vielzahl von Punkten werden diese Datensätze auch Punktwolken genannt. Das rosbag bildet dabei immer eine bestimmte Aufnahmedauer ab. Über diese Aufnahmedauer hinweg existieren, in Abhängigkeit zur Frequenz des Sensors, diverse Punktwolken, welche jeweils die Detektionen zu einem bestimmten Zeitpunkt abbilden.

Die Manipulation der Daten erfolgt mit der Software MathWorks Matlab. Wichtig für die Weiterverarbeitung im Data Replay ist es, dass die rosbag Datei im Nachhinein die gleiche Struktur hinsichtlich der integrierten Datensätze aufweist, d.h., es ist nicht ausreichend, nur PointCloud2 zu extrahieren, manipulieren und in ein rosbag zurückzuschreiben, sondern die Kombination aller vier Datensätze muss am Ende wieder umgesetzt werden.

Der Algorithmus zur Manipulation der Daten ist so aufgebaut, dass alle relevanten Eingaben nach dem Start abgefragt werden.

Die erste Instanz bildet dabei die Auswahl der zu manipulierenden rosbag Datei. Diese wird über ein Eingabefenster realisiert, welches sich automatisiert nach dem Start öffnet (Abbildung 2).

[3] https://wiki.ros.org/rosbag

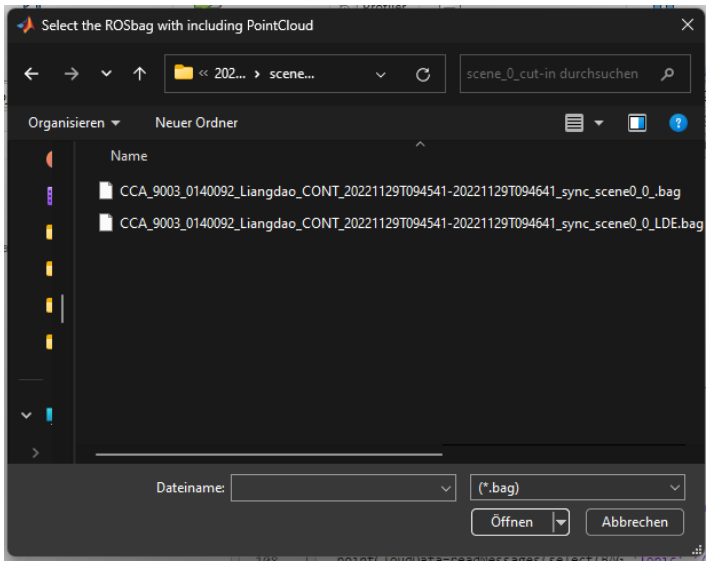


Abbildung 11: Eingabefenster zur Auswahl der zu verändernden rosbag Datei

Nach dem Einlesen der Datei und dem Extrahieren der PointCloud2 Daten wird der User befragt, auf welche Art die Daten manipuliert werden sollen (Abbildung 3). Insgesamt wurden hierfür die folgenden vier Mechanismen implementiert, welche in den folgenden Abschnitten noch genauer erläutert werden:

- Änderung von Punkten
- Verlust von Punkten
- Rauschen
- Sensor Kontamination

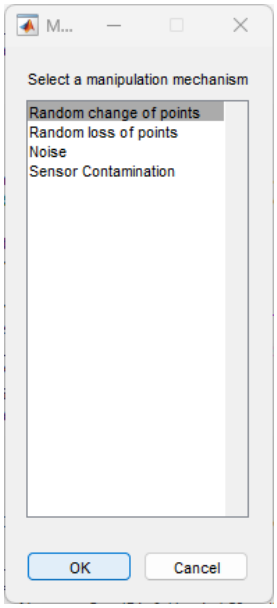


Abbildung 12: Auswahl der Manipulationsmechanismen

Abschließend muss der User die Ausprägung der Manipulation definieren. Dies kann im Falle des Datenverlusts und der Änderung von Punkten beispielsweise über die Definition einer Wahrscheinlichkeit angegeben werden (Abbildung 4).

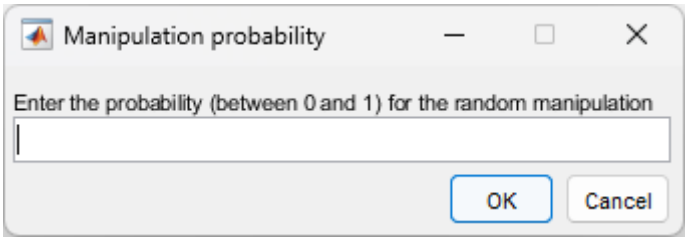


Abbildung 13: Eingabemaske zur Definition der Ausprägung der Manipulation

Nachdem alle notwendigen Eingaben erfolgt sind, erfolgt die eigentliche Datenmanipulation. Hierfür werden alle Datensätze für alle Zeitpunkte, welche im Datensatz PointCloud2 enthalten sind, nacheinander manipuliert. Abschließend wird der manipulierte Datensatz PointCloud2 mit den drei weiteren Datensätzen des ursprünglichen rosbags wieder zusammengesetzt und in einem neuen rosbag abgespeichert, welches anschließend an dSpace zur Verwendung im Data Replay übergeben wird.
In den folgenden Unterabschnitten werden die einzelnen Manipulationsmechanismen nun genauer beschrieben.

Änderung von Punkten

Bei der Änderung von Punkten werden die Koordinaten von Punkten aus der Punktwolke zufällig verändert. Die Entscheidung, ob die jeweiligen Punkte verändert werden, wird durch den Abgleich einer gleichverteilten Zufallszahl zwischen 0 und 1 und der, durch den User definierten, Wahrscheinlichkeit realisiert. Die Logik hierbei ist die folgende:

- Zufallszahl ≤ Wahrscheinlichkeit → Punkt wird geändert
- Zufallszahl > Wahrscheinlichkeit → Punkt bleibt unverändert.

Dieser Abgleich wird für jeden einzelnen Punkt in der Punktwolke durchgeführt, d.h., es werden für jedes Bild genauso viele Zufallszahlen erzeugt, wie es Bildpunkte gibt. Aus der Abhängigkeit ist ersichtlich, dass bei einer geringen Wahrscheinlichkeit weniger Punkte manipuliert werden als bei einer hohen Wahrscheinlichkeit.
Die Veränderung der detektierten Punkte wird ebenfalls über gleichverteile Zufallszahlen realisiert. Diese sind jedoch nicht zwischen 0 und 1 verteilt, sondern für die Grenzen wird sich hierbei an den Werten des aktuellen Sichtfelds orientiert, d.h. aus dem vorliegenden Datensatz werden alle Werte der x-, y- und z-Koordinate ermittelt. Da es teils Ausreißer in den Datensätzen gab, wurden nun nicht die minimalen und maximalen Werte als Grenzwerte verwendet, sondern jeweils das 5 %, bzw. 95 % Quantil der x-, y-, und z-Werte als Grenzen für die Zufallszahlen herangezogen.
In Abbildung 6 sind die Auswirkungen der Manipulation mit steigenden Wahrscheinlichkeiten zu sehen. Es ist erkennbar, wie mit steigender Wahrscheinlich die Strukturen der originalen Punktwolke verloren gehen und diese in ein gleichverteiltes Rauschen übergehen.

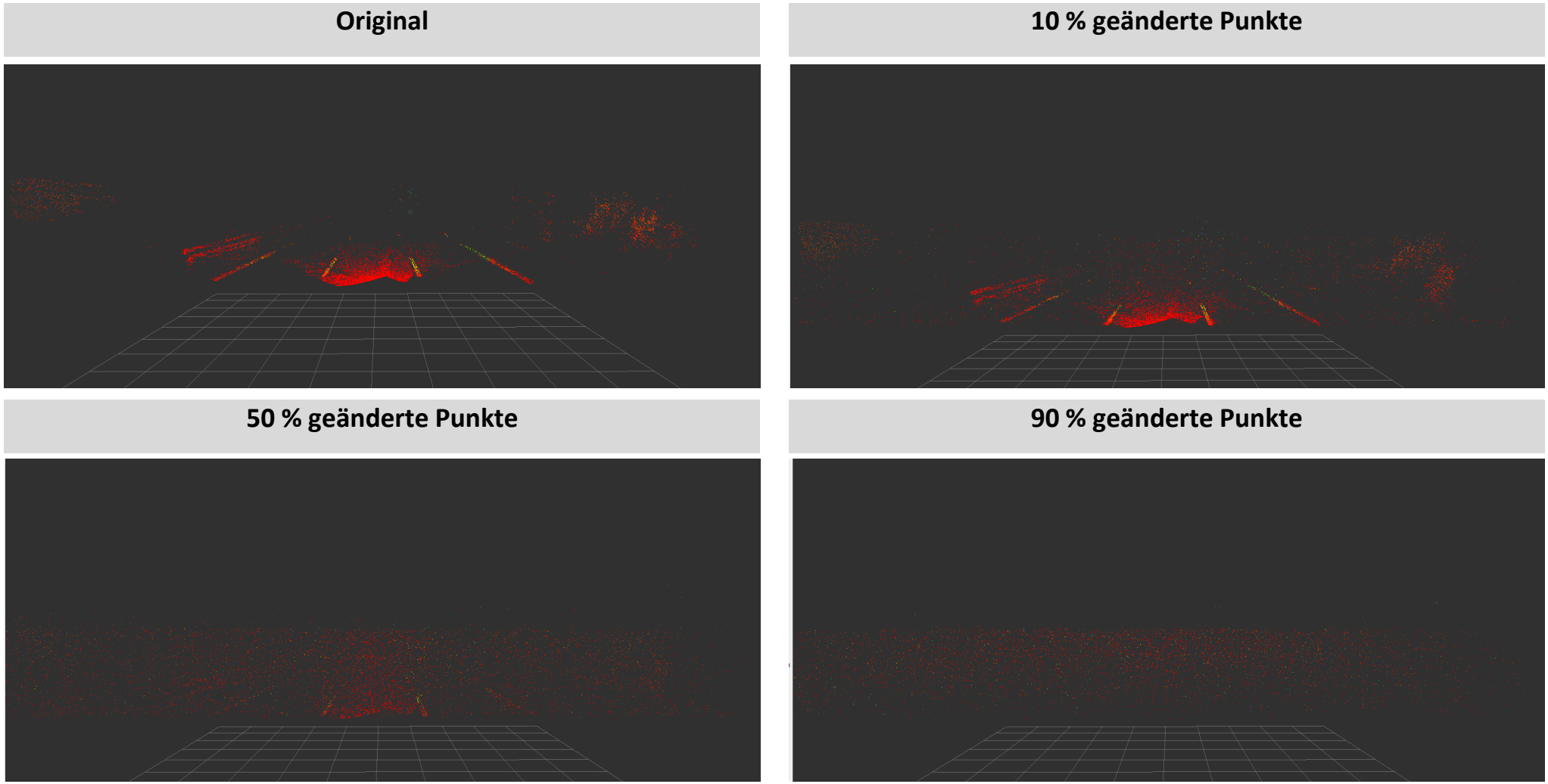


Abbildung 14: Visualisierung der Sensordatenmanipulation über die zufällige Änderung von Datenpunkten

Verlust von Punkten

Der Verlust von Punkten ist ähnlich aufgebaut wie die Änderung von Punkten. Auch hier wird ein Abgleich zwischen einer gleichverteilten Zufallszahl und der durch den User definierten Wahrscheinlichkeit durchgeführt. Die Folgen hierbei sind jedoch nicht die Änderung von Punkten, sondern das Löschen der jeweiligen Punkte. Die Logik hierbei ist die folgende:

- Zufallszahl ≤ Wahrscheinlichkeit → Punkt wird gelöscht
- Zufallszahl > Wahrscheinlichkeit → Punkt bleibt bestehen.

Mit steigender Wahrscheinlich nehmen somit die Punkte zu, welche verloren gehen.
Auch in diesem Fall wird für jeden existierenden Punkt in der Punktwolke eine Zufallszahl generiert und diese mit der definierten Wahrscheinlichkeit abgeglichen.

Die Auswirkung der Manipulation über steigende Wahrscheinlichkeiten ist in Abbildung 7 zu sehen. Wie in Abbildung 6 ist zu erkennen, wie die Strukturen der ursprünglichen Punktwolke verloren gehen, es kommen jedoch keine neuen Punkte hinzu, welche sich wie ein Rauschen über das Sichtfeld legen.

Rauschen

Die dritte Möglichkeit die Daten zu manipulieren ist, ein gleichverteiltes Rauschen über die Punktwolke zu legen. Hierbei bleiben die vorhandenen Punkte bestehen, es werden jedoch weitere Punkte generiert, welche innerhalb des Sichtfelds platziert werden.
Als Eingabegröße durch den Nutzer muss die Anzahl an Punkten definiert werden, mit welchen die Punktwolke überlagert werden.
Für die Generierung des Rauschens werden zufällige Werte für die x-, y- und z-Koordinate in definierten Bereichen erzeugt. Zur Bestimmung dieser Bereiche werden, wie bei der Änderung von Punkten, alle x-, y- und z-Werte des aktuellen Punktwolke betrachtet und das jeweilige 5 % und 95 % Quantil ermittelt, welches dann als Grenzwert für die Zufallszahlendefinition verwendet wird.
Die Auswirkung des Rauschens ist in Abbildung 8 zu sehen. Durch die Überlagerung der Punktwolke mit einem Rauschen gehen mit steigender Anzahl an generierten Punkten insbesondere die Strukturen der originalen Punktwolke verloren, bei denen die Dichte an detektierten Punkten nicht so hoch ist.

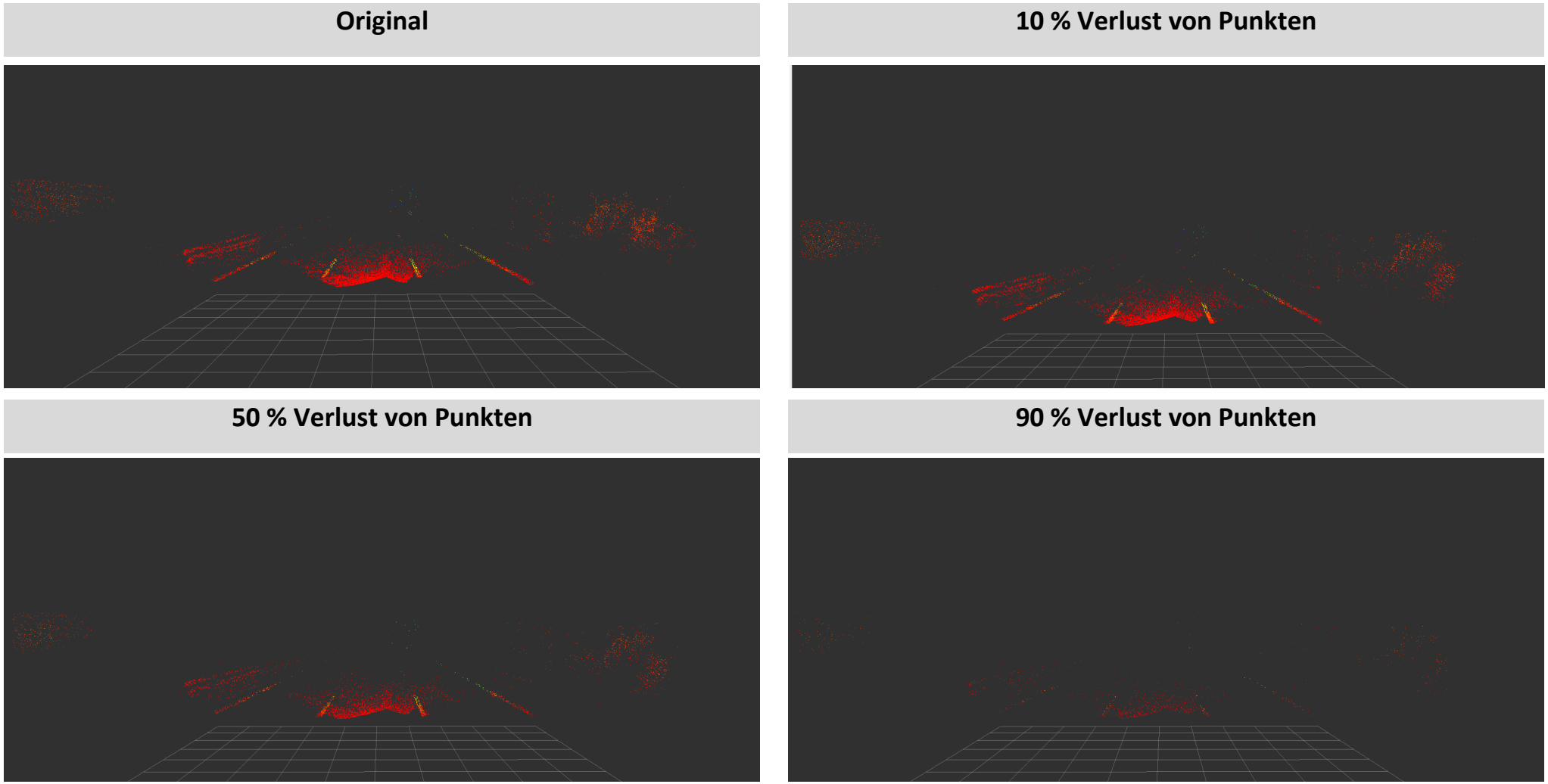
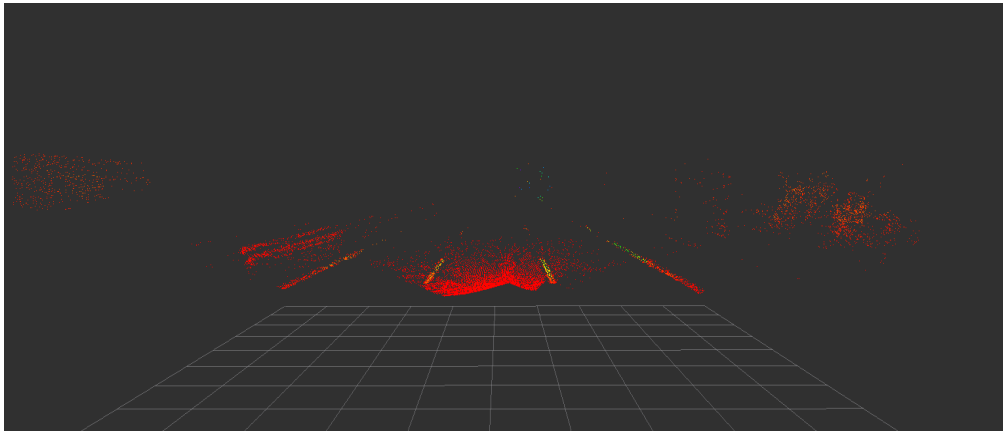
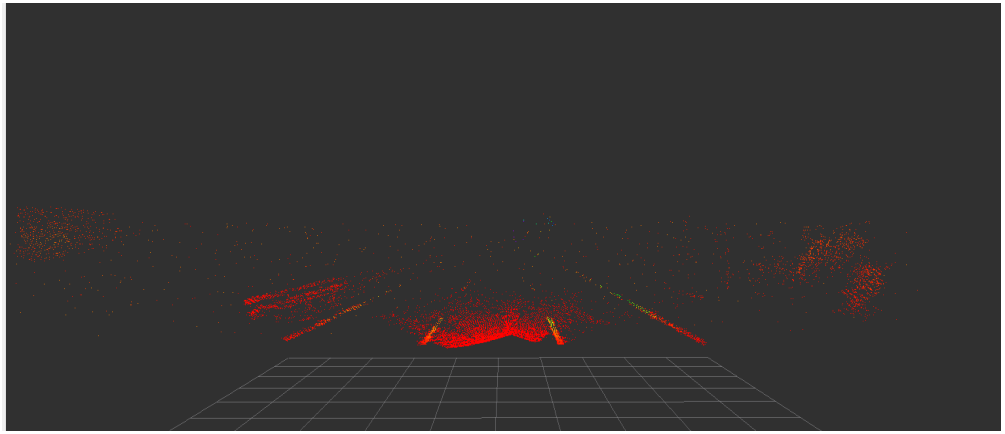


Abbildung 15: Visualisierung der Sensordatenmanipulation über den zufälligen Verlust von Datenpunkten

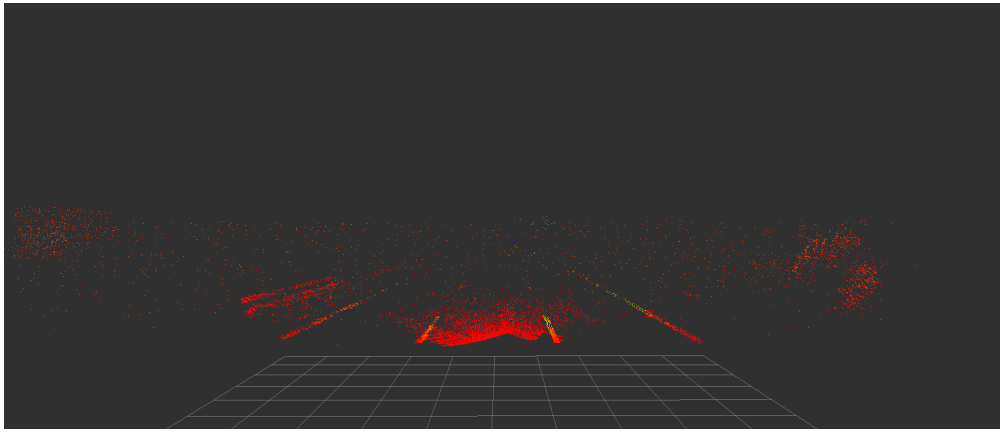
Original



5k Punkte Rauschen



10k Punkte Rauschen



20k Punkte Rauschen

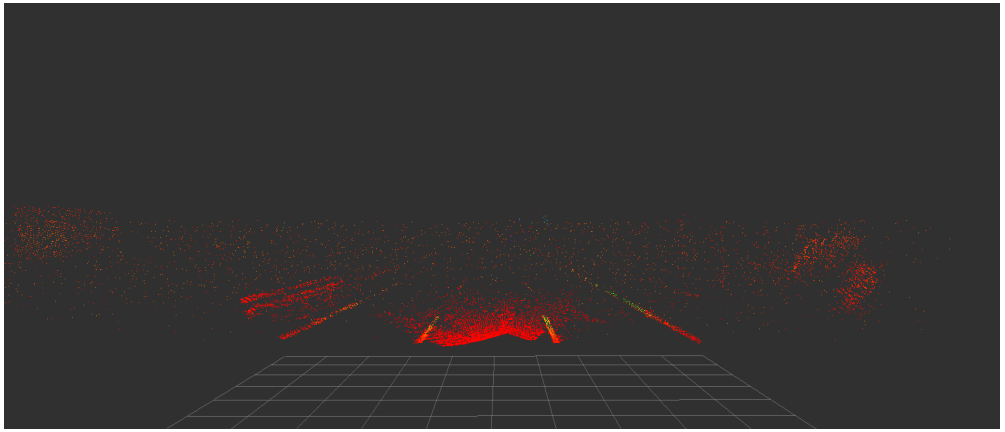


Abbildung 16: Visualisierung der Sensordatenmanipulation über das Hinzufügen eines gleichverteilten Rauschens

Sensor Kontamination

Der Manipulationsmechanismus Sensor Kontamination stellt die Verschmutzung des Sensors dar. Die Verschmutzung wird mit Hilfe des Perlin Rauschens imitiert. Das Perlin Rauschen (Abbildung 9) zeichnet sich dadurch aus, dass es kein gleichverteiltes Rauschen ist, sondern sich Strukturen ausbilden, die Wolken oder Wasser ähneln, wodurch sie wiederum geeignet dazu sind, die Verschmutzung eines Sensors nachzubilden.

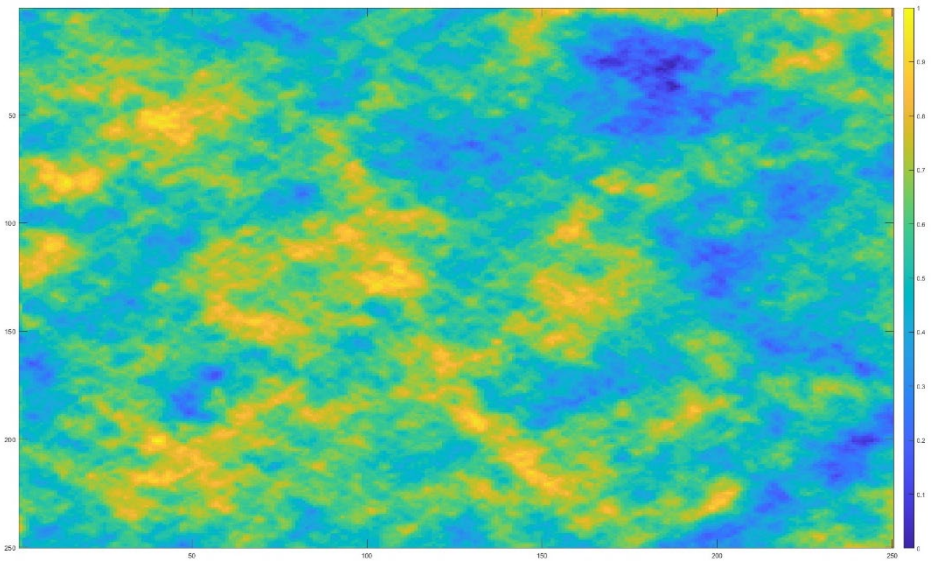


Abbildung 17: Beispielhafte Visualisierung eines Perlin Rauschens

Für die Modellierung der Verschmutzung wird angenommen, dass das Rauschen vor dem Sensor platziert wird und dort die Verschmutzung imitiert. Um die Auswirkungen der Verschmutzung zu bestimmen, wird der Verlauf des Laser-Strahls, welcher den jeweiligen Punkt in der Punktwolke detektiert, nachvollzogen und die Koordinate ermittelt, durch welche der Laserstrahl das Rauschen schneidet (s. Abbildung 9).

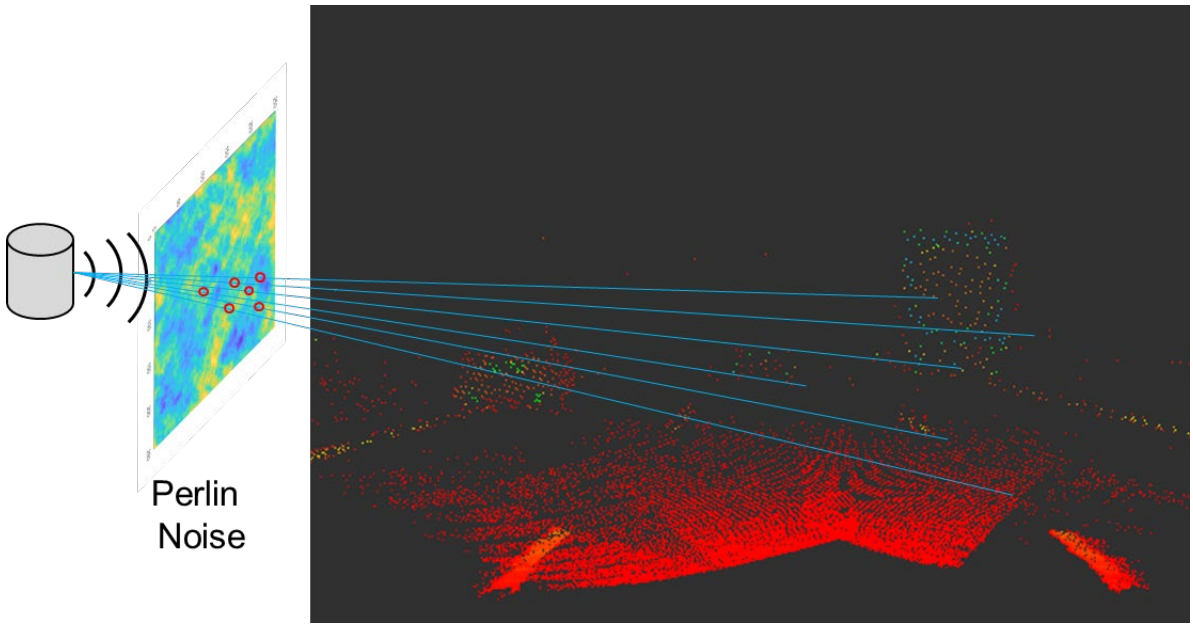


Abbildung 18: Sensordatenmanipulation mittels des Perlin Rauschens

Um nun die Auswirkung der Verschmutzung auf die Daten zu ermitteln, ist die Definition von zwei Parametern (val_low, val_high) notwendig, welche mit dem z-Wert im Schnittpunkt (val_per) aus dem Rauschen abgeglichen werden. Dabei können folgende Szenarien eintreten:

1. Der Parameter val_low definiert die Grenze, ab wann die Verschmutzung Auswirkungen auf die Daten hat. Somit gilt:

- val_per < val_low → keine Auswirkungen auf den detektierten Punkt.

2. Der Parameter val_high legt den Wert fest, ab wann die Laserstrahlen nicht die Verschmutzung durchdringen können und somit das Objekt nicht detektierten können. Daraus folgt:

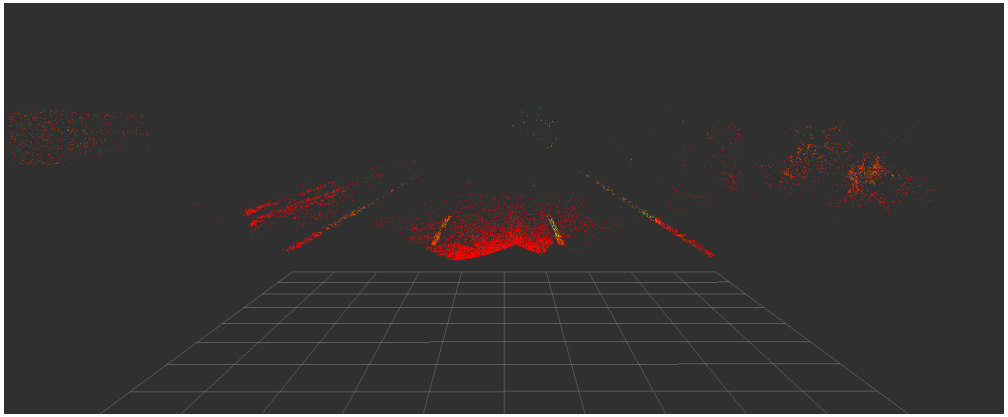
- val_high ≤ val_per → Der detektierte Punkt kann nicht durch die Verschmutzung detektiert werden und wird aus der Punktwolke gelöscht.

3. Liegt der Wert val_per zwischen dem unteren und oberen Parameter (val_low, val_high), wird die Reichweite des Sensors linear zur Höhe des Wertes, basierend auf einer maximalen Reichweite des Sensors (dist_max), eingegrenzt:

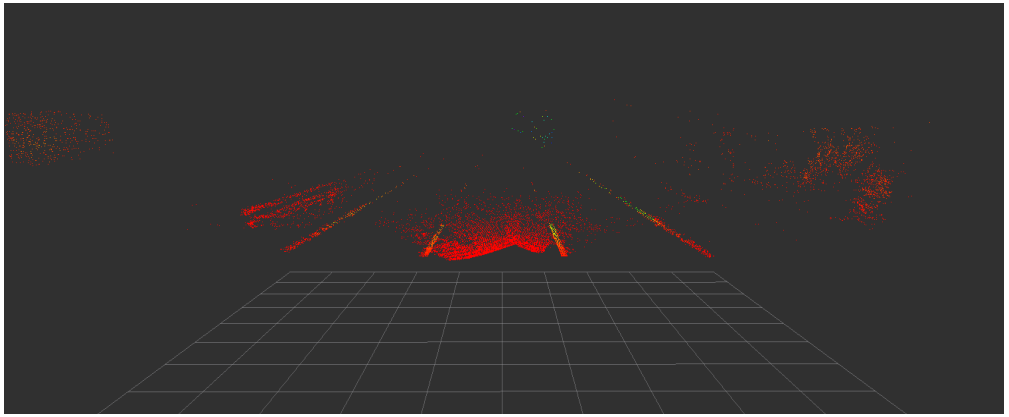
- val_low ≤ val_per < val high
    - ➔ Begrenzung der Reichweite auf: (val_per-val_low)/(val_high-val_low)·dist_max
    - ➔ Liegt die Entfernung des detektierten Punktes außerhalb des neu berechneten maximalen Reichweite, wird er aus der Punktwolke gelöscht.
    - ➔ Liegt die Entfernung des detektierten Punktes innerhalb der neu berechneten Reichweite, wird er in der Punktwolke beibehalten.

Abbildung 11 zeigt die Auswirkungen der Manipulation mittels des Perlin Rauschens mit unterschiedlichen Ausprägungen der Parameter val_low und val_high auf die Punktwolke. Es ist zu erkennen, dass mit sinkenden Werten, sowohl für val_low als auch val_high, die Auswirkungen der Manipulation größer werden und insbesondere die Punkte wegfallen, welche eine höhere Entfernung zum Sensor haben.

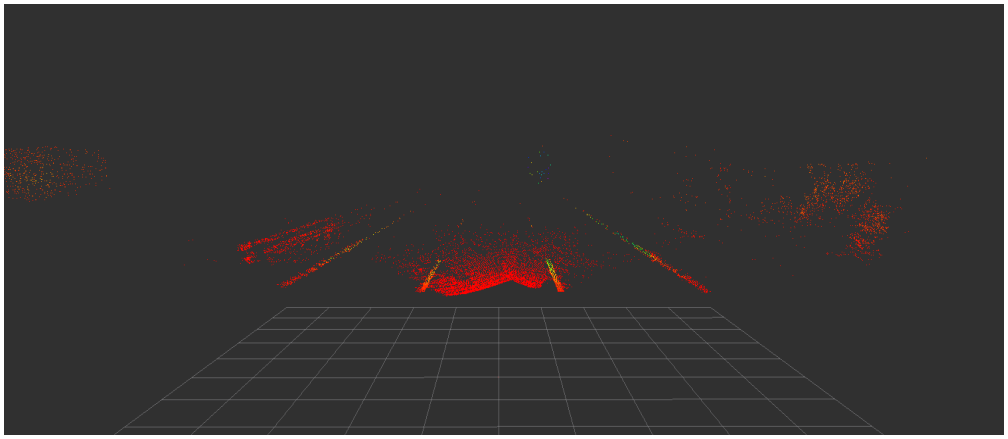
Original



val_low=0.5 val_high=0.95



val_low=0.2 val_high=0.8



val_low=0.05 val_high=0.5

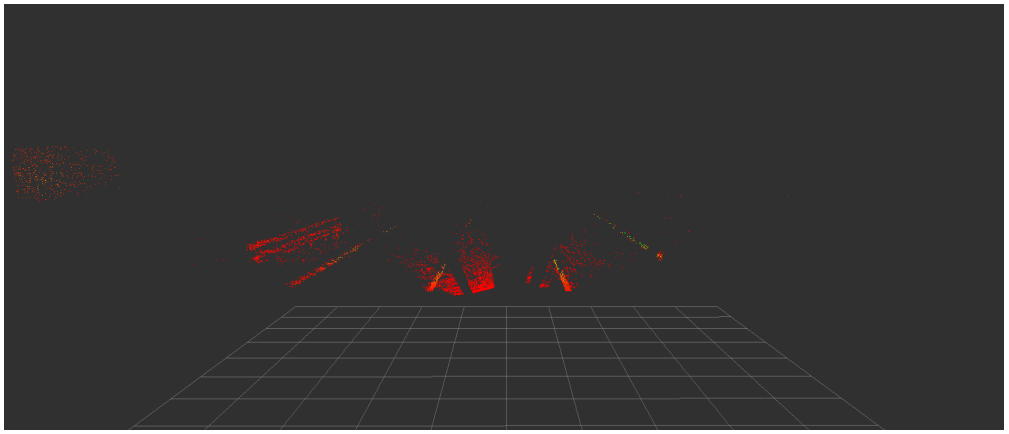


Abbildung 2: Visualisierung der Sensordatenmanipulation über die Sensorkontamination

Zusammenfassung und Ausblick

Im Rahmen des Use Case 1 wurde durch das IQZ ein Modul erarbeitet, mit welchem LiDAR-Daten, die in sogenannten rosbag-Dateien abgespeichert sind, mit unterschiedlichen Manipulationsmechanismen verändert und in einem Data Replay verwendet werden können. Diese Manipulation dient insbesondere dafür, die Robustheit der Perzeptionsalgorithmen zu überprüfen und mögliche Schwachstellen bzw. Verbesserungspotential in diesen zu identifizieren.

Die Arbeiten bilden einen Proof-of-Concept dar und sollen die Möglichkeit zur nachträglichen Datenmanipulationen aufzeigen. Die implementierten Modelle zur Manipulation wurden beispielhaft ausgewählt und erfordern zukünftig eine weitere Verifikation und Validierung mit realen Daten, welche entsprechende Fehler in den Daten aufweisen. Darüber hinaus die Integration weiterer Modelle zur Manipulation denkbar, um noch mehr Szenarien zur Überprüfung der Robustheit des Perzeptionsalgorithmus generieren zu können.

2.2. Wichtigste Positionen des zahlenmäßigen Nachweises

Mit Ausnahme der Reisekosten wurden die vorkalkulierten Gesamtkosten im Projekt eingehalten.

Position	Abgerechnete Kosten	Kosten lt. Vorkalkulation	Abweichung [€]	Abweichung [%]
0837 Personalkosten	773.369,19 €	775.730,25 €	-2.361,06 €	- 0,3 %
0838 Reisekosten	14.153,74 €	16.500,00 €	-2.346,26 €	- 14,2 %
0847 Abschreibungen	14.413,21 €	15.211,20 €	-797,99 €	- 5,2 %
0850 Vorhabenskosten	24.112,46 €	24.112,46 €	-0,00 €	0,0 %

Tabelle 2: Übersicht über die Einhaltung der Kostenplanung

Die erhebliche Abweichung der Reisekosten kommt dadurch zustande, dass durch die langfristigen Folgen der Corona-Pandemie eine Vielzahl von Treffen online durchgeführt wurde. Somit entfielen einige der geplanten Reisen zu Projektpartnern.

Zudem wurden Konsortialtreffen genutzt, um bilaterale Workshops einzelner Partner im Vorfeld oder im Nachgang anzuhängen. Auch durch diese Maßnahme wurden Reisekosten eingespart.

2.3. Notwendigkeit und Angemessenheit der geleisteten Arbeit

Sicherheit, Verfügbarkeit und Zuverlässigkeit sind die wichtigsten nicht-funktionalen Anforderungen bei der Entwicklung automatisierter oder autonomer Fahrzeugsysteme. Ihre Beschaffenheit mit vernetzter Hard- und Software, funktionalen Abhängigkeiten, geforderten (Teil-)Redundanzen führt zu Systemarchitekturen, die mit konventionellen Methoden zuverlässigkeits- und sicherheitstechnisch nicht abgebildet werden können.

Bislang konnten lediglich hardware- oder softwareseitige Betrachtung linearer Systemarchitekturen zuverlässigkeitstechnisch quantitativ analysiert werden und dies auch nur bis zu einem bestimmen Detaillierungslevel. Worst-Case-Betrachtungen erforderten eine aufwendigere Auslegung der Systeme oder frühzeitigen Degradationsstrategien (Limp-Aside, Limp-Home), die die Wirtschaftlichkeit bzw. Akzeptanz der Systeme stark gemindert haben.

Innerhalb dieses Teilvorhabens wurde nicht nur die mathematischen Modellierungen durchgeführt, sondern es wurde auch ein Stand-Alone-Tool geschaffen, welche unmittelbar durch OEM, 1st-TIER oder andere Kunden direkt anwendbar ist. Das Tool wurde mit einem TRL 8 abgeschlossen (höher als geplant) und kann unmittelbar eingesetzt und vertrieben werden.

2.4. Voraussichtlicher Nutzen, Verwertbarkeit der Ergebnisse im Sinne des fortgeschriebenen Verwertungsplan

Mit RT-CARS steht erstmal ein Tool zur Verfügung mit dem Zuverlässigkeits-, Sicherheits- und Verfügbarkeitskenngrößen für vernetzte E/E/PE/KI-Systeme auf Basis von Gesamtsystemzuständen unter Berücksichtigung komplexer funktionaler Abhängigkeiten und technischer Regeln simulationsbasiert modelliert werden können.

Diese Art der Analysen ist im Rahmen des Produktentstehungsprozess unabdingbar und werden unmittelbar benötigt. In der Design- und Vorentwicklungsphase können unterschiedliche Systemarchitekturen auf Basis ihrer funktionalen Pfade zuverlässigkeits- und sicherheitstechnisch verglichen werden. Dies hat den unmittelbaren Vorteil bei OEM und 1st-TIER, dass die Systeme zuverlässigkeits- und sicherheitstechnisch maximal effizient entwickelt werden können. Gerade in Bereichen mit sehr hohen Sicherheitsanforderungen in denen Redundanzen vorgeschrieben und technisch absolut notwendig sind, besteht hier ein großer Mehrwert.

In der Phase der Systemvalidierung und -verifizierung können mittels RT-CARS homologationsnotwendige quantitative Zuverlässigkeits- und Sicherheitsnachweise erbracht werden.

RT-CARS lässt sich sowohl kurz- als auch langfristig direkt in Projekten verwerten. Die Möglichkeit auf Systemebene funktionspfadabhängige E/E/PE-Architekturen zu modellieren und anschließend über Simulationsmethoden bzgl. Ihrer Zuverlässigkeits-, Funktionsverfügbarkeits- und Sicherheitseigenschaften in Form von quantitativen KPIs zu bewerten ist nach aktuellem Stand in der Funktionalität einzigartig.

RT-CARS ist nicht branchenspezifisch und kann in allen technischen Disziplinen mit (teil-)redundanten Systemauslegungen verwendet werden. Neben den bereits vorgesehenen Anwendungen in der Luftfahrt und der Windenergietechnik sind ebenfalls bereits erste Projekte gestartet, in denen RT-CARS auch in den Sicherheits- und Verteidigungsindustrien angewendet wird. Gerade in dieser Industrie ist der Innovationsgrad aktuell sehr noch und es wird eine Vielzahl neuer E/E/PE/KI-Systeme entwickelt, auf die RT-CARS unmittelbar zur Zuverlässigkeits- und Sicherheitsqualifikation angewendet werden kann.

Die in innerhalb von Use-Case 1 entwickelte Möglichkeit zur rosbag-basierten nachträglichen Sensorsimulation lässt sich für beliebige MiL-Anwendungen (Model-In-The-Loop) adaptieren. Innovativ ist hierbei die Methodik der Fehlerinjektion statt der klassischen Sollfunktionssimulation. Dies ermöglicht Aussagen über die Robustheit der geprüften Anwendungen in Hinblick auf zu erwartende Störungen (z.B. Verschmutzung). Die Verwertbarkeit erfordert, anders als RT-CARS, einen klaren Anwendungsfall und eine individuelle Anpassung der Algorithmik, während RT-CARS universell einsetzbar ist.

2.5. Fortschritte an anderen Stellen

Es ist nicht bekannt das in Bezug auf das Teilvorhaben an anderer Stelle methodisch oder inhaltlich vergleichbare Fortschritte veröffentlich wurden.

2.6. Veröffentlichungen der Ergebnisse

Horeis, Timo Frederik; Heinrich, Johannes; Plinke, Fabian (2022): A Self-Adapting Reconfiguration Process for The Failure Management of Highly Automated Vehicles. In: Maria Chiara Leva, Edoardo Patelli, Luca Podofillini und Simon Wilson (Hg.): Proceeding of the 32nd European Safety and Reliability Conference. 32nd European Safety and Reliability Conference, 28.08.2022 - 01.09.2022. Singapore: Research Publishing Services, S. 838–845.

Horeis, Timo Frederik; Rinaldo, Rhea C. (2023) Towards Verification of Self-Healing for Autonomous Vehicles. In: Mário P. Brito, Terje Aven, Piero Baraldi, Marko Čepin und Enrico Zio (Hg.): Proceeding of the 33rd European Safety and Reliability Conference. 33rd European Safety and Reliability Conference, 03.09.2023 - 07.09.2023. Singapore: Research Publishing Services, S. 334–341.