HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN

FAKULTÄT MASCHINENBAU

STUDIENGANG FAHRZEUGTECHNIK

STUDIENRICHTUNG KRAFTFAHRZEUGTECHNIK

Diplomarbeit

Entwicklung und Implementierung von neuen Testverfahren für automatisierte Fahrfunktionen

vorgelegt von

Jan Blumenstein

Betreuer HTW	Prof. Dr. rer. nat. Toralf Trautmann Dipl.-Ing. (FH) Franziskus Mendt
Abgabetermin	16.10.2024

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACC	**A**daptive **C**ruise **C**ontrol
AFGBV	**A**utonome-**F**ahrzeuge-**G**enehmigungs-und-**B**etriebs-**V**erordnung
BMDV	**B**undes**m**inisterium für **D**igitales und **V**erkehr
BMJ	**B**undes**m**inisterium der **J**ustiz
BMWK	**B**undes**m**inisterium für **W**irtschaft und **K**limaschutz
DBSCAN	**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise
DGPS	**D**ifferential **G**lobal **P**ositioning **S**ystem
DSD	**D**riving **S**cenario **D**esigner
ECU	**E**lectronic **C**ontrol **U**nit (Steuergerät)
GPS	**G**lobal **P**ositioning **S**ystem
HiL	**H**ardware-**i**n-the-**L**oop
ICP	**I**terative-**C**losest-**P**oint
INS	**I**nertial **N**avigation **S**ystem (Trägheitsnavigationssystem)
LiDAR	**L**ight **D**etection **a**nd **R**anging
MiL	**M**odel-**i**n-the-**L**oop
MSAC	**M**-estimator **Sa**mple **C**onsensus
Near-IR	**Near-I**nf**r**ared (Nahinfrarot)
PLVs	**P**ark**l**ücken**v**erstelleinheiten
PTP	**P**recision **T**ime **P**rotocol
SDK	**S**oftware **D**evelopment **K**it
SLAM	**S**imultaneous **L**ocalization **a**nd **M**apping
SiL	**S**oftware-**i**n-the-**L**oop
TBC	**T**est **b**ench **c**onfiguration (Testbenchkonfiguration)
TCF	**T**est **c**on**f**iguration (Testkonfiguration)
ToF	**T**ime **o**f **F**light
ViL	**V**ehicle-**i**n-the-**L**oop
VUT	**V**ehicle **u**nder **T**est (Versuchsfahrzeug)
WLAN	**W**ireless **L**ocal **A**rea **N**etwork

Verzeichnis der Formelzeichen und Symbole

Zeichen	Einheit	Bedeutung
$\boldsymbol{A}$	m	Gemessener Abstand des VUT während der Vorbeifahrt
$\boldsymbol{B}$	m	Berechnete Breite der Parklücke
$\boldsymbol{c}$	$\frac{m}{s}$	Lichtgeschwindigkeit
$\boldsymbol{d}$	m	Entfernung
$\boldsymbol{dt}$	s	Zeitintervall
$\boldsymbol{F}$	m	Breite des VUT
$\boldsymbol{I}$	$-$	Einheitsmatrix
$\boldsymbol{K}$	$-$	Kreuzproduktmatrix von k
$\boldsymbol{k}$	$-$	Kreuzprodukt aus Ebenennormale und z-Achse
$\boldsymbol{L}$	m	Berechneter Abstand zum linken benachbarten Fahrzeug
$\boldsymbol{m}$	$-$	Parameter der Geradengleichung
$\boldsymbol{n}$	$-$	Ebenennormale / Parameter der Ebenengleichung
$\boldsymbol{n_x, n_y, n_z}$	$-$	Komponenten der Ebenennormalen
$\boldsymbol{n'}$	$-$	Kreuzprodukt
$\boldsymbol{p}$	$-$	Punkt im Raum
$\boldsymbol{p'}$	$-$	Rotierter Punkt im Raum
$\boldsymbol{r}$	m	Abstand zwischen Objekt und Sensor
$\boldsymbol{R}$	m	Gemessener Abstand zum rechten benachbarten Fahrzeug
$\boldsymbol{R_O}$	$-$	Rodrigues-Rotationsformel
$\boldsymbol{R_X(\alpha)}$, $\boldsymbol{R_Y(\beta)}$, $\boldsymbol{R_Z(\gamma)}$	$-$	Rotationsmatrix für jede Raumachse, abhängig vom jeweiligen Winkel
$\boldsymbol{S}$	m	Spurweite des VUT
$\boldsymbol{ToF}$	s	Laufzeit des Laserimpulses
$\boldsymbol{T}$	m	Translation
$\boldsymbol{T_{x-FS}}$	m	Translation in x-Richtung zwischen Fahrzeug und Sensor

$\boldsymbol{T_{x-WF}}$	m	Translation in x-Richtung zwischen Welt und Sensor
$\boldsymbol{T_{y-FS}}$	m	Translation in y-Richtung zwischen Fahrzeug und Sensor
$\boldsymbol{t_x, t_y, t_z}$	m	Translationskomponenten von *t* für jede Raumachse
$\boldsymbol{U_{Rad}}$	m	Radumfang des VUT
$\boldsymbol{v, w}$	$-$	Richtungsvektoren
$\boldsymbol{X_L}$	$-$	Koordinatenachse Ouster Sensor
$\boldsymbol{x_W}$	m	x-Koordinate eines Punktes im Weltkoordinatensystem
$\boldsymbol{x_F}$	m	x-Koordinate eines Punktes im Fahrzeugkoordinatensystem
$\boldsymbol{x_S}$	m	x-Koordinate eines Punktes im Sensorkoordinatensystem
$\boldsymbol{Y_L}$	$-$	Koordinatenachse Ouster Sensor
$\boldsymbol{y_W}$	m	y-Koordinate eines Punktes im Weltkoordinatensystem
$\boldsymbol{y_F}$	m	y-Koordinate eines Punktes im Fahrzeugkoordinatensystem
$\boldsymbol{y_S}$	m	x-Koordinate eines Punktes im Sensorkoordinatensystem
$\boldsymbol{z}$	$-$	Koordinatenachse
$\boldsymbol{Z_L}$	$-$	Koordinatenachse Ouster Sensor
α	$°\ bzw. rad$	Winkel für Rotationsmatrix um die x-Achse
β	$°\ bzw. rad$	Winkel für Rotationsmatrix um die y-Achse
γ	$°\ bzw. rad$	Winkel für Rotationsmatrix um die z-Achse
Θ	$°\ bzw. rad$	Horizontaler Winkel (Azimut) / Gierwinkel des VUT
$\boldsymbol{\theta_{encoder}}$	$°\ bzw. rad$	Encoder-Winkel
Φ	$°\ bzw. rad$	Vertikaler Winkel
$\boldsymbol{\omega}$	$\frac{1}{s}$	Raddrehzahl

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Entwicklung automatisierter Fahrfunktionen ist eines der spannendsten und zugleich anspruchsvollsten Themen der Fahrzeugtechnik. Indem sie bestimmte Aufgaben wie Spurhalten oder automatisches Bremsen übernehmen, bilden diese Systeme die Grundlage für autonomes Fahren. In den letzten Jahren ist die Vision des autonomen Fahrens immer greifbarer geworden, sodass selbstfahrende Autos in naher Zukunft zum alltäglichen Straßenbild gehören könnten. Um diese Technologie zuverlässig und sicher zu gestalten, müssen umfassende Test- und Validierungsverfahren entwickelt und kontinuierlich verbessert werden.

In Deutschland wurde mit der Autonome-Fahrzeuge-Genehmigungs-und-Betriebs-Verordnung (AFGBV) eine rechtliche Grundlage geschaffen, die den Betrieb autonomer Fahrzeuge regelt. Diese Verordnung schreibt vor, dass Fahrzeuge mit autonomer Fahrfunktion täglich vor Fahrtantritt einer „erweiterten Abfahrtskontrolle“ unterzogen werden müssen. Diese Kontrolle ist umfassend und beinhaltet neben der Überprüfung von Lenk-, Licht-, Fahrwerks-, Sicherheits- und Bremssystemen auch die Prüfung „sicherheitsrelevanter elektronisch geregelter Fahrzeugsysteme sowie der Sensorik zur Erfassung externer und interner Parameter“ ([1] BMDV, BMJ und BMWK 2022, S.10). Hierbei spielt die Umfeldsensorik eine große Rolle, da sie für die Erfassung der Umgebung verantwortlich ist und damit wichtige Informationen für die Entscheidungsfindung des Fahrzeugs liefert [1].

Da die AFGBV erst 2022 in Kraft getreten ist, befinden sich derzeit Konzepte in der Entwicklung, wie die vorgeschriebene erweiterte Abfahrtskontrolle konkret gestaltet und mit welchen Mitteln sie effizient durchgeführt werden kann. Vor diesem Hintergrund zielt diese Diplomarbeit darauf ab, bestehende Testverfahren für automatisierte Fahrfunktionen weiterzuentwickeln. Im Rahmen dieser Arbeit soll ein bereits vorhandener Testfall in der Testautomatisierungs-software ecu.test erweitert und optimiert werden.

2 Theoretische Grundlagen

Zu Beginn dieser Arbeit werden die theoretischen Grundlagen erläutert. Der Schwerpunkt liegt auf den In-the-Loop-Testverfahren im Rahmen des V-Modells, um die Bedeutung dieser in verschiedenen Entwicklungsstadien zu verdeutlichen. Im Anschluss wird auf die Softwarelösungen, insbesondere ecu.test, eingegangen, die eine zentrale Rolle in der Testautomatisierung spielen. Abschließend wird die verwendete Sensorik näher betrachtet.

2.1 In-the-Loop Testverfahren im V-Modell

Das V-Modell ist ein etabliertes Vorgehensmodell in der System- und Softwareentwicklung, besonders in sicherheitskritischen Bereichen wie der Entwicklung automatisierter Fahrfunktionen. Es teilt den Entwicklungsprozess in klar definierte Phasen auf, die von der Anforderungsanalyse über das Design bis hin zur Integration und Verifikation reichen. Jede dieser Phasen wird direkt durch eine passende Testphase ergänzt, wodurch eine enge Verbindung zwischen Entwicklung und Prüfung entsteht. Auf diese Weise lassen sich mögliche Fehler frühzeitig erkennen und beheben – was besonders wichtig ist, da Fehler in späteren Phasen oft deutlich teurer und aufwändiger zu korrigieren sind. Das V-Modell sorgt damit für eine transparente und nachvollziehbare Entwicklung und sichert die Qualität und Einhaltung von Sicherheitsstandards.

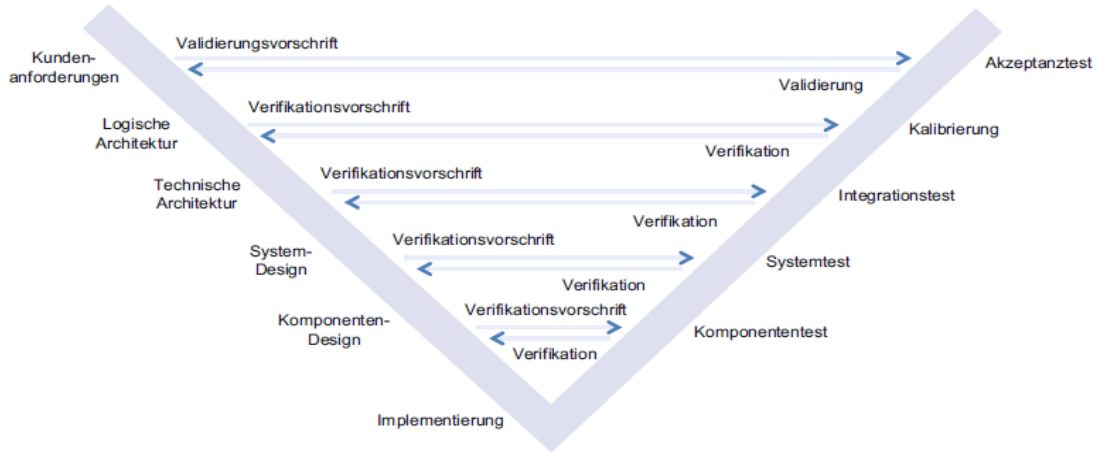


Abbildung 2.1 Entwicklungsprozess nach dem V-Modell [2]

Abbildung 2.1 zeigt das vollständige V-Modell mit allen Phasen. Es beginnt bei der Analyse der Produktanforderungen und reicht über die Implementierung bis

zur abschließenden Validierung, die sicherstellt, dass alle Kundenanforderungen erfüllt sind. Jeder Schritt im absteigenden Ast (Designphase) wird dabei durch einen entsprechenden Testschritt im aufsteigenden Ast (Testphase) verifiziert. [2]

In-the-Loop-Tests wie Hardware-in-the-Loop (HiL) oder Software-in-the-Loop (SiL) sind besonders wichtig in der Validierungs- und Verifikationsphase des V-Modells. Sie simulieren reale Bedingungen und erlauben es, das Verhalten der Fahrzeugsysteme in einer kontrollierten, aber dennoch realistischen Umgebung auf ihre Zuverlässigkeit und Sicherheit hin zu prüfen. Dadurch lassen sich potenzielle Probleme frühzeitig erkennen und beheben, bevor das System tatsächlich im Fahrzeug eingesetzt wird. In Abbildung 2.2 ist zu sehen, wann genau die verschiedenen In-the-Loop-Tests im V-Modell zum Einsatz kommen.

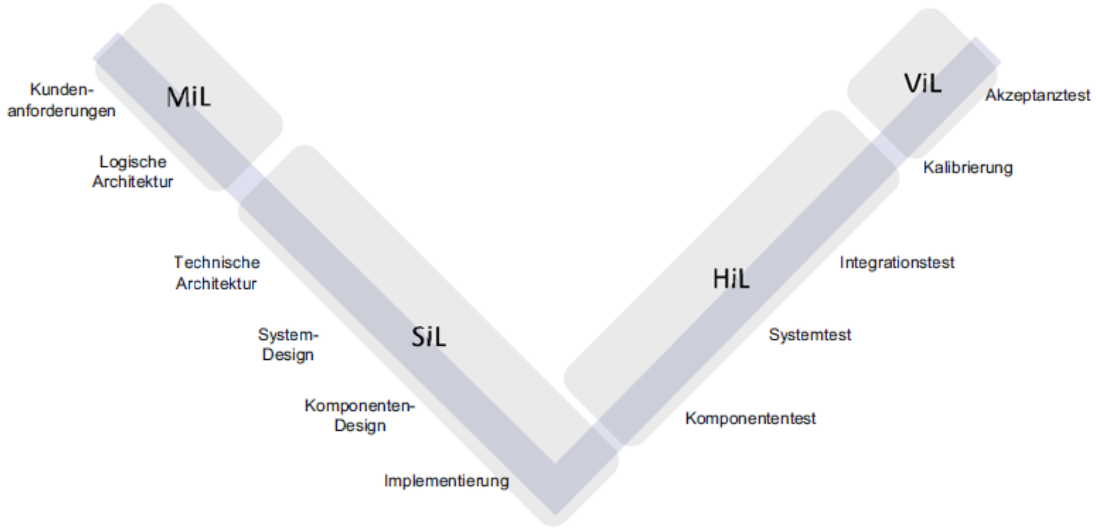


Abbildung 2.2 In-the-Loop-Methoden im V-Modell [2]

Die Verfahren sind dabei die bereits erwähnten Software-in-the-Loop und Hardware-in-the-Loop. Hinzu kommen Vehicle-in-the-Loop (ViL) und Model-in-the-Loop (MiL). Auf die ersten drei genannten Verfahren wird in den folgenden Kapiteln näher eingegangen.

2.1.1 Software-in-the-Loop

Bei Software-in-the-Loop Tests wird die Software eines Systems nicht auf der finalen Hardware, sondern in einer virtuellen Umgebung getestet. Die zu prüfende Software läuft auf einem Rechner, der die Umgebung des Fahrzeugs simuliert. Dabei werden die Umgebungsbedingungen und die Interaktionen des Systems durch ein Simulationstool (z. B. MATLAB Simulink) nachgebildet [3].

Der Vorteil von SiL-Tests liegt darin, dass Fehler und Schwächen der Software sehr früh im Entwicklungsprozess entdeckt und behoben werden können. Das spart sowohl Zeit als auch Kosten, da spätere Korrekturen viel aufwändiger wären. Ein weiterer Vorteil ist die Wiederholbarkeit. Da die Tests in einer virtuellen Umgebung stattfinden, können sie unter exakt denselben Bedingungen erneut durchgeführt werden.

Ein Beispiel für solch einen Test bei automatisierten Fahrfunktionen ist die Prüfung eines Spurhalteassistenten. Hierbei wird die Steuerungssoftware des Systems in einer simulierten Umgebung getestet, ohne dass reale Fahrzeughardware benötigt wird. Virtuelle Sensordaten, wie Kamerabilder zur Erkennung von Fahrbahnmarkierungen, werden der Software für die Berechnungen zugeführt. Das Szenario simuliert ein Fahrzeug, das von der Spur abweicht, woraufhin die Software das Fahrzeug wieder in die Spur lenken muss. Der Test überprüft, ob korrekt auf die Spurabweichung reagiert wird. So kann validiert werden, dass die Software fehlerfrei arbeitet, noch bevor sie in einem realen Fahrzeug zum Einsatz kommt. [4, 5]

2.1.2 Hardware-in-the-Loop

Beim Hardware-in-the-Loop Test kommt die reale Fahrzeug-Hardware in einer simulierten Umgebung zum Einsatz. Das Steuergerät funktioniert wie im echten Fahrzeug, während die Umgebung mithilfe einer Simulation nachgebildet wird. So lässt sich überprüfen, wie das Steuergerät auf bestimmte Fahrbedingungen reagiert – ohne ein tatsächliches Fahrzeug zu benötigen. Abbildung 2.3 veranschaulicht das Konzept eines HiL-Tests. Das Steuergerät verhält sich so, als würde es auf echte Fahrbedingungen auf offener Straße reagieren. Die virtuelle Fahrzeugumgebung stellt dabei die Eingangssignale für das Steuergerät bereit. Dieses verarbeitet die Signale wie im realen Betrieb und gibt die

Ausgangsparameter an die jeweiligen Aktoren weiter. Ein Simulationscomputer steuert dabei die virtuelle Fahrzeugumgebung.

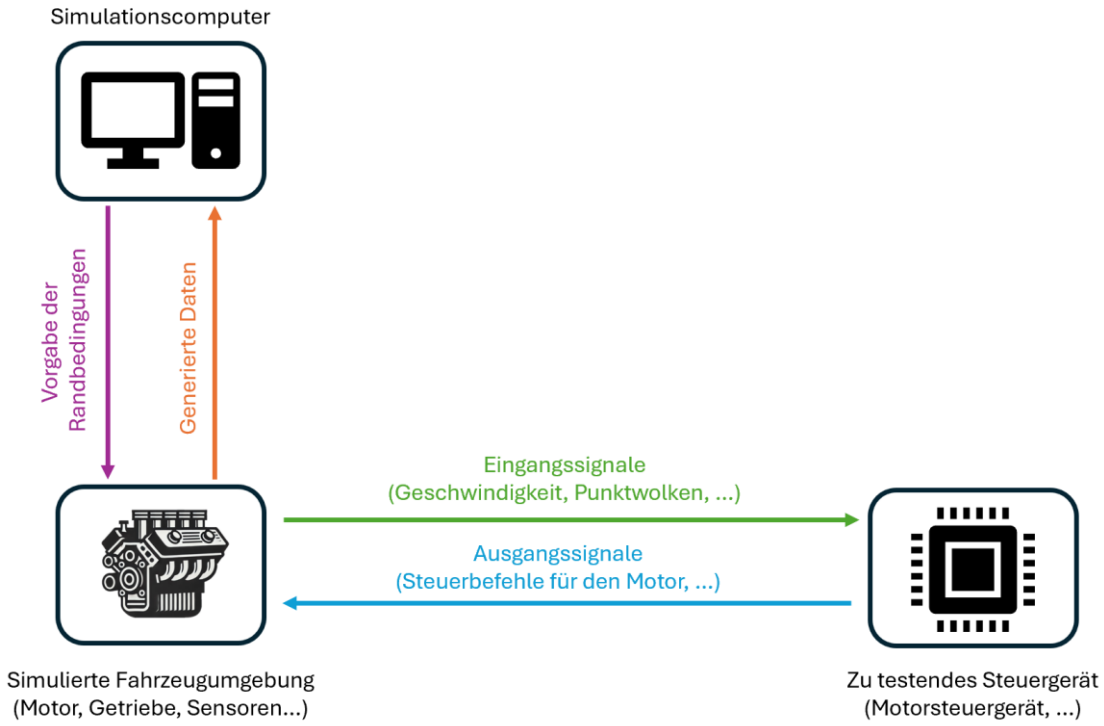


Abbildung 2.3 Schematische Darstellung des HiL-Tests

Genau wie SiL-Tests, bieten HiL-Tests eine hohe Flexibilität, Kosteneffizienz und Sicherheit. Zudem lassen sich verschiedene Szenarien durchspielen, darunter auch solche, die im echten Verkehr zu riskant oder schwer reproduzierbar wären. [2]

2.1.3 Vehicle-in-the-Loop

Im Gegensatz zu SiL und HiL, bei denen Software oder Hardware isoliert getestet werden, ermöglicht ViL-Testing eine ganzheitliche Validierung des gesamten Fahrzeugs in einer simulierten Umgebung. Dabei wird ein Fahrzeug unter realen Bedingungen getestet, jedoch in einer Umgebung, in der die Sensordaten aus einer Simulation stammen. Das Fahrzeug fährt auf einer Teststrecke, aber die Steuergeräte erhalten Daten aus einer simulierten Umgebung. Zum Beispiel könnte das zu testende Steuergerät auf ein simuliertes kreuzendes Fahrzeug mit einer Notbremsung reagieren, obwohl dieses Fahrzeug in der Realität nicht vorhanden ist. [6]

Abbildung 2.4 zeigt einen ViL-Testfall. Rechts ist die simulierte Umgebung dargestellt, auf die das reale Fahrzeug links reagiert. Konkret wird eine Notbremsungssituation simuliert, bei der das vorausfahrende Fahrzeug abrupt bremst. Das zugehörige Video ist unter der angegebenen Bildquelle abrufbar.

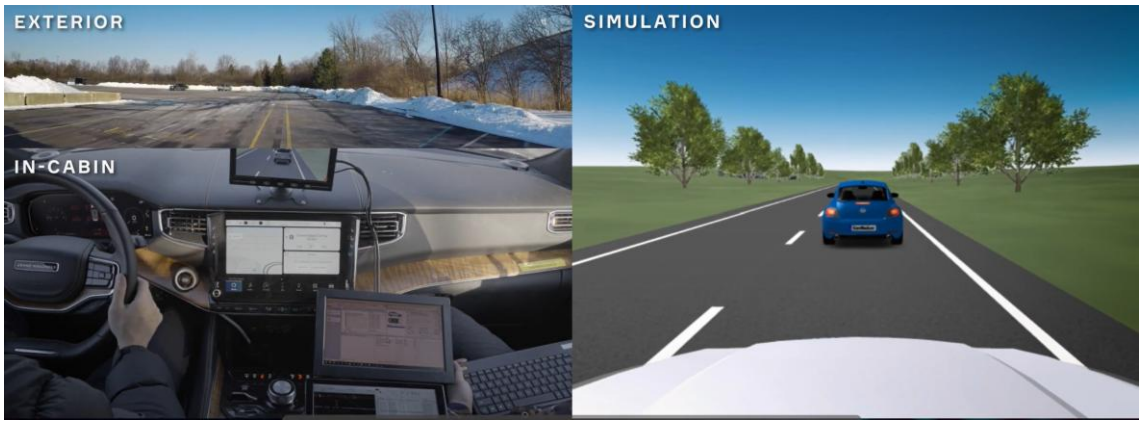


Abbildung 2.4 Vehicle in the Loop [6]

Der Vorteil von ViL liegt darin, dass das Zusammenspiel von Software, Hardware und anderen Fahrzeugkomponenten in einer realitätsnahen Umgebung geprüft werden kann. Dies ermöglicht den Entwicklern, den Einfluss unterschiedlicher Szenarien auf das Fahrzeug zu beobachten, ohne die Risiken und Kosten von Erprobungsfahrten einzugehen. Besonders nützlich ist das Verfahren für die Überprüfung komplexer Fahrerassistenzsysteme, die in einer simulierten Umgebung umfassend getestet werden können, bevor sie auf die Straße kommen. [2]

2.2 ecu.test

Um den steigenden technischen und rechtlichen Anforderungen, sowie der zunehmenden Komplexität der Software in automatisierten Fahrfunktionen gerecht zu werden, sind leistungsfähige Testautomatisierungswerkzeuge unverzichtbar. Die Software ecu.test, entwickelt von tracetronic in Dresden, ist ein auf Python basierendes Tool, welches eine umfassende Lösung zur Durchführung automatisierter Funktions- und Systemtests im Automobilsektor bietet. Sie zielt darauf ab, Testprozesse für elektronische Steuergeräte und eingebettete Systeme zu automatisieren.

Wie in Abbildung 2.5 dargestellt, lässt sich ecu.test in allen Phasen des V-Modells einsetzen. Auf diese Weise wird eine durchgängige und konsistente Testumgebung geschaffen, die den gesamten Entwicklungsprozess umfasst. [7]

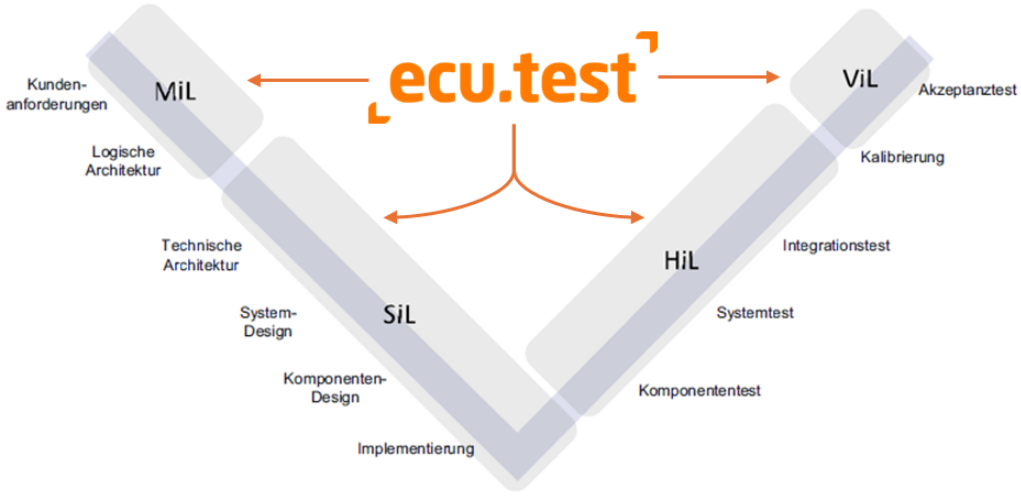


Abbildung 2.5 ecu.test Anwendungsmöglichkeiten (in Anlehnung an [2])

Eine Vielzahl von Fahrzeugdiagnose-Tools und -Schnittstellen wie CAN, LIN, und Ethernet können integriert werden. Ebenfalls besteht die Möglichkeit andere Programme wie MATLAB/Simulink, Carla, CarMaker etc. in den Automatisierungsprozess einzubinden [8].

Die aktuelle Version von ecu.test ist 2024.2 (Stand: September 2024). Es wurde jedoch die Version 2023.4.4 für diese Arbeit gewählt, da in ihr die Implementierung der verwendeten Sensorik fehlerfrei funktioniert.

2.2.1 Aufbau und Handhabung der Software

Für die erfolgreiche Umsetzung der Testverfahren ist ein fundiertes Verständnis des Aufbaus und der Bedienung der Software ecu.test von Bedeutung. Im Folgenden werden kurz die Schritte zur Erstellung eines Workspaces, die Konfiguration von Testfällen, sowie die Einbindung relevanter Tools beschrieben. Detaillierte Informationen sind der ecu.test Dokumentation zu entnehmen.

Die Arbeit mit ecu.test beginnt mit der Erstellung eines Workspaces. Dieser dient als zentrales Verzeichnis für alle Dateien, die ecu.test zur Ausführung von Testfällen benötigt.

Durch die Nutzung von relativen Pfaden in der Arbeitsumgebung ist es möglich, Einstellungen projektübergreifend konsistent zu halten. Relative Pfade sind Dateipfade, die den Ablageort einer Datei oder eines Verzeichnisses im Verhältnis zum aktuellen Arbeitsverzeichnis oder einer anderen Referenzposition angeben [9]. Dies ist besonders hilfreich, wenn an einem Projekt auf mehreren Rechnern oder von mehreren Mitarbeitern gearbeitet wird. Es ist sinnvoll, für unterschiedliche Projekte oder Anwendungsfälle separate Workspaces zu erstellen, um gegenseitige Beeinflussungen zu minimieren. [8]

Nachdem ein Workspace erstellt wurde, wird ein Package angelegt. Dieses Package enthält die einzelnen Testschritte, mit denen das zu prüfende Produkt (z. B. ein Steuergerät) getestet wird. Testschritte sind grundlegende Aktionen, die typischerweise Anweisungen, wie das Lesen und Schreiben von Daten, die Berechnung von Werten sowie Kontrollstrukturen (z. B. Schleifen) umfassen können. Ein Testfall besteht aus mehreren dieser Testschritte, die in der Regel miteinander verknüpft sind. Diese Verknüpfung ermöglicht den Austausch von Daten und Variablen zwischen den einzelnen Schritten. Der gesamte Testfall wird schließlich als Package gespeichert und angezeigt. Abbildung 2.6 visualisiert den Aufbau eines Packages mit unterschiedlichen Testschritten.

#	Aktion / Name	Parameter	Erwartung / Wert	Bemerkung
1	Wait for User		'Breite des Fahrzeug...	
2	Berechnung	False	|-> userStop	
3	Loop Until (userStop is True)	1000	|-> loopCounter	
4	JOB-Ausführen: GenerateRecording	add_descrip: 'loop' + str(loop...	|-> filepath	
5	Wait	0.1 s		
6	BUS-Lesen: Fahrstufe	TEXT	'R' |-> Fahrstufe	
7	If (Fahrstufe == 'R')			
8	Then			
9	Berechnung	True	|-> userStop	
10	Else			
11	Berechnung	False	|-> userStop	
12	Berechnung_part1			
13	Loop Until (userStop is True)	1000		
14	Wait	2 s		
15	BUS-Lesen: Fahrstufe	TEXT	'P' |-> Fahrstufe	
16	If (Fahrstufe == 'P')			
17	Then			
18	JOB-Ausführen: GenerateRecording	add_descrip: 'side'; n_seconds: 1	|-> filepath	
19	Berechnung	True	|-> userStop	
20	Else			
21	Berechnung	user.run_matlab_script_part2.e...	|-> Right_Side_Dista...	
22	Wait	2 min		
23	Aus Inbox lesen: IN_Breite		|-> (Breite, Distance_...	

Abbildung 2.6 Package in ecu.test mit verschiedenen Testschritten

Parallel zum Package werden die Testkonfiguration (TCF - test configuration) und die Testbenchkonfiguration (TBC - test bench configuration) konfiguriert. Die

TCF umfasst Einstellungen für Plattform, Steuergeräte, Buszugriff und globale Konstanten. Hier können HiL-/SiL-Modelle, Fehlersimulationen und weitere Funktionszugriffe ausgewählt und angepasst werden. Steuergeräte und ihre Softwareschnittstellen werden parametriert und mit den entsprechenden Ports verknüpft.

Bei der TBC werden prüfplatzspezifische Einstellungen vorgenommen. Ein eigener Testbenchkonfigurationseditor steht zur Verfügung, um Hosts, Tools und Ports zu verwalten. Ein Host ist ein Rechner, auf dem die benötigten Tools installiert sind. Dies kann entweder der Rechner sein, auf dem ecu.test läuft, oder ein entfernter Rechner im Netzwerk, auf dem ein Tool-Server installiert ist.

In ecu.test wird jedes Software-Plugin, das den Zugriff auf Testgrößen ermöglicht, als Tool bezeichnet. Ein Tool steht in der Testbenchkonfiguration nur dann zur Verfügung, wenn es auf dem zugehörigen Host installiert ist. Tools bieten in der Regel mehrere Schnittstellen (Ports) an, um auf den Prüfling zuzugreifen. Zu den unterstützenden Tools gehören unter anderem MATLAB/Simulink, Vector, CARLA und viele weitere, die aus *ecu.test* Hilfe entnommen werden können.

Die Software unterstützt auch die Erstellung von eigenen Tooladaptern, um damit innerhalb des Packages zu arbeiten. Beispielsweise zeigt Abbildung 2.7 die Konfiguration des Ouster LiDAR-Tooladapters (LiDAR – Light Detection and Ranging) im TBC-Editor, welcher in der Master-Thesis von Schrocke entwickelt wurde und auf die in dieser Arbeit zurückgegriffen wird [10].

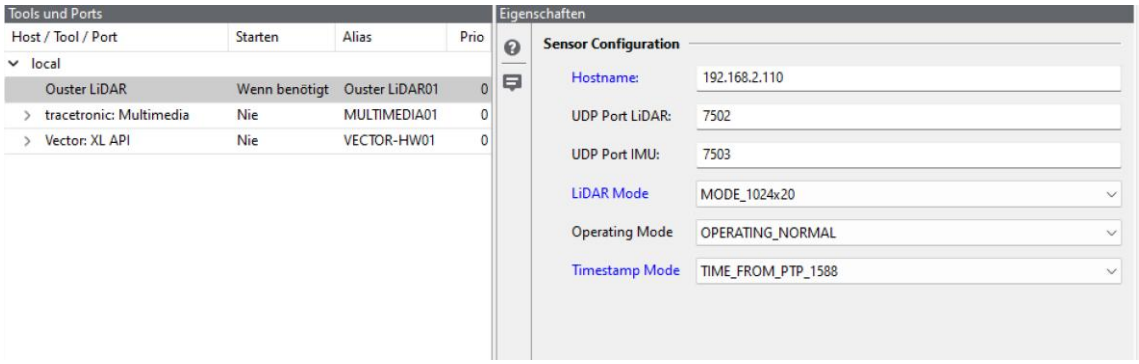


Abbildung 2.7 Anbindung des Ouster-Tooladapters in der TBC

Nachdem die Testbenchkonfiguration und die Testkonfiguration erstellt worden sind, werden sie gespeichert und gestartet. [8]

2.2.2 Traceanalyse

Die Traceanalyse in ecu.test spielt eine wichtige Rolle bei der Auswertung und Analyse der Daten, die während des Tests aufgezeichnet werden. Die Aufzeichnung der Kommunikationsprozesse zwischen unterschiedlichen Systemkomponenten wird als „Trace“ bezeichnet. Während der Durchführung der Tests erfolgt ein Austausch von Daten zwischen den Steuergeräten und anderen vernetzten Systemen. Die Traceanalyse erlaubt es, die Kommunikationsprotokolle und Signale ausführlich zu analysieren. Mögliche Signale sind beispielsweise die Informationen der Raddrehzahlsensoren oder die Fahrpedalstellung.

Einzelne Signale können aus den Tracedateien extrahiert und visualisiert werden. Dies hilft bei der Identifikation von Abweichungen im Kommunikationsverlauf. Mit Hilfe von Skripten und automatisierten Analysemethoden können wiederkehrende Analyseaufgaben automatisiert und effizient durchgeführt werden. [8, 7]

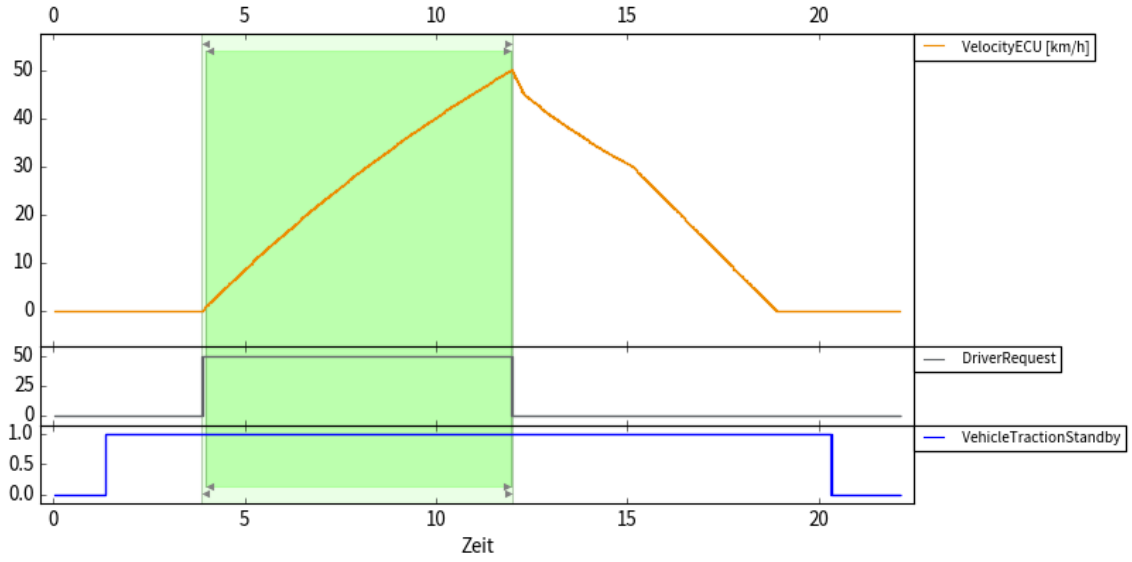


Abbildung 2.8 Beispiel einer Traceanalyse [11]

Abbildung 2.8 veranschaulicht beispielhaft eine Methode der Traceanalyse. Zuvor wurde der Antriebsstrang eines Elektrofahrzeugs mithilfe von ecu.test sowie weiteren Tools wie ETAS INCA und ControlDesk simuliert. Die hierbei generierten Simulationsdaten können anschließend gemäß den Anforderungen des Anwenders durch die Traceanalyse ausgewertet und visualisiert werden. In

der Abbildung ist der Geschwindigkeitsverlauf des simulierten Fahrzeugs bei einer Fahrpedalstellung (DriverRequest) von 50 % dargestellt. Der grün markierte Bereich kennzeichnet einen Abschnitt, in dem die vom Anwender festgelegten Bedingungen erfüllt sind – in diesem Fall eine Fahrpedalstellung größer als 0 %.

2.3 MATLAB

MATLAB (MATrix LABoratory) ist eine leistungsstarke Software, die für numerische Berechnungen, Datenanalysen und die Entwicklung von Algorithmen eingesetzt wird. In dieser Arbeit spielt MATLAB eine zentrale Rolle bei der Simulation und Analyse von Sensordaten [12].

Für die in dieser Arbeit durchgeführten Berechnungen und Simulationen wurde MATLAB in der Version R2024a verwendet. In der Anlage C-1 sind die benutzten Toolboxen aufgelistet.

2.3.1 Funktionen und Anwendungen

Zu den Hauptfunktionen von MATLAB gehören die Durchführung mathematischer Berechnungen, die Analyse und Visualisierung von Daten sowie die Entwicklung und Implementierung von Algorithmen. MATLAB bietet zahlreiche Toolboxen, die für verschiedene Anwendungen wie maschinelles Lernen, Robotik, Finanzanalyse oder Biotechnologie entwickelt wurden. Ein wichtiges Erweiterungstool ist Simulink. Simulink dient zur Modellierung, Simulation und Analyse dynamischer Systeme mit Hilfe von Blockdiagrammen. Es bietet entsprechende Bausteine für Übertragungsglieder, wodurch die Erstellung von Modellen vereinfacht wird [13]. In der Steuerungs- und Regelungstechnik wird MATLAB zur Modellierung, Simulation und Implementierung von Regelungssystemen genutzt. Durch die vielen Einsatzmöglichkeiten findet MATLAB breite Anwendung in den mathematisch-naturwissenschaftlichen Disziplinen. [12, 14]

2.3.2 Integration von MATLAB in ecu.test

Mit ecu.test lässt sich MATLAB/Simulink nahtlos in den Testprozess integrieren. Dabei können bestehende Simulink-Modelle auf Parameter aus dem Testfall zugreifen und anschließend simuliert werden. Die gewonnenen Ergebnisse und Variablen kann Simulink an ecu.test zurückübermitteln. Darüber hinaus bietet

ecu.test die Möglichkeit, ein Initialskript in MATLAB auszuführen, das ein Simulink-Modell parametriert und die Simulation startet. Ebenso ist es möglich, ecu.test von MATLAB/Simulink aus fernzusteuern. Eine detaillierte Anleitung dazu findet sich in der ecu.test Dokumentation.

2.3.3 Driving Scenario Designer

Mit dem Driving Scenario Designer (DSD), der Teil der Automated Driving Toolbox in MATLAB ist, lassen sich vielfältige Fahrszenarien abbilden und simulieren. Er dient als Werkzeug für die Entwicklung, Simulation und Validierung von Fahrerassistenzsystemen (SiL/HiL). Von einer Fahrt auf einer normalen Landstraße bis hin zu komplexen Kreuzungen mit Fußgänger- und Fahrradverkehr ist alles darstellbar. Das Versuchsfahrzeug (VUT - Vehicle under Test) kann dabei mit sämtlichen gängigen Sensoren ausgestattet werden, einschließlich Radar, Kamera, Ultraschall, Trägheitsnavigationssystemen (INS-Integrated Navigation System) und LiDAR. Die Aufzeichnung der Sensordaten ermöglicht eine spätere Auswertung in der regulären MATLAB-Umgebung.

Abbildung 2.9 zeigt einen Ausschnitt der Arbeitsumgebung im DSD. Das linke Fenster, das „Scenario Canvas“, bietet einen Überblick über das virtuelle Szenario. Dieses stellt den Straßenverlauf und alle beteiligten Fahrzeuge dar: das blaue VUT, ein rot-brauner LKW, ein grüner PKW und drei gelbe Radfahrer.

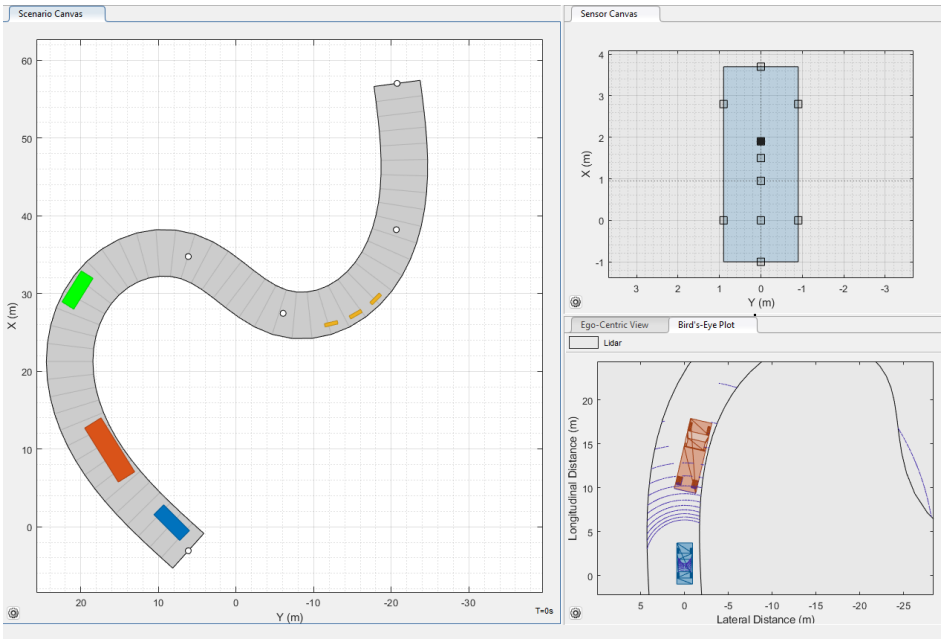


Abbildung 2.9 Arbeitsumgebung im Driving Scenario Designer

Rechts oben im Ausschnitt befindet sich das „Sensor Canvas“, welches eine Vogelperspektive des VUT zeigt, in der die verschiedenen Sensoren auf dem Testfahrzeug platziert werden können. Es gibt neun vorgeschlagene Platzierungspunkte für die Sensoren, der Nutzer kann nach Bedarf auch beliebige andere Punkte am VUT auswählen. In diesem Beispiel wurde ein LiDAR-Sensor im Bereich der Frontscheibe installiert, erkennbar an dem schwarz ausgefüllten Viereck.

Rechts unten kann zwischen einer Ego-Perspektive und einer weiteren Vogelperspektive gewechselt werden. In der dargestellten Ansicht ist die Vogelperspektive ausgewählt, die auch die vom LiDAR-Sensor erfassten Daten durch blaue Linien anzeigt. Es ist wichtig zu erwähnen, dass die vorgestellte Ansicht der Arbeitsumgebung vom Nutzer beliebig angepasst werden kann.

Jeder Sensor kann individuell angepasst und parametriert werden, sodass die Eigenschaften realer Sensoren in der Simulation nachgebildet werden können. Besonders erwähnenswert ist die Integration einer 3D-Umgebung, die es ermöglicht, das erstellte Szenario zusätzlich zur rudimentären Standardansicht darzustellen. [15]

2.4 Light Detection and Ranging

Light Detection and Ranging (LiDAR) ist ein optisches Messverfahren zur Bestimmung von Objektabständen im Raum. LiDAR ähnelt dem Radarverfahren, jedoch kommen statt elektromagnetischen Wellen Ultraviolett-, Infrarot- oder Lichtstrahlen im sichtbaren Bereich zum Einsatz. Der Sensor sendet diese Lichtimpulse aus und misst die Zeit, die benötigt wird, bis die Impulse von Objekten reflektiert und wieder empfangen werden. LiDAR wird in verschiedenen Anwendungen eingesetzt, wie zum Beispiel in der Kartografie, Umweltüberwachung und Fahrzeugen mit automatisierten Fahrfunktionen [16].

2.4.1 Funktionsweise

Im Bereich der Automobilentwicklung wird meist das Prinzip der „Time of Flight“ (ToF) Messung angewendet. Dabei besteht eine Messung aus folgenden Teilschritten:

1. **Lichtimpulse senden**: Das LiDAR-System sendet Lichtimpulse aus. Diese Impulse bestehen aus Lichtstrahlen, die wie unter Abschnitt 2.4 erläutert, in unterschiedlichen Wellenlängenbereichen liegen, abhängig von der spezifischen LiDAR-Technologie.

2. **Reflexion an Objekten**: Die Lichtimpulse treffen auf Objekte in der Umgebung (wie Gebäude, Bäume, Fahrzeuge usw.) und werden von diesen reflektiert.

3. **Empfang der reflektierten Impulse**: Ein Detektor im LiDAR-Sensor misst die Zeit, die vergeht, bis die reflektierten Impulse zurückkommen.

4. **Berechnung der Entfernung**: Indem die Laufzeit der Lichtimpulse gemessen wird, kann die Entfernung zu den Objekten berechnet werden. Die Entfernung entspricht der halben Zeit, die das Licht benötigt, um von der Objektoberfläche zurück zum Sensor zu gelangen.

5. **Erzeugung von 3D-Punktwolken**: Durch wiederholte Messungen aus verschiedenen Blickwinkeln kann das LiDAR-System eine detaillierte 3D-Punktwolke der Umgebung erstellen. Diese Punktwolke enthält genaue räumliche Informationen über die Position und Entfernung der reflektierenden Objekte. [17, 18]

Abbildung 2.10 zeigt eine Gegenüberstellung zwischen einer LiDAR-Punktwolke (oben) und einem Foto (unten), wobei zwei Dummies und ein Verkehrsschild als Referenzobjekte verwendet wurden.

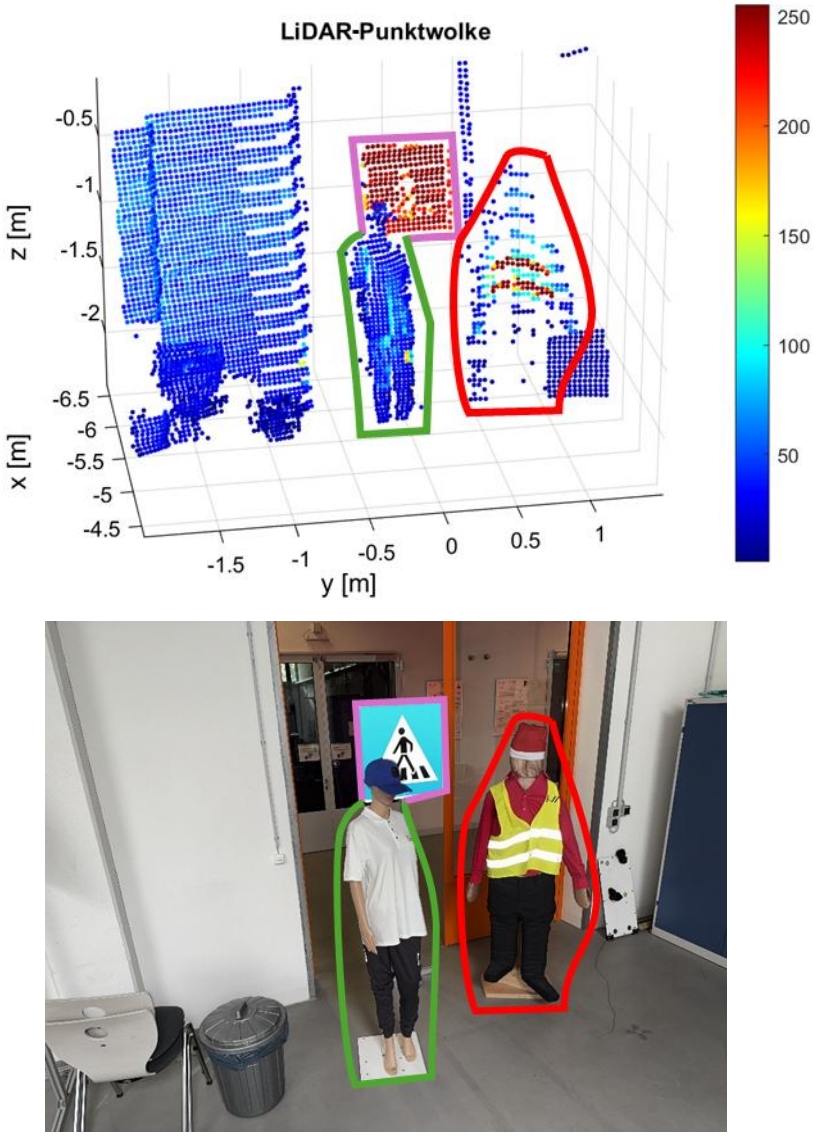


Abbildung 2.10 Gegenüberstellung LiDAR-Punktwolke und Foto

Die beiden Dummies und das Verkehrsschild werden als Ansammlungen von Punkten dargestellt. Jeder Punkt repräsentiert eine durch den LiDAR-Sensor gemessene Reflexion des ausgesendeten Impulses. Die Farbskala an der rechten Seite der Punktwolke zeigt die Intensität der Reflexion (einheitenlos), wobei blaue Punkte für niedrige und rote Punkte für hohe Reflexionswerte stehen. Besonders stark reflektieren, aufgrund der retroreflektierenden Eigenschaften, das Verkehrsschild und die Streifen auf der Warnweste des

rechten Dummies. Die übrige Kleidung, wie die schwarze Jeans und das rote Hemd, reflektieren schwächer bis gar nicht. Im Gegensatz dazu ist die Kleidung des linken Dummies in der Punktwolke besser zu erkennen, was auf unterschiedliche Materialeigenschaften zurückzuführen ist.

Die Entfernung d zu einem Objekt kann anhand von Gleichung (2.1) berechnet werden. Hierbei steht c für die Ausbreitungsgeschwindigkeit der Welle im vorliegenden Medium, und T_{oF} für die Zeitdauer zwischen dem Aussenden und Empfangen des Lichtimpulses. Das Produkt aus der Laufzeit und der Ausbreitungsgeschwindigkeit wird halbiert, da der Lichtimpuls die Strecke zwischen der Messeinrichtung und dem Objekt zweimal zurücklegt [16].

$$d = \frac{c * T_{oF}}{2} \qquad (2.1)$$

Aus der Entfernung d zu jedem gemessenen Punkt kann mithilfe der mathematischen Beziehungen, die in Abbildung 2.11 dargestellt sind, dem Punkt eine x-, y- und z-Koordinate zugewiesen werden. Der Ursprung der Koordinaten liegt dabei im Sensor.

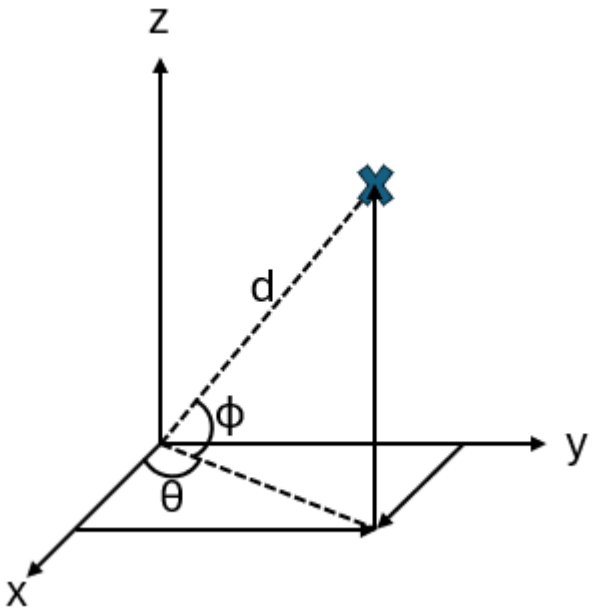


Abbildung 2.11 Zusammenhang zwischen dem gemessenen Abstand und den Punktkoordinaten (in Anlehnung an [19] und [20])

$$x = d * \cos(\Phi) * \sin(\Theta) \quad (2.2)$$

$$y = d * \cos(\Phi) * \cos(\Theta) \quad (2.3)$$

$$z = d * \sin(\Phi) \quad (2.4)$$

Die gezeigten Gleichungen (2.2), (2.3) und (2.4) beschreiben die Umrechnung von sphärischen Koordinaten (Kugelkoordinaten) in kartesische Koordinaten. Dabei ist *d* der radiale Abstand des Punktes vom Ursprung, Φ ist der Polarwinkel (gemessen von der z-Achse) und θ der Azimutalwinkel (gemessen von der x-Achse in der xy-Ebene). Der LiDAR-Sensor kennt diese drei Größen und kann damit die kartesischen Koordinaten des Punktes berechnen.

2.4.2 Grundlegender Aufbau und Einteilung

Ein LiDAR-Sensor ist aus mehreren Hauptkomponenten aufgebaut: dem Sendezweig, dem Empfangszweig und der Recheneinheit.

Der Sendezweig eines LiDAR-Systems besteht aus einer Lichtquelle, die Lichtimpulse erzeugt und diese in Richtung der zu vermessenden Umgebung sendet. Die Wellenlänge des emittierten Lichts liegt dabei im Bereich von 250 nm bis etwa 2 µm. Um ein möglichst breites Sichtfeld zu erfassen, wird der Lichtstrahl bei einem mechanischen System durch einen rotierenden Spiegel oder ein Prisma in verschiedene Richtungen gelenkt. Bei einem Flash-Lidar hingegen werden keine rotierenden Teile benötigt, da ein einzelner Laserblitz ausgesendet wird, der die Umgebung auf einmal erfasst [21–23].

Der Empfänger besteht aus Fotodetektoren, die das reflektierte Licht von den Objekten in der Umgebung erfassen. Ein Fotodetektor ist ein elektronisches Bauelement, das Licht in elektrische Signale umwandelt. Diese Umwandlung erfolgt durch den photoelektrischen Effekt, bei dem Licht auf ein Material trifft und Elektronen freisetzt, was zu einem elektrischen Stromfluss führt [24]. Entscheidend für eine gute Sensorgenauigkeit ist die Empfindlichkeit des Empfängers und dessen Messgeschwindigkeit. Die Zeitmessungselektronik erfasst die Zeitdifferenz zwischen dem Aussenden des Lichtimpulses und dem Empfang des reflektierten Signals. Diese Laufzeit wird verwendet, um die Entfernung zu berechnen [21].

Eine zentrale Recheneinheit verarbeitet die Signale und berechnet die Distanzen basierend auf der gemessenen Laufzeit. Die Daten werden oft in eine Punktwolke umgewandelt, die die dreidimensionale Struktur der Umgebung darstellt. LiDAR-Sensoren benötigen eine Stromversorgung für die Elektronik und oft auch ein Kühlsystem, um die Wärme, die bei der Laseremission und in der Elektronik entsteht, abzuleiten. [18, 25]

Der grundlegende Aufbau wird in Abbildung 2.12 anhand eines LiDAR, basierend auf der MEMS (mikro-elektromechanische Systeme)-Technologie veranschaulicht. Der Sendezweig besteht dabei aus den Laserdioden, der Optik und des MEMS-Scanners. Fotodetektor und Optik bilden den Empfangszweig.

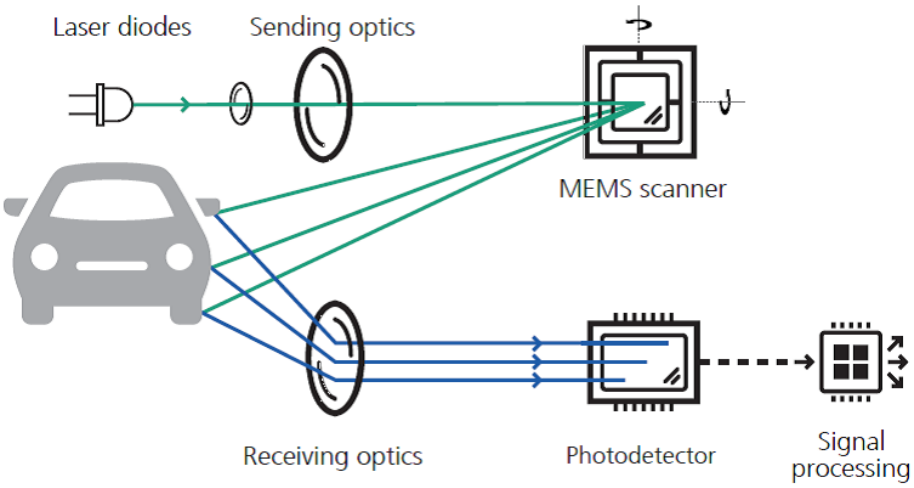


Abbildung 2.12 Aufbau eines LiDAR mit MEMS-Technologie [26]

LiDAR Sensoren können nach verschiedene Kategorien unterteilt werden:

Basierend auf der Wellenlänge des Lasers

- Grün-LiDAR
- Nahinfrarot LiDAR

Basierend auf der Anwendung

- Topografische LiDAR
- Bathymetrische LiDAR

Basierend auf der Bauweise

- Mechanische LiDAR
- Solid-State LiDAR

Wang, Watkins und Xie unterteilen die LiDAR Sensoren in ihrem Paper, wie in Abbildung 2.13 dargestellt, anhand der Abtastmethode.

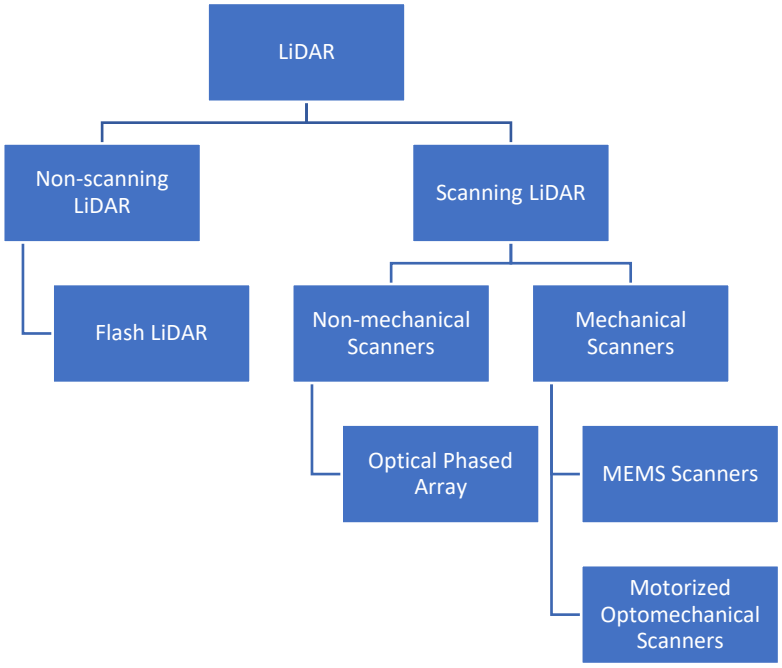


Abbildung 2.13 Überblick LiDAR-Systeme in Anlehnung an [27]

Scanning-LiDAR Systeme erfassen das Umfeld mittels vieler einzelner, gepulster Laserstrahlen. Diese Strahlen werden durch eine Ablenkoptik, wie rotierende Spiegel oder mikro-elektromechanische Systeme (MEMS) bei denen die Spiegel elektromagnetisch bewegt werden, über das zu erfassende Bildfeld geleitet. Nachteile dieser Sensoren sind die hohen Herstellungskosten, der Verschleiß aufgrund der beweglichen mechanischen Komponenten und die Größe der Bauweise. Neben einer Ablenkoptik existiert die Möglichkeit der Nutzung eines phasengesteuerten Scanning LiDAR (Optical Phased Array - OPA). Dabei wird die Phasenverschiebung des ausgesendeten und reflektierten Lasers genutzt, um die Entfernung zu berechnen [21, 27].

Non-Scanning LiDAR Systeme, auch Flash LiDAR genannt, verwenden eine Laserquelle, die das Sichtfeld vollständig beleuchtet. Ähnlich einer Kamera mit Blitz. Das reflektierte Licht wird von einem Array aus Fotodetektoren erfasst. Dank des Verzichts auf bewegliche Teile sind diese Sensoren kompakter und robuster als Scanning-LiDAR Systeme [27].

2.5 Simultane Positionsbestimmung und Kartierung

Simultane Positionsbestimmung und Kartierung (SLAM, englisch Simultaneous Localization and Mapping) hat seinen Ursprung in der Roboterindustrie der 1980er und 1990er Jahre. SLAM ermöglicht einem autonomen Fahrzeug, sich in einer unbekannten Umgebung zu orientieren und gleichzeitig eine Karte dieser Umgebung zu erstellen. Da sich das Fahrzeug oft in dynamischen und komplexen Umgebungen wie dem Stadtverkehr bewegen muss, in welchem herkömmliche GPS-basierte Systeme allein nicht ausreichen, um das Fahrzeug zentimetergenau zu orten, liefert SLAM einen zusätzlichen Ansatz für die Orientierung. Zur Kartierung und Lokalisierung können verschiedene Sensoren wie Kameras oder LiDAR-Sensoren eingesetzt werden. [28, 29].

MATLAB stellt mit der Robotics System Toolbox und der Computer Vision Toolbox wichtige Werkzeuge für die Implementierung von SLAM bereit. Folgend soll kurz allgemein auf die LiDAR-SLAM-Methode eingegangen werden.

Bei der Anwendung von SLAM wird die Bewegung des Fahrzeugs oder Sensors schrittweise geschätzt, indem aufeinanderfolgende LiDAR-Punktwolken miteinander registriert werden. Registrierung ist der Prozess, bei dem zwei oder mehr Punktwolken so ausgerichtet werden, dass sie in einem gemeinsamen Koordinatensystem übereinstimmen. Ziel ist es, die Übereinstimmung zwischen den Punktwolken zu maximieren, was bedeutet, dass Punkte, die dieselben physischen Orte in der realen Welt repräsentieren, auch in der registrierten Punktwolke übereinander liegen.

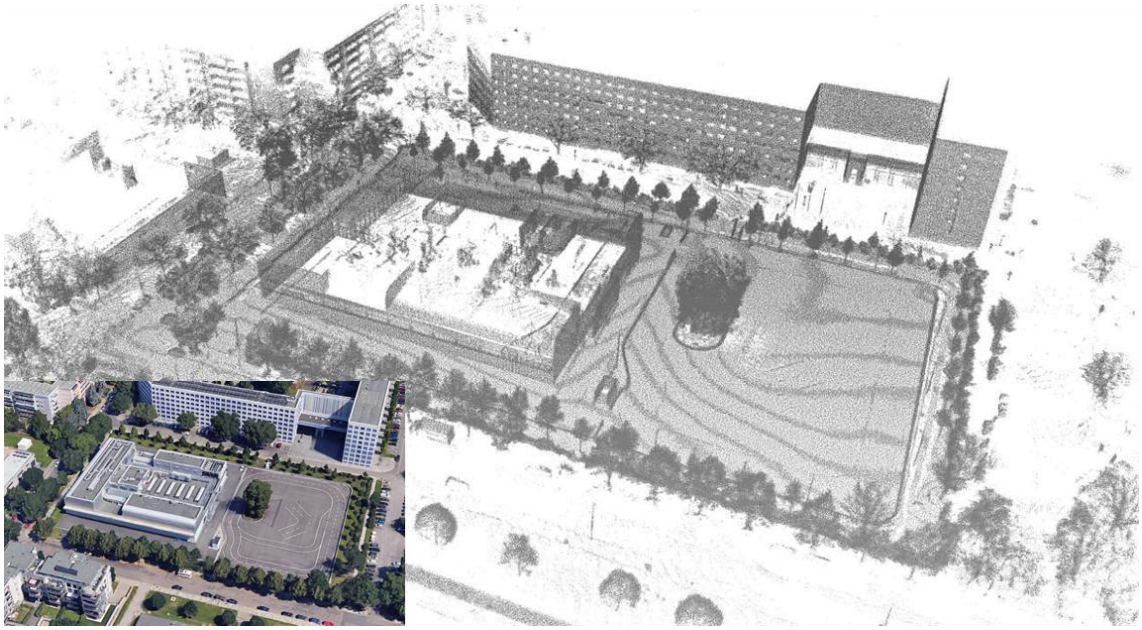


Abbildung 2.14 Beispiel einer SLAM-Punktwolke [30]

In Abbildung 2.14 ist beispielhaft eine mit dem SLAM-Verfahren zusammengesetzte Punktwolke dargestellt. Kartiert wurde das Gelände des Fahrzeugtechnikums der HTW Dresden. Zum Vergleich ist links unten eine Satellitenaufnahme der Umgebung abgebildet.

Um die relative Transformation zwischen den Punktwolken zu berechnen, stehen verschiedene Registrierungsalgorithmen zur Verfügung. Ein gängiger Ansatz ist der Iterative Closest Point (ICP)-Algorithmus, der darauf abzielt, die beste Übereinstimmung zwischen zwei Punktwolken durch iterative Anpassung zu finden. Die iterative Anpassung im ICP-Algorithmus beginnt mit einer groben Schätzung der Transformation zwischen zwei Punktwolken, einer festen und einer beweglichen Punktwolke. Der Algorithmus sucht für jeden Punkt der beweglichen Punktwolke den nächstgelegenen Punkt in der festen Punktwolke. Auf Basis dieser Paare berechnet er eine Transformation, die die bewegliche Punktwolke so ausrichtet, dass die Abstände zwischen den entsprechenden Punkten minimiert werden. Diese Transformation wird auf die bewegliche Punktwolke angewendet, und der Vorgang wiederholt sich, bis die Änderungen zwischen den Iterationen minimal sind. Schließlich endet der Algorithmus, wenn die Punktwolken optimal ausgerichtet sind und keine weiteren signifikanten Verbesserungen erzielt werden können. Diese Metrik des ICP-Algorithmus nennt

sich Punkt-zu-Punkt-Metrik und ist eine der grundlegenden Methoden zur Registrierung von Punktwolken.

Die Punkt-zu-Fläche-Metrik geht über die Betrachtung der Abstände zwischen einzelnen Punkten hinaus, indem sie auch die Ausrichtung der Flächen berücksichtigt, die durch diese Punkte gebildet werden. Anstatt nur den direkten Abstand zwischen zwei Punkten zu minimieren, versucht der Algorithmus hier, den Abstand eines Punktes zu der Tangentialebene zu minimieren, die durch einen entsprechenden Punkt in der festen Punktwolke verläuft. Dies kann zu einer schnelleren und stabileren Anpassung führen, insbesondere wenn die Punktwolken komplexe Oberflächenstrukturen aufweisen. [31–33]

2.6 Messtechnik

2.6.1 Ouster OS1-64 LiDAR

Der Ouster OS1-64 LiDAR-Sensor, der in dieser Arbeit verwendet wird, ist ein mobiler LiDAR-Sensor, der sich für Anwendungen im Bereich der automatisierten Fahrfunktionen eignet. Er bietet ein 360°-Sichtfeld und nutzt dabei ein rotierendes Messverfahren. Auf 64 vertikalen Ebenen wird die Umgebung erfasst. Jede dieser Ebenen repräsentiert eine horizontale Linie im Sichtfeld des Sensors. Innerhalb jeder dieser Ebenen kann der Sensor eine Auflösung von 512, 1024 oder 2048 Messpunkten erreichen. Die Rotationsfrequenz des Sensors beträgt entweder 10 Hz oder 20 Hz. Dies ermöglicht die Erfassung von 1.310.720 Punkten pro Sekunde. Der OS1-64 hat eine Reichweite von bis zu 110 m bei einer Wellenlänge von 855 nm und einer Reflexionsrate von 80 %. Unter diesen Bedingungen erreicht er eine Detektionswahrscheinlichkeit von 90 % [34]. In Abbildung 2.15 ist das Koordinatensystem des Ouster dargestellt. Detailliertere Informationen sind aus der Betriebsanleitung zu entnehmen [35].

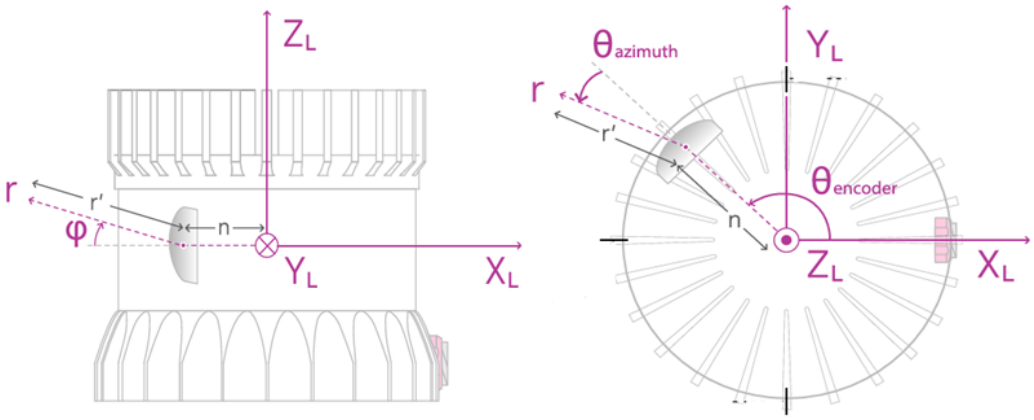


Abbildung 2.15 Ouster OS1-64 Koordinatensystem [35]

Tabelle 2.1 Beschriftung Abbildung 2.15 Ouster OS1-64 Koordinatensystem [35]

Zeichen	***Bedeutung***
X_L, Y_L, Z_L	Koordinatenachsen
φ	Vertikaler Winkel
r	Gemessener Abstand
n	Abstand Laserursprung und Koordinatensystem
$\theta_{azimuth}$	Azimut
$\theta_{encoder}$	Encoder-Winkel

Das Ouster Software Development Kit (SDK) für Python unterstützt die Integration des Sensors in das Programm ecu.test. Das SDK bietet umfangreiche Funktionen zur Sensorkonfiguration, Datenaufzeichnung im PCAP-Format, Extraktion von Messdaten wie Entfernungs-, Signal-, Reflexions- und Near-IR-Bildern (Nahinfrarot) und zur effizienten Erzeugung kartesischer Koordinaten aus Entfernungswerten. Es ermöglicht auch die Visualisierung von Live- und aufgezeichneten LiDAR-Daten sowie den framebasierten Zugriff auf diese Daten. Innerhalb der Master-Thesis von Schrocke wurde die Ouster SDK bereits erfolgreich in ecu.test integriert [10].

2.6.2 Vector VN1630

Der Vector VN1630 ist ein Interface-Gerät von Vector Informatik, das entwickelt wurde, um eine Verbindung zu verschiedenen Fahrzeugnetzwerken wie CAN, CAN FD, CAN XL und LIN herzustellen und zu verwalten. Mit einer Vector Box können Entwickler und Ingenieure direkt auf Fahrzeugnetzwerke zugreifen, um

Daten zu überwachen, zu analysieren und zu diagnostizieren. Um ausführliche Busanalysen durchzuführen, erfasst die Vector Box die Netzwerkkommunikation und führt Echtzeitanalysen durch. Sie unterstützt auch Restbussimulationen, bei denen ein Teil des Fahrzeugnetzwerks simuliert wird, um die Funktionalität von Steuergeräten zu testen. [36, 37]

Die Vector Box kann nahtlos in die Testumgebung ecu.test integriert werden, um Daten von Fahrzeugnetzwerken wie dem CAN zu lesen und zu verwenden. Diese Integration ermöglicht es, Echtzeitdaten von CAN-Bussen und anderen unterstützten Netzwerken zu erfassen und in den Testabläufen zu verwenden.

2.7 Einparkassistenz

Unter Einparkassistenz versteht sich eine Fahrerassistenzfunktion, die das Einparken eines Fahrzeugs vereinfacht, indem sie den Fahrer beim Einparken unterstützt oder das Fahrzeug sogar vollständig automatisiert – unter Überwachung des Fahrers – in eine Parklücke steuert. Dabei werden Kombinationen von verschiedenen Sensoren genutzt, darunter Ultraschallsensoren, Kameras und Radarsensoren. Die Sensoren messen den Abstand zu Objekten in der Umgebung des Fahrzeugs und analysieren die Größe und Lage der Parklücke.

Generell lässt sich die Einparkassistenz in rein informierende Systeme und geführte Systeme unterscheiden. Die informierenden Systeme stellen dem Fahrer Informationen zu Objekten bereit, die sich in der Trajektorie befinden. Sie können aber auch die Fahrzeugumgebung durch Kamerabilder darstellen. Näher erläutert werden in den folgenden Absätzen die geführten Systeme, bei denen der Fahrer die Querführung abgibt oder das Fahrzeug komplett automatisiert einparkt.

Der in Abbildung 2.16 gezeigte Parkassistent eines Aiways U5 führt einen vollautomatischen Parkvorgang in eine Querparklücke durch. Die Darstellung zeigt die Kombination aus informierendem und geführtem System. Links im Display wird der Fortschritt des Einparkvorgangs visualisiert, rechts die Rückfahrkamera mit Hilfslinien für die Trajektorie des Fahrzeuges und rechts unten als Ergänzung dazu eine 360° Ansicht der Umgebung.

Abbildung 2.16 Ansicht des Parkassistenten eines Aiways U5 [38]

Die geführten Systeme nutzen meist Ultraschallsensoren, um während des Vorbeifahrens an Quer- oder Längsparklücken deren Abmessungen zu erfassen. Anschließend wird überprüft, ob die Parklücke ausreichend breit für einen Einparkvorgang ist. Fahrzeughersteller geben dabei bestimmte Kriterien vor, die erfüllt sein müssen, damit das Fahrzeug die Parklücke erfolgreich erkennt. Volkswagen gibt bei Parklücken parallel zur Fahrbahn an, dass diese bis zu einer Geschwindigkeit von ungefähr 40 km/h erkannt werden, vorausgesetzt die Parklücke misst in ihrer Länge mehr als Fahrzeuglänge + 0,8 m. Bis etwa 20 km/h werden Querparklücken erkannt, wenn diese eine Breite von mindestens 0,8 m + Fahrzeugbreite aufweisen. In beiden Fällen ist beim Vorbeifahren ein Abstand von 0,5 bis 2 Metern zur Parklücke einzuhalten [39].

Bei semiautomatischen Systemen, die nur die Querführung übernehmen, erhält der Fahrer eine Benachrichtigung, dass der Parkvorgang gestartet werden kann, sobald eine geeignete Parklücke erkannt wird. Zu diesem Zeitpunkt hat das Fahrzeug bereits die Trajektorie des Parkvorganges berechnet. Dafür wird eine Kombination aus Odometrie (Verfahren zur Positionsbestimmung durch Messung der zurückgelegten Strecke), Lenkwinkel, Gierrate und Beschleunigungen genutzt. Der Fahrer muss lediglich die Anweisungen auf dem Display des Fahrzeugs befolgen, was beim semiautomatischen Einparken in der Regel das Einlegen des Rückwärtsgangs, Gangwechsel und das Bedienen der Bremse und des Fahrpedals umfasst. Der Parklenkassistent übernimmt dabei die Steuerung des Lenkrads, um das Fahrzeug präzise in die Parklücke zu

manövrieren. Durch eine elektromechanische Servolenkung oder eine konventionelle Lenkung mit Elektromotor kann die automatische Querführung realisiert werden. [40, 41, 39]

Moderne Fahrzeuge sind in der Lage den Parkvorgang vollautomatisiert durchzuführen, der Fahrer muss lediglich überwachen und in Notfällen eingreifen. Darüber hinaus bieten diese Systeme auch die Option an, bei einem vom Fahrer selbst gestarteten Parkvorgang mit einzugreifen, sollte der Fahrer Probleme beim Einparken haben. Eine Ausparkunterstützung wird ebenfalls angeboten [42].

Der Parklenkassistent bietet zahlreiche Vorteile, darunter die Reduzierung von Stress und die Minimierung des Risikos von Parkunfällen. Er ist besonders hilfreich für Fahrer, die Schwierigkeiten beim Einparken haben oder in beengten städtischen Umgebungen unterwegs sind.

3 Erweiterung des Testfalls „Parking-Assistant“

In diesem Kapitel wird der gesamte Entwicklungsprozess der Erweiterung des „Parking Assistant“ ausführlich beschrieben – von der Erstellung einer Simulationsumgebung über die Durchführung realer Versuche bis hin zur finalen Integration in die Testplattform ecu.test.

3.1 Bestehender Testfall „Parking-Assistant“

Eine entscheidende Rolle auf dem Weg zum automatisierten Fahren spielt der Parklenkassistent. Da das Parken zu den grundlegenden Fahrmanövern gehört und typischerweise das Ende jeder Fahrt darstellt, muss ein automatisiertes System in der Lage sein, eine geeignete Parklücke zu identifizieren und den Parkvorgang sicher durchzuführen. Dafür sind umfangreiche Validierungen und Tests erforderlich, um die Zuverlässigkeit und Sicherheit des Systems unter verschiedenen Bedingungen zu gewährleisten. Ein beispielhaftes Szenario ist ein vollautomatisiertes Fahrzeug, das die Insassen vor einem Zielort absetzt und anschließend selbstständig einen Parkplatz sucht und sicher einparkt – ohne jegliches menschliche Eingreifen.

Der bereits existierende Ansatz des Testfalls in der Testautomatisierungs-software ecu.test sieht vor, dass während der Vorbeifahrt an der Parklücke der Prüfingenieur beobachtet, ob die Parklücke vom Versuchsfahrzeug erkannt wird. Diese Information wird für den späteren Bericht dokumentiert. Nach erfolgreicher Erkennung der Parklücke durch das Versuchsfahrzeug initiiert der Testfahrer den automatisierten Parkvorgang des Fahrzeugs. Abbildung 3.1 zeigt das Konzept, wobei die bisherige Einschränkung besteht, dass nur Querparklücken auf der rechten Fahrzeugseite berücksichtigt werden. In diese Parklücken wird das Fahrzeug rückwärts eingeparkt.

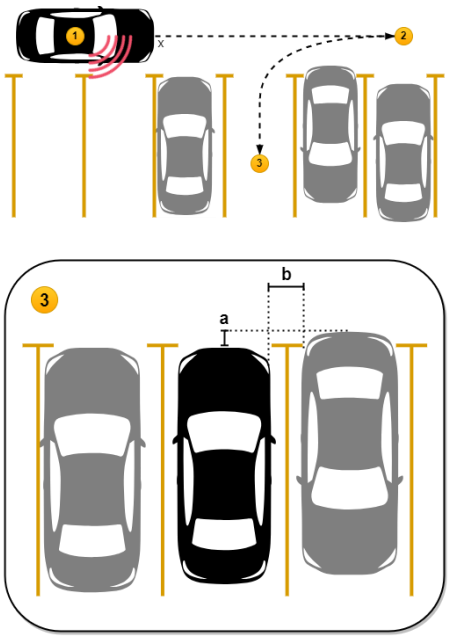


Abbildung 3.1 Konzept des Testfalls „Parking-Assistant“ [43]

Im Verlauf des gesamten Tests wird der Parkplatz aktuell an drei charakteristischen Punkten entlang der Fahrtrajektorie mithilfe eines LiDAR vermessen. Diese Messungen sollen zur Bewertung der Genauigkeit und Effizienz des Parkassistenten dienen. Die Auswertung der Punktwolken und Bewertung der Parkposition ist bisher nicht realisiert worden. An diesem Punkt setzt die vorliegende Diplomarbeit an, indem die notwendigen Verfahren zur Analyse der Punktwolken und zur Bewertung der Parkposition entwickelt und implementiert werden.

3.2 Konzept und Vorgehen

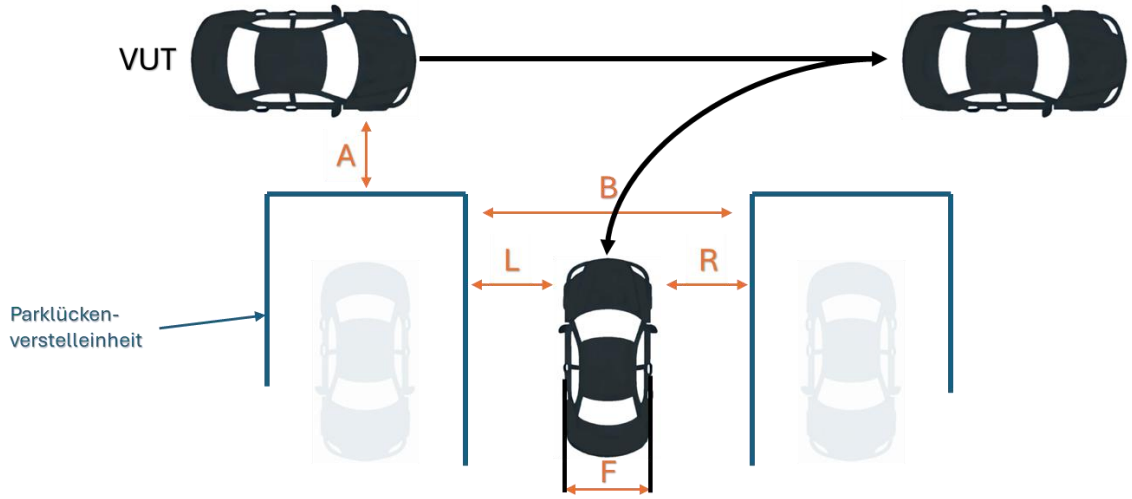
Das Ziel dieser Erweiterung besteht darin, die finale Parkposition des Fahrzeugs zu evaluieren, um zu prüfen, ob der Parklenkassistent das Fahrzeug zentriert oder versetzt in der Parklücke positioniert hat.

Im bisherigen Testfall wird die Parklücke durch den LiDAR-Sensor an drei Punkten erfasst, was im Hinblick auf die im folgenden Konzept erläuterte Methodik zur Bewertung der Parkposition unzureichend ist. Daher sollen während der Vorbeifahrt an der Parklücke kontinuierlich LiDAR-Punktwolken aufgenommen werden. Mithilfe dieser Punktwolken wird die Parklücke dreidimensional kartiert, um daraus die Breite der Lücke bestimmen zu können.

Nach Abschluss des automatisierten Parkvorgangs wird der rechte Abstand (R) des Testfahrzeugs zum benachbarten Fahrzeug im Bereich des Kotflügels ermittelt. Anhand der Parklückenbreite (B), der Fahrzeugbreite ohne Außenspiegel (F) und des rechten Abstands (R) kann der linke Abstand (L) zum benachbarten Fahrzeug berechnet werden:

$$L = B - F - R \tag{3.1}$$

Abbildung 3.2 veranschaulicht dieses Konzept.



A Abstand des VUT während der Vorbeifahrt
B Breite der Parklücke
L Abstand zum linken benachbarten Fahrzeug
R Abstand zum rechten benachbarten Fahrzeug
F Breite des VUT (ohne Außenspiegel)

Abbildung 3.2 Konzeptzeichnung (Fahrzeuggrafik mit KI generiert)

Sind der linke und der rechte seitliche Abstand zu den benachbarten Fahrzeugen bekannt, kann eine Aussage darüber getroffen werden, ob der Parklenkassistent das VUT zentriert oder asymmetrisch in der Parklücke positioniert hat.

Voraussetzung für eine präzise Berechnung der Position ist, dass die benachbarten Fahrzeuge möglichst parallel zueinander ausgerichtet sind. Hierfür kommen Parklückenverstelleinheiten (PLVs) zum Einsatz, die eine genaue Positionierung und Ausrichtung ermöglichen.

In der gesamten Arbeit wird die Fahrzeugbreite ohne Außenspiegel verwendet, um eine standardisierte und konsistente Basis für die Berechnungen zu

gewährleisten. Außenspiegel variieren in der Größe und können eingeklappt werden, was zu Abweichungen führen könnte. Daher definieren die Kotflügel die Breite des VUT und bieten eine zuverlässige Möglichkeit, die Abstände zwischen den PLVs und dem Fahrzeug für spätere Kontrollen des Algorithmus auch manuell zu vermessen.

Darüber hinaus werden in dem Testfall weiterhin nur Querparklücken betrachtet, in die das Fahrzeug rückwärts einparken soll. Zusätzlich wird der Abstand vermessen, mit dem das Versuchsfahrzeug an der Parklücke vorbeifährt (A).

Der erste Schritt besteht darin, eine Simulationsumgebung aufzubauen, die den Testfall möglichst genau widerspiegelt, die benötigten Sensordaten generiert und zur Bearbeitung bereitstellt. Auf Basis der LiDAR-Daten wird anschließend ein Algorithmus entwickelt, der sowohl die Parklücke kartiert, als auch die seitlichen Abstände bestimmt. Im nächsten Schritt wird der Algorithmus mit realen Daten getestet und in den bestehenden Testfall in ecu.test integriert.

3.3 Simulationsumgebung in MATLAB

3.3.1 Simulationsumgebung des „Parking-Assistant“

Ein effektiver Ansatz zur Lösung einer derartigen Problemstellung besteht darin, das Szenario zunächst in einer Simulation abzubilden. MATLAB bietet mit der Automated Driving Toolbox ein leistungsfähiges Werkzeug, das hilfreiche Tools und Algorithmen für die Entwicklung, Simulation und das Testen von Fahrerassistenzsystemen bereitstellt. Eines dieser Werkzeuge ist der Driving Scenario Designer. Dieser ist in Abschnitt 2.3.3 bereits näher erläutert worden.

Die Simulationsumgebung für den „Parking-Assistant“ wurde auf das Notwendigste beschränkt. Grundvoraussetzung ist die Integration einer Straße, da der DSD ohne diese keine Simulation zulässt. Weiterhin erforderlich sind ein Versuchsfahrzeug mit LiDAR-Sensorik und einer vorgegebenen Trajektorie, sowie zwei benachbarte Fahrzeuge, zwischen denen sich die Parklücke befindet. Abbildung 3.3 veranschaulicht das Szenario im DSD.

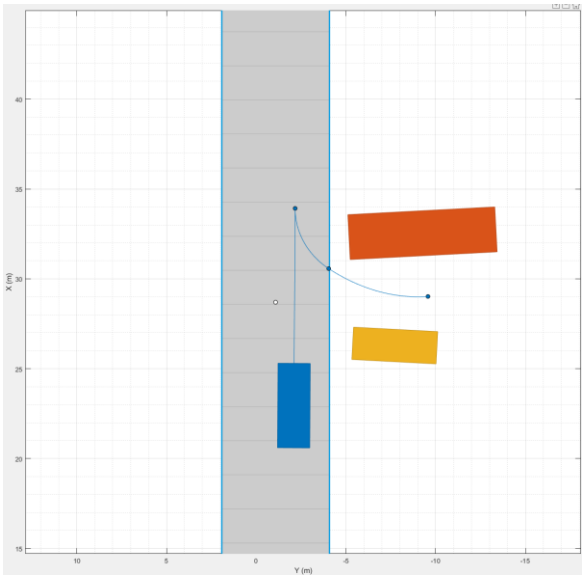


Abbildung 3.3 Aufbau des Szenarios

Das gelbe Rechteck simuliert dabei einen PKW. Für ein heterogenes Szenario ist als zweites Fahrzeug ein LKW (orangenes Rechteck) ausgewählt worden. Obwohl im realen Testfall die benachbarten Fahrzeuge möglichst parallel zueinander stehen – eine Voraussetzung für die genaue Berechnung der Position des Testfahrzeugs innerhalb der Parklücke – wurden sie in der Simulation bewusst schräg positioniert. Dies liegt daran, dass zum Zeitpunkt der Erstellung der Simulation ein anderer Ansatz zur Berechnung der seitlichen Abstände verfolgt wurde. Dieser Ansatz wurde später zwar verworfen, ist aber dennoch erwähnenswert, um den Entwicklungsprozess und die getroffenen Entscheidungen nachvollziehbar zu machen.

Nach der Erstellung des Grundszenarios wird die Sensorik implementiert. Im realen Testverfahren ist der LiDAR-Sensor durch Saugnäpfe rechts oben an der Windschutzscheibe montiert (vom Fahrer aus betrachtet, Abbildung 3.5). Diese Position muss in der Simulation entsprechend beibehalten werden, wie in Abbildung 3.4 dargestellt.

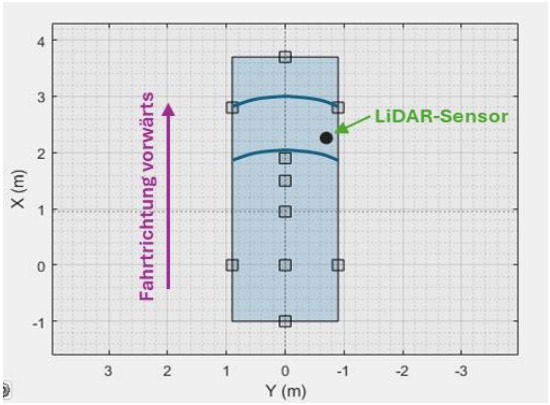


Abbildung 3.4 Platzierung des LiDAR-Sensors in der Simulation

Abbildung 3.5 Platzierung des LiDAR-Sensors am realen VUT

Hintergrund dieser ungewöhnlichen Sensorposition ist der Gedanke, dass die Konturen der Parklückenverstelleinheiten im realen Versuch besser vom LiDAR-Sensor erfasst werden können.

Bei der Implementierung des Sensors in die Testumgebung muss neben der Platzierung auch der Neigungswinkel der Windschutzscheibe berücksichtigt werden, da dieser sich ebenfalls auf den Sensor überträgt. Dieser Winkel variiert je nach Fahrzeugtyp und kann sich auch entlang der Windschutzscheibe verändern. In der Regel ist der Winkel zum Dach hin flacher als zur Motorhaube. Im Rahmen der Simulation wurde das spätere reale VUT, ein VW Passat, als Referenz verwendet. Dessen Windschutzscheibe weist im Bereich der Sensormontierung einen Längswinkel von ca. 26° und einen Querwinkel von ca. 8° auf.

Zusätzlich wird der simulierte LiDAR-Sensor so parametriert, dass er dem real verwendeten Ouster OS1 LiDAR gleicht. Dabei wird sich am Datenblatt des Sensors orientiert [34]. Die Parameter Reichweite, Reichweitengenauigkeit, maximale Aufnahmefrequenz sowie der horizontale und vertikale Sichtbereich wurden erfolgreich aus dem Datenblatt in die Simulation übernommen.

Nach Erstellung der virtuellen Umgebung wird der Einparkvorgang simuliert. Die vom DSD erfassten Daten können am Ende der Simulation als *.mat*-Datei in den MATLAB-Workspace importiert und dort weiterverarbeitet werden.

In der *.mat*-Datei stehen die Punktwolken sowie Informationen über die Position, Geschwindigkeit und Ausrichtung der Akteure – also der an der Simulation beteiligten Fahrzeuge, Personen und Radfahrer – zur Verfügung, wobei Letztere in diesem Fall nicht vorkommen.

3.3.2 Kartierung der Parklücke

Die erste Teilaufgabe besteht in der Kartierung der Parklücke. Hierzu sollen die einzelnen LiDAR-Aufnahmen zu einer dreidimensionalen Abbildung der Parklückensituation zusammengesetzt werden. Für ein korrektes Zusammenfügen jeder LiDAR-Aufnahme sind die exakte Position und Ausrichtung des Versuchsfahrzeugs zum jeweiligen Aufnahmezeitpunkt erforderlich. Der Datensatz, der nach der Simulation im Driving Scenario Designer bereitgestellt wird, enthält zu jeder LiDAR-Aufnahme die Position und Orientierung des VUT.

In der ersten realen Umsetzung soll die Positions- und Ausrichtungsbestimmung mittels Dead Reckoning erfolgen, einem Verfahren, das in diesem Anwendungsfall zur Lokalisierung auf die Raddrehzahlen zurückgreift. Da der DSD jedoch keine Räder und deren Drehungen simulieren kann, wird diese Thematik erst in Kapitel 3.5 erläutert.

MATLAB stellt mit dem Befehl *pcmerge* eine Funktion zur Verfügung, um zwei Punktwolken zu einer einzigen zu kombinieren. Dies ist besonders nützlich, wenn mehrere Punktwolken aus verschiedenen Blickwinkeln oder zu verschiedenen Zeitpunkten vorliegen und zu einer einzigen Punktwolke zusammengeführt werden sollen. Dabei werden Punkte die innerhalb einer vom Benutzer angegeben Distanz liegen zu einem Punkt zusammengefasst. Wichtig ist, dass beide Punktwolken im selben Koordinatensystem liegen. Ist dies nicht der Fall muss als Vorbereitung eine Koordinatentransformation erfolgen [44].

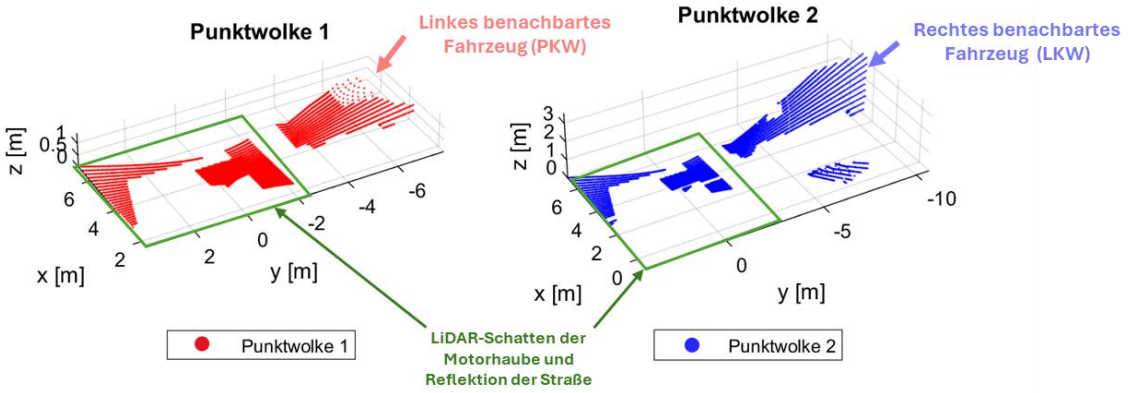


Abbildung 3.6 Zwei Punktwolken die zusammengeführt werden sollen

Abbildung 3.6 zeigt beispielhaft zwei Punktwolken, die zu unterschiedlichen Zeitpunkten aus der Simulation im DSD aufgenommen wurden.

Beide Punktwolken werden nun mittels *pcmerge* zusammengeführt, um mit der Kartierung der Parklücke zu beginnen.

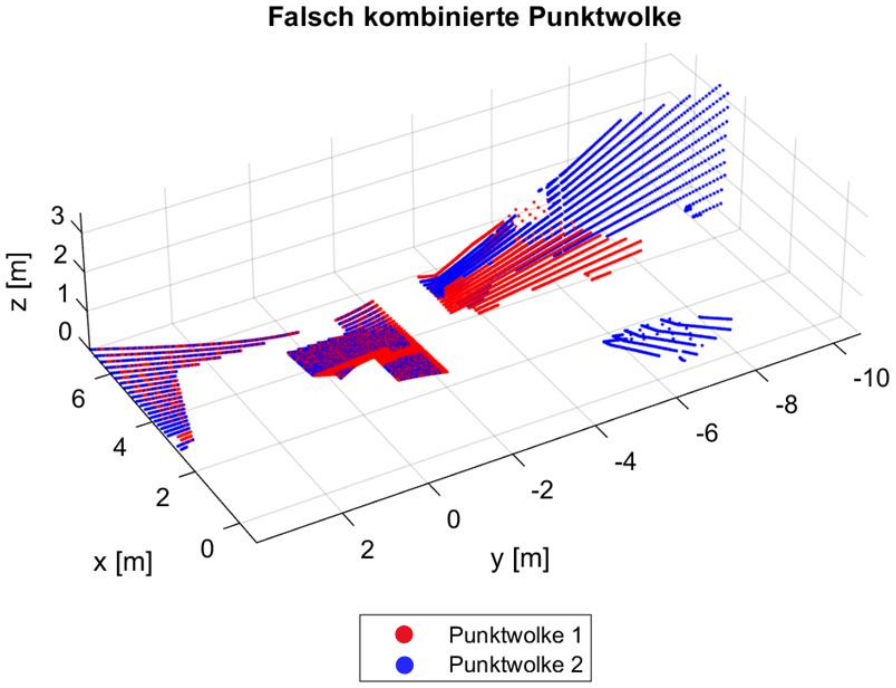


Abbildung 3.7 Fehlerhaft zusammengeführte Punktwolke

Die in Abbildung 3.7 dargestellte, kombinierte Punktwolke weist jedoch Fehler auf, da der PKW fälschlicherweise mitten im LKW positioniert ist, wodurch keine Parklücke zwischen den beiden Fahrzeugen sichtbar wird. Diese Ungenauigkeit resultiert daraus, dass die Bewegungen des Versuchsfahrzeugs bisher nicht korrekt berücksichtigt wurden.

Während des Messvorgangs bewegt sich das VUT durch den Raum, was sowohl eine Translation (Positionsänderung) als auch eine Rotation (Änderung der Orientierung) umfasst. Werden diese Bewegungen nicht präzise einbezogen, entstehen Ungenauigkeiten beim Zusammensetzen der durch den LiDAR-Sensor erfassten Punktwolken.

Der am Fahrzeug montierte LiDAR-Sensor misst die Umgebung aus seiner eigenen Position und Orientierung heraus. Da der Sensor jedoch am VUT befestigt ist, bewegt er sich ebenfalls mit dem Fahrzeug. Um eine korrekt zusammengesetzte Punktwolke zu generieren, müssen die Bewegungen des Fahrzeugs – und somit die des Sensors – in das globale Koordinatensystem übertragen werden. Jede einzelne Messung des Sensors muss entsprechend der aktuellen Position und Ausrichtung des VUT korrigiert werden. Hierbei werden drei Koordinatensysteme verwendet:

Sensorkoordinatensystem: Dies ist das lokale Koordinatensystem des LiDAR-Sensors. Die Punktwolken werden in diesem Koordinatensystem erfasst, wobei der Sensor selbst den Ursprung bildet.

Fahrzeugkoordinatensystem: Das Fahrzeug verfügt über ein eigenes Koordinatensystem. Da der Sensor sich mit dem Fahrzeug bewegt, ist seine Position relativ zum Fahrzeugkoordinatensystem von Bedeutung. Im DSD liegt der Ursprung im Mittelpunkt der Hinterachse, was nicht dem Ursprung gemäß ISO 8855 entspricht, da dieser auf den Schwerpunkt des Fahrzeugs festgelegt ist.

Globales Koordinatensystem: Das Globales Koordinatensystem ist das übergeordnete Koordinatensystem, in dem die Umgebung insgesamt beschrieben wird. Um die korrekten Positionen der Objekte in der Punktwolke zu bestimmen, müssen die Daten vom Sensor- und Fahrzeugkoordinatensystem in dieses globale System übertragen werden. Der Ursprung kann vom Anwender festgelegt werden, es bietet sich aber an den Startpunkt einer Messung oder einen markanten Punkt in der Karte zu wählen.

Für ein besseres Verständnis stellt Abbildung 3.8 vereinfacht die Beziehung zwischen den unterschiedlichen Koordinatensystemen dar. Dabei wurde sich auf die xy-Ebene beschränkt und die unterschiedlichen Rotationen vernachlässigt.

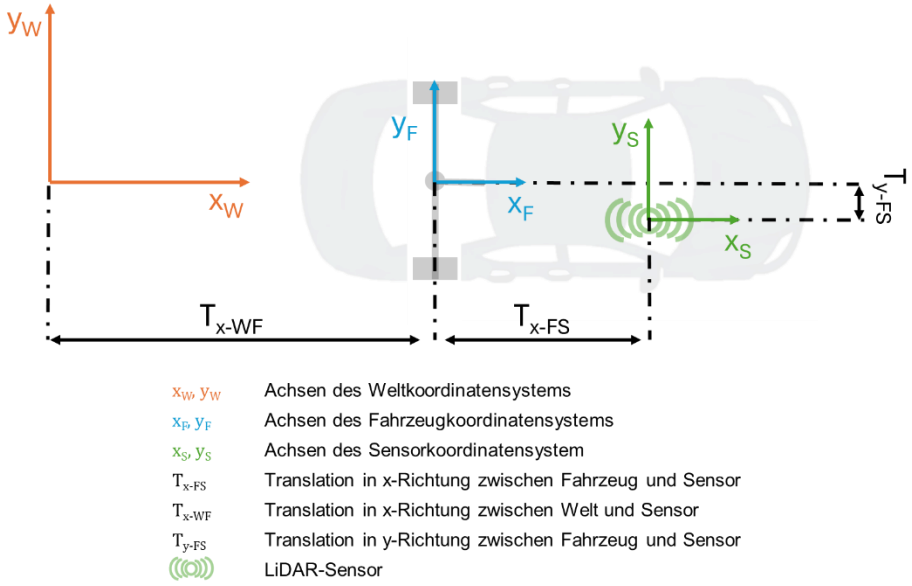


Abbildung 3.8 Vereinfachte Darstellung der drei Koordinatensysteme (Fahrzeuggrafik KI generiert)

Die Punktwolken müssen also zunächst vom Sensorkoordinatensystem in das Fahrzeugkoordinatensystem und anschließend in das globale Koordinatensystem überführt werden. Für diese Transformationen werden sogenannte Rotationsmatrizen und Translationsvektoren eingesetzt.

Eine Rotationsmatrix ist ein mathematisches Werkzeug, das dazu dient, Punkte im Raum zu drehen. Sie ist besonders nützlich in der Robotik, Computeranimation und anderen Bereichen, in denen die Orientierung und Position von Objekten im dreidimensionalen Raum wichtig sind. In drei Dimensionen gibt es drei grundlegende Rotationen um die x-, y- und z-Achsen, wobei jede dieser Rotationen durch eine 3x3-Matrix beschrieben wird [45].

Rotationsmatrix um die x-Achse:

$$R_X(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\,\alpha & -sin\,\alpha \\ 0 & sin\,\alpha & cos\,\alpha \end{pmatrix} \qquad (3.2)$$

Rotationsmatrix um die y-Achse:

$$R_Y(\beta) = \begin{pmatrix} cos\,\beta & 0 & sin\,\beta \\ 0 & 1 & 0 \\ -sin\,\beta & 0 & cos\,\beta \end{pmatrix} \quad (3.3)$$

Rotationsmatrix um die z-Achse:

$$R_Z(\gamma) = \begin{pmatrix} cos\,\gamma & -sin\,\gamma & 0 \\ sin\,\gamma & cos\,\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (3.4)$$

Die Translation T wird durch einen Vektor beschrieben:

$$T = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \quad (3.5)$$

Hierbei sind:

- t_x die Translation entlang der x-Achse
- t_y die Translation entlang der y-Achse
- t_z die Translation entlang der z-Achse

Ein Punkt $P = (x, y, z)$ im Raum wird durch diese Translation in einen neuen Punkt P' verschoben:

$$P' = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \quad (3.6)$$

Für die Überführung vom Sensorkoordinatensystem in das Weltkoordinatensystem sind also zwei Transformationen erforderlich, jeweils bestehend aus den Rotationsmatrizen und dem Translationsvektor.

Der Driving Scenario Designer speichert bei jeder LiDAR-Aufnahme sowohl die Position als auch die Ausrichtung des VUT. Diese Informationen sind entscheidend, um die korrekten Transformationen auf die Punktwolken anzuwenden. Mithilfe der MATLAB-Funktion *rigidtform3d* können die Parameter für Translation und Rotation in einer Matrix zusammengefasst werden. *pctransform* wendet die Transformation auf die jeweiligen Punktwolken an.

Um die Punktwolken vom Sensorkoordinatensystem in das Fahrzeugkoordinatensystem zu übertragen, wird der Längswinkel des Sensors von 26°, der durch die Platzierung auf der Windschutzscheibe entsteht, in die Rotationsmatrix der y-Achse integriert, und der Querwinkel von 8° in die Rotationsmatrix der x-Achse. Gleichzeitig muss die Position des Sensors in Abhängigkeit vom Ursprung des Fahrzeugkoordinatensystems bekannt sein. Der Driving Scenario Designer gibt die Sensorposition direkt mit x-, y- und z-Koordinaten vom Mittelpunkt der Hinterachse an.

Mit dieser Rotationsmatrix und den Positionsdaten des Sensors wird eine 3D-Transformation erstellt, die den Versatz des Sensors zum Fahrzeug berücksichtigt. Anschließend wird diese Transformation auf die Punktwolken angewendet, um die Punkte vom Sensorkoordinatensystem ins Fahrzeugkoordinatensystem zu überführen. Um die Punktwolken weiter vom Fahrzeugkoordinatensystem ins Weltkoordinatensystem zu transformieren, werden die Translation und der Gierwinkel des VUT herangezogen. Diese Daten, ebenfalls vom DSD bereitgestellt, ermöglichen die Erstellung einer weiteren 3D-Transformation, die die Position und Ausrichtung des Fahrzeugs in der Welt beschreibt.

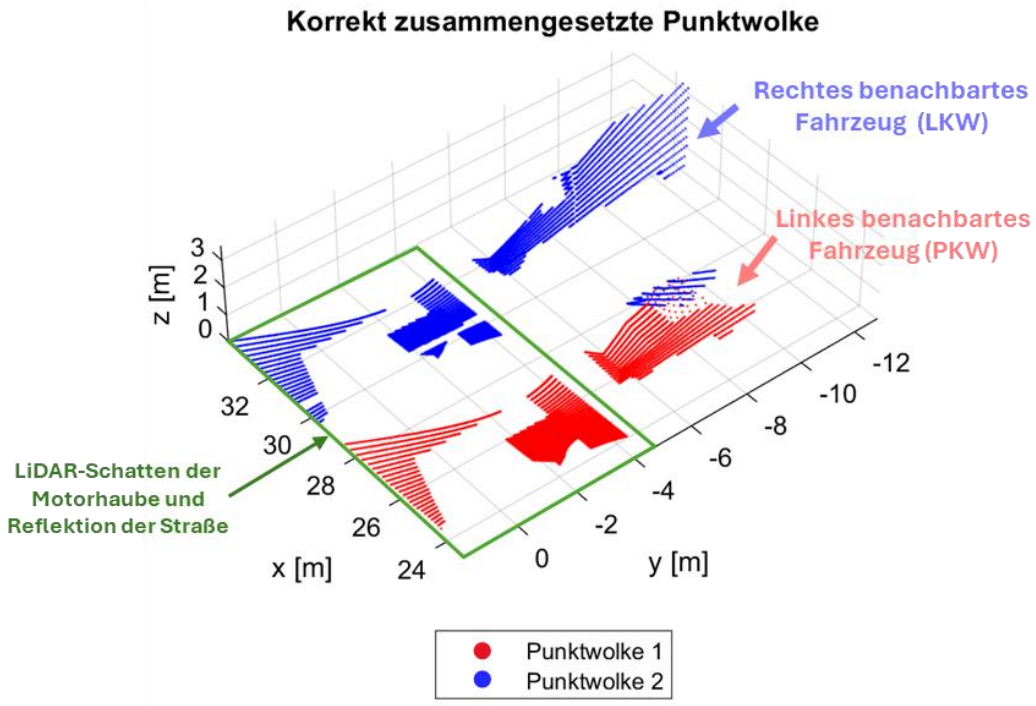


Abbildung 3.9 Korrekt zusammengesetzte Punktwolken

Das Ergebnis ist eine Punktwolke, die die Parklücke zwischen den Fahrzeugen korrekt darstellt (Abbildung 3.9).

All diese Schritte werden in eine Schleife eingebunden, um die LiDAR-Aufnahmen nacheinander zu verarbeiten und die Punktwolken schrittweise zusammenzuführen. Dabei werden für jede LiDAR-Aufnahme die jeweilige Position und Ausrichtung des VUT erfasst und die entsprechenden Transformationen angewendet. Abbildung 3.10 zeigt das finale Ergebnis der kartierten Parklücke.

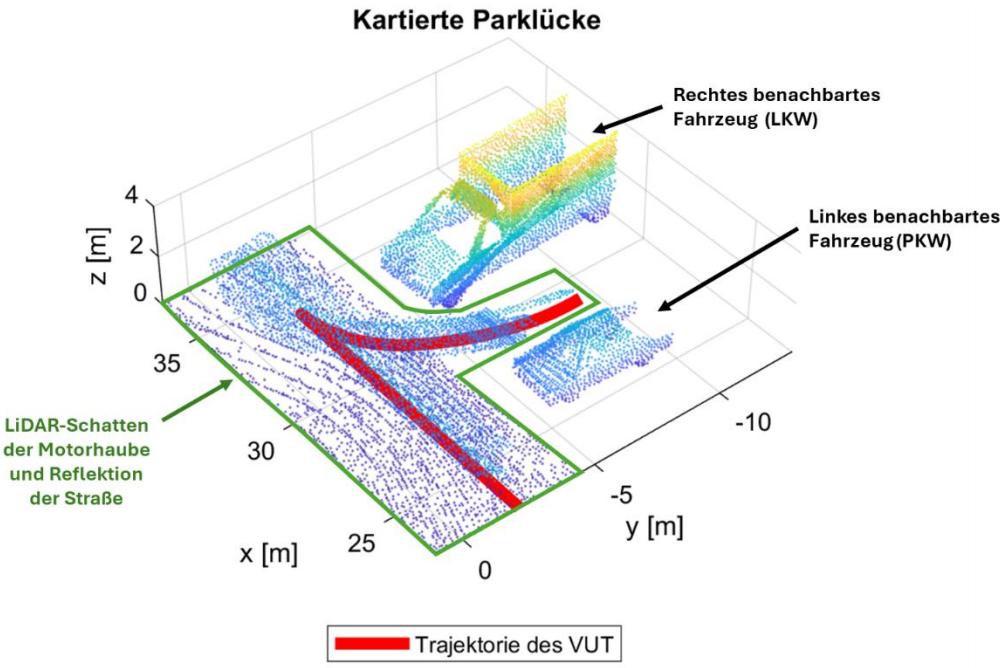


Abbildung 3.10 Vollständig kartierte Parklücke

Es ist zu beachten, dass im Driving Scenario Designer in der Sensorparametrierung standardmäßig „Ego Cartesian“ ausgewählt ist. Bei dieser Einstellung beziehen sich die x-, y- und z-Koordinaten der vom LiDAR erfassten Punkte auf das Ego-Fahrzeug (d.h. das VUT), dessen Ursprung in der Mitte der Hinterachse liegt. Um sicherzustellen, dass sich der Ursprung der Punktwolkendaten auf den Sensor selbst bezieht, muss die Einstellung auf „Sensor Cartesian“ geändert werden.

3.3.3 Berechnung der seitlichen Abstände

Zur Bestimmung der Abstände zu den benachbarten Fahrzeugen wurde der Ansatz gewählt, eine Objekterkennung als Funktion zu implementieren und darauf aufbauend die Distanzen zu ermitteln.

Bevor die Objekterkennung angewendet wird, muss die Punktwolke vorbereitet werden. Als erster Schritt wird die Bodenebene identifiziert, um zu verhindern, dass diese als Objekt erkannt wird. Entlang der z-Achse wird ein Referenzvektor definiert und ein Bodenschwellenwert festgelegt. Alle Punkte, die innerhalb dieses Schwellenwertes liegen, werden der Bodenebene zugeordnet, markiert und in den nächsten Berechnungsschritten nicht mehr berücksichtigt.

Die Funktion *pcsegdist* in MATLAB wird verwendet, um eine Punktwolke basierend auf der euklidischen Punkt-zu-Punkt-Distanz („Luftlinie“ zwischen zwei Punkten) zu segmentieren. Punkte, die näher als eine festgelegte Distanz beieinander liegen, werden demselben Cluster zugeordnet. Dadurch können in einer Punktwolke verschiedene Objekte identifiziert und voneinander getrennt werden. Um die erkannten Cluster wird anschließend mittels *pcfitcuboid* ein kubischer Begrenzungskörper (Cuboid) gelegt.

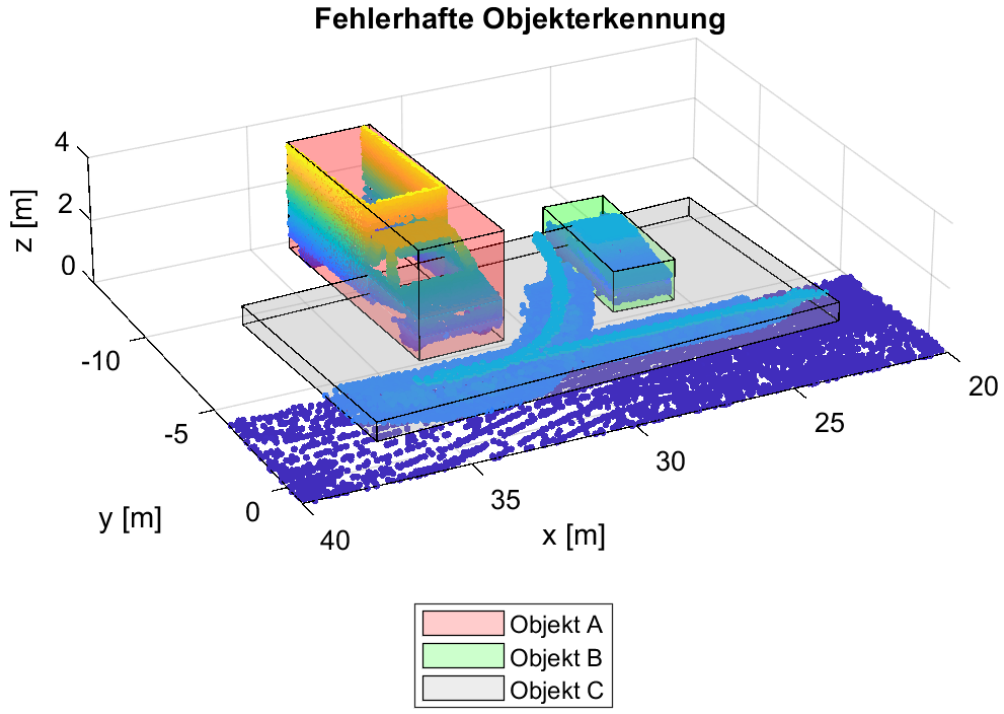


Abbildung 3.11 Fehlerhafte Objekterkennung

Wie in Abbildung 3.11 zu sehen ist, werden neben den benachbarten Fahrzeugen auch andere, unerwünschte Cluster erkannt. In diesem Fall ist erkennbar, dass der LiDAR-Schatten der Motorhaube als Objekt identifiziert wurde (Objekt C). Mithilfe der Länge, Breite und Höhe können die benachbarten Fahrzeuge identifiziert und herausgefiltert werden. Im realen Testfall ist diese Differenzierung jedoch nicht immer ideal, da die Dimensionen von Objekten variieren und ähnliche Größenverhältnisse aufweisen können, was zu erneuten Fehlidentifizierungen führen kann. Abbildung 3.12 zeigt das finale Ergebnis der korrigierten Objekterkennung.

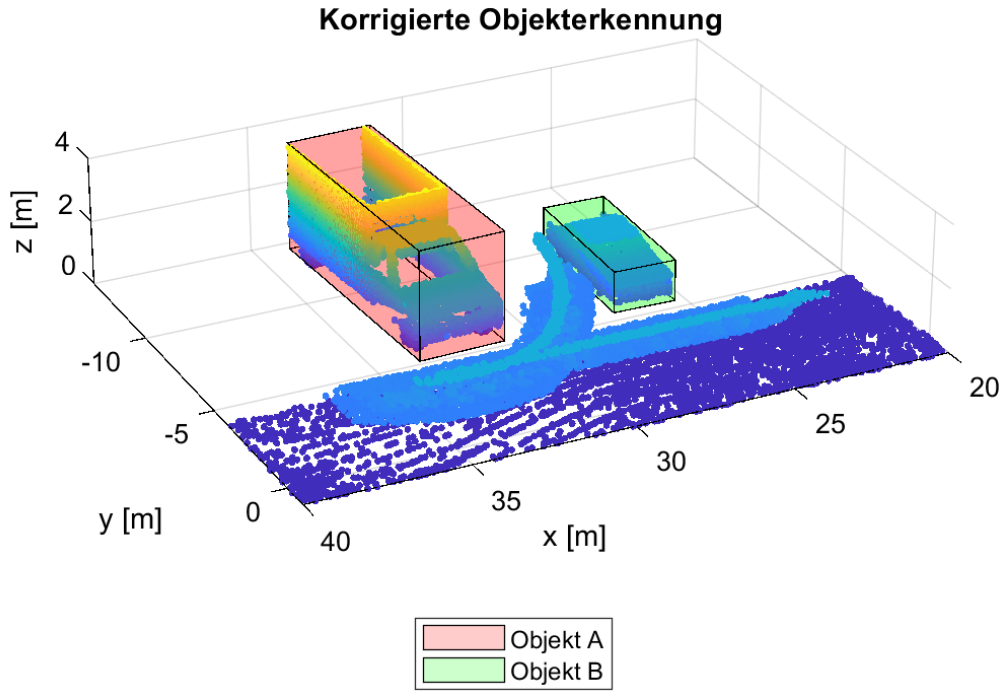


Abbildung 3.12 Korrigierte Objekterkennung

Der nächste Schritt besteht darin, die finale Position des VUT in die kartierte Parklücke zu integrieren und die seitlichen Abstände zu bestimmen. Da die Position und Abmessungen des VUT bekannt sind, kann ein weiteres Cuboid erstellt und eingefügt werden. Im realen Testfall soll dieser Schritt mithilfe der durch Dead Reckoning berechneten endgültigen Position des Fahrzeugs und dessen Abmessungen realisiert werden.

Um die seitlichen Abstände zu bestimmen, wird für die linke und rechte Seite des VUT-Cuboids die Ebenengleichung aufgestellt. Dies erfolgt in zwei Schritten:

Zuerst werden die Eckpunkte für jede Seite extrahiert. Drei der Eckpunkte werden ausgewählt, und aus diesen Punkten werden durch Subtraktion zwei Richtungsvektoren ($v\ und\ w$) gebildet. Durch das Kreuzprodukt der beiden Vektoren wird schließlich der Normalenvektor n' der Ebene berechnet.

$$v = P_1 - P_2 = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} - \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} \quad (3.7)$$

$$w = P_3 - P_1 = \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix} - \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \quad (3.8)$$

$$n' = v \times w \quad (3.9)$$

Die allgemeine Gleichung einer Ebene im 3D-Raum lautet:

$$A(x) + B(y) + C(z) + D = 0 \quad (3.10)$$

Hierbei sind $A = n'_x$, $B = n'_y$ und $C = n'_z$ die Komponenten des Normalenvektors.

Um den minimalen Abstand zwischen dem VUT und den benachbarten Fahrzeugen zu bestimmen, wird die Distanz zwischen den Punkten, die zum benachbarten Fahrzeug gehören ($P(x_p, y_p, z_p)$), zur jeweiligen Seitenebene des VUT berechnet.

$$Abstand = \frac{|A * x_p + B * y_p + C * z_p + D|}{\sqrt{A^2 + B^2 + C^2}} \quad (3.11)$$

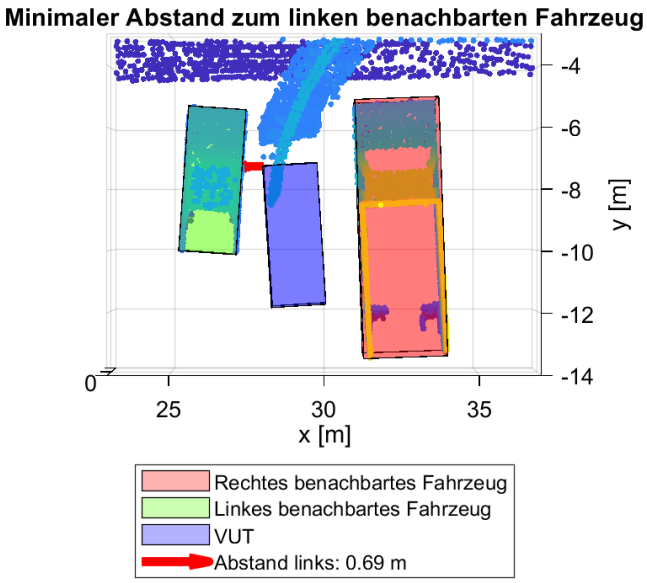


Abbildung 3.13 Minimaler Abstand zum linken benachbarten Fahrzeug

Abbildung 3.13 veranschaulicht aus der Vogelperspektive den Punkt des linken benachbarten Fahrzeuges mit dem geringsten Abstand zum VUT.

3.3.4 Zusammenfassung und Bewertung der Simulation

Der Einsatz der Automated Driving Toolbox und des Driving Scenario Designers in MATLAB ermöglicht eine detaillierte und praxisnahe Simulation des Szenarios, bei dem die Umgebungsdaten durch den LiDAR-Sensor erfasst und analysiert werden. Die erfolgreiche Kartierung der Parklücke durch die Kombination mehrerer LiDAR-Bilder liefert wichtige Einblicke in die Funktionsweise des Systems. Dabei zeigt die Simulation, dass wesentliche Herausforderungen, wie die korrekte Transformation der Punktwolken zwischen den verschiedenen Koordinatensystemen, die präzise Sensorausrichtung sowie die richtige Konfiguration der Sensoreinstellungen, erfolgreich umgesetzt wurden.

Besonders die Koordinatentransformation erwies sich als zentraler Aspekt, da sie gewährleistet, dass die Bewegungen des Fahrzeugs und des Sensors korrekt in das globale Koordinatensystem übertragen werden – eine Methode, die auch in der realen Anwendung übernommen werden kann.

Die Simulation verdeutlicht, dass die Einschätzung der Parkposition mittels LiDAR-Daten prinzipiell möglich ist, jedoch muss angemerkt werden, dass das vorgestellte Verfahren in seiner derzeitigen Form noch rudimentär ist. Während

einige wichtige Aspekte erfolgreich umgesetzt wurden, wurde die genaue Breitenbestimmung der Parklücke zu diesem Zeitpunkt der Simulation nicht berücksichtigt, da sich erst im späteren Verlauf des Entwicklungsprozesses herausstellte, dass diese von Vorteil sein könnte. Des Weiteren wird die Bestimmung der Sensorposition relativ zur Hinterachse unweigerlich zu Messungenauigkeiten führen, ebenso die schiefe Sensorpositionierung auf der Frontscheibe.

3.4 Realer Versuchsaufbau und Randbedingungen

Im Gegensatz zur Simulation werden im realen Testfall keine echten benachbarten Fahrzeuge verwendet. Stattdessen kommen Parklückenverstelleinheiten (Abbildung 3.14) zum Einsatz.

Abbildung 3.14 Parklückenverstelleinheiten

Der Einsatz der PLVs bietet mehrere Vorteile: Die Testfälle und Parklücken können sehr flexibel und schnell angepasst werden. Bei möglichen Kollisionen des Versuchsträgers mit den PLVs entsteht ein geringerer Sachschaden. Zudem sind PLVs in der Anschaffung günstiger als echte Fahrzeuge und ermöglichen eine leichtere Reproduzierbarkeit der Testfälle.

Das Versuchsfahrzeug ist ein VW Passat GTE, Baujahr 2016, der optional mit dem Parklenkassistenten ausgestattet ist und damit teilautomatisch in Parklücken einparken kann. Der Fahrer muss lediglich das Fahr- und

Bremspedal betätigen und die vom System geforderte Fahrstufe einlegen. Die Querführung übernimmt das Fahrzeug. Zusätzlich besteht die Möglichkeit, über eine Vector-Box auf den CAN-Bus zuzugreifen.

Als LiDAR-Sensor wird der Ouster OS1 eingesetzt, der in Kapitel 2.6.1 näher erläutert wurde. Der Sensor wird oben rechts auf der Windschutzscheibe platziert, wodurch er die Konturen der PLVs gut erfassen kann. Die Verbindung zum Messlaptop erfolgt über ein Ethernet-Kabel, und die Stromversorgung wird durch eine mobile Spannungsversorgung im Beifahrerfußraum sichergestellt.

Der schematische Aufbau und die Vernetzung der Messtechnik können Abbildung 3.15 entnommen werden. Blaue Pfeile repräsentieren den Datenverkehr zwischen den einzelnen Komponenten, gelbe Pfeile die Spannungsversorgung.

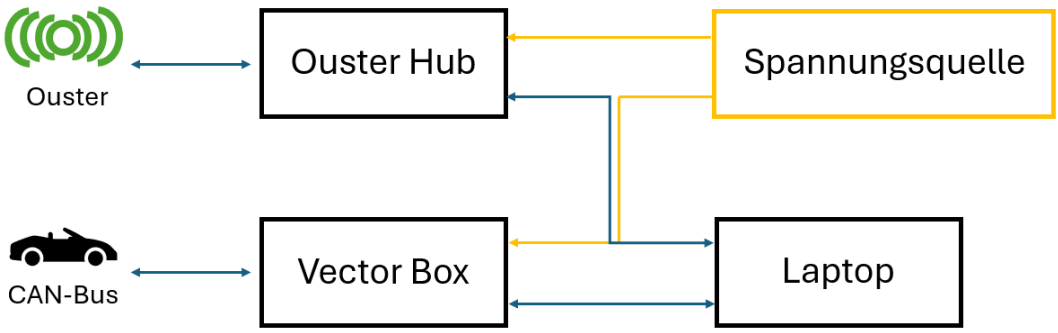


Abbildung 3.15 Aufbau und Vernetzung der Messtechnik

Für einen erfolgreichen Testfall sind folgende Randbedingungen definiert: Die PLVs werden für den Versuch parallel zueinander positioniert, wobei die Parallelität und der Abstand mittels Laserentfernungsmessung verifiziert wird. Der in dieser Diplomarbeit zu entwickelnde Testfall konzentriert sich erneut auf eine Querparklücke, an der das VUT links vorbeifährt und anschließend rückwärts einparkt. Es soll ein Geschwindigkeitsbereich von 3 bis 5 km/h eingehalten werden, um eine ausreichende Anzahl an Punktwolken zu sichern und die Erkennungswahrscheinlichkeit der Parklücke durch den Parklenkassistenten des Passats zu maximieren. Zum Start steht das VUT 3 bis 5 Meter vor der ersten PLV. Um eine möglichst störungsfreie Erfassung der

LiDAR-Daten zu ermöglichen, sollen keine zusätzlichen Hindernisse in der Umgebung platziert werden.

3.5 Umsetzung 1 – Kartierung der Parklücke mittels Dead Reckoning

3.5.1 Dead Reckoning

Um die in der Simulation erarbeitete Methodik in die Realität zu übertragen, muss zunächst die Position und Ausrichtung des Versuchsfahrzeugs im Weltkoordinatensystem bestimmt werden, bevor die LiDAR-Daten zusammengefügt werden können. Es existieren verschiedene Methoden zur Lokalisierung, darunter:

- **GPS (Global Positioning System)**: Ermöglicht eine metergenaue Positionsbestimmung bei freier Sicht zum Himmel, kann aber in urbanen Gebieten, Wäldern oder Tunneln an Genauigkeit verlieren.
- **DGPS (Differential GPS)**: Erhöht die Genauigkeit auf wenige Zentimeter durch Korrektursignale von festen Referenzstationen, erfordert jedoch zusätzliche Infrastruktur und hat eine begrenzte Reichweite.
- **Trägheitsnavigationssysteme (INS)**: Nutzen Beschleunigungssensoren und Gyroskope zur Positionsbestimmung, deren Genauigkeit kann jedoch über die Zeit driften.
- **WLAN- und Mobilfunk-Triangulation**: Nützlich in städtischen Umgebungen und Innenräumen, bieten jedoch je nach Netzdichte unterschiedliche Genauigkeiten. [46–49]

Da die notwendige Technik für die genannten Varianten nicht zur Verfügung stand oder zu ungenau war, wurde im ersten Konzept die Methode des Dead Reckoning gewählt. Dead Reckoning ist eine Navigationsmethode, bei der die aktuelle Position eines Fahrzeugs unter Verwendung einer zuvor bekannten Position und des seitdem zurückgelegten Weges berechnet wird. Diese Berechnung basiert auf der Geschwindigkeit, der Richtung und der Zeit, die seit der letzten bekannten Position vergangen ist.

Zur Bestimmung des zurückgelegten Weges können die Raddrehzahlen verwendet werden. In Kombination mit dem statischen Reifenradius lässt sich die zurückgelegte Wegstrecke berechnen. Der statische Reifenradius wird genutzt, da die dynamischen Effekte wie Fliehkraft und Reifenverformung aufgrund der geringen Geschwindigkeit minimal sind.

Diese Methode ist unabhängig von externen Signalen und erfordert keine zusätzliche Infrastruktur, wodurch sie kostengünstig und einfach zu implementieren ist. Allerdings nimmt die Genauigkeit mit der Zeit und der Entfernung ab, da sich Fehler kumulieren. Außerdem können Schlupf und Veränderungen der Fahrbahnoberfläche zu Abweichungen führen. Bei niedrigen Geschwindigkeiten kann Dead Reckoning jedoch sehr genau sein und eine zuverlässige Positionsbestimmung liefern [50, 51].

3.5.2 Datenaufnahme

Wie bereits in Abschnitt 2.6.1 erläutert, konnte der Ouster LiDAR erfolgreich in ecu.test integriert werden. Dadurch ist es möglich, die LiDAR-Daten direkt über ecu.test aufzuzeichnen und zu speichern. Während das VUT an der Parklücke vorbeifährt, werden kontinuierlich Punktwolken aufgezeichnet. Für jede Punktwolke werden eine .pcap-Datei und eine .json-Datei bereitgestellt:

- .json-Datei: Speichert die Metadaten des Sensors.
- .pcap-Datei: Enthält die eigentlichen Sensordaten (Punktwolken).

Mithilfe einer Schleife mit nachgestellter Bedingungsprüfung werden die Bilder so lange aufgenommen, bis das Testfahrzeug wieder eine Geschwindigkeit von 0 km/h erreicht, um in den Rückwärtsgang zu wechseln und damit den Parkvorgang zu starten. Die Geschwindigkeit wird durch eine CAN-Abfrage über die Vector-Box ermittelt.

Die Raddrehzahlen der Hinterachse und die gewählte Fahrstufe werden sowohl beim Vorbeifahren an der Parklücke als auch beim Einparken über ecu.test mithilfe der Vector-Box erfasst und in einer .csv-Datei gespeichert.

3.5.3 Berechnung der Trajektorie aus den Raddrehzahlen

Die Berechnung der Trajektorie mittels Dead Reckoning erfolgt in mehreren Schritten. Zunächst wird in das Verzeichnis der Raddrehzahldaten gewechselt, um die benötigten Daten in MATLAB zu laden. Eine .csv-Datei mit Raddrehzahlen, Zeitstempeln und der gewählten Fahrstufe (vorwärts oder rückwärts) wird eingelesen.

Die Zeitstempel werden um einen definierten Offset korrigiert. Dadurch wird eine konsistente Zeitskala für die Synchronisation mit anderen Datenquellen, wie z.B. LiDAR-Daten, geschaffen. Eine genauere Erklärung der Synchronisation erfolgt in Abschnitt 3.5.4. Da auf dem CAN-Bus die Raddrehzahl nicht gleichzeitig zur gewählten Fahrstufe übertragen wird, müssen den jeweiligen Raddrehzahl-Wertepaaren noch die richtige gewählte Fahrstufe zugeordnet werden.

Insbesondere werden die Raddrehzahlen bei Rückwärtsfahrt negativ gesetzt, um die korrekte Fahrtrichtung zu berücksichtigen. Dabei wird auf den hinterlegten Wert der Fahrstufe zurückgegriffen.

Eine neue, gleichmäßige Zeitskala wird erstellt, um eine kontinuierliche Trajektorie berechnen zu können. Das Zeitintervall (dt) beträgt dabei 0,1 Sekunden. Die Raddrehzahlen der linken und rechten Räder werden auf diese neue Zeitskala dezimiert und von Umdrehungen pro Minute in Umdrehungen pro Sekunde umgerechnet.

Die Orientierung und Position des Fahrzeuges wird festgelegt. Die Startkoordinaten werden auf *xPos* = 0 und *yPos* = 0 gesetzt, während die Orientierung θ auf 0 Radiant eingestellt wird.

Für jedes Zeitintervall werden die zurückgelegten Distanzen d der linken und rechten Räder berechnet. Dies erfolgt durch Multiplikation der interpolierten Raddrehzahlen ω mit dem Radumfang U_{Rad} und der Zeitspanne dt.

$$d = \omega * U_{Rad} * dt \qquad (3.12)$$

Um nun auf die Orientierung θ (Gierwinkel) zu schließen, wird die zurückgelegte Distanz des linken Rades von der des rechten Rades abgezogen und dieser Wert

durch die Spurweite S geteilt. Der Hintergrund ist, dass das kurveninnere Rad eine kürzere Strecke zurücklegt als das kurvenäußere Rad, wodurch ein Winkel entsteht.

$$\theta = \frac{(d_{Rechts} - d_{Links})}{S} \tag{3.13}$$

Mit Hilfe des Gierwinkels kann die momentane Position des Fahrzeugs in einem kartesischen Koordinatensystem bestimmt werden. Dazu werden die x- und y-Koordinaten berechnet, indem zur vorherigen Position der mittlere Weg des linken und rechten Rades addiert wird. Dieser mittlere Abstand wird dann je nach Berechnung mit dem Kosinus oder dem Sinus des Gierwinkels multipliziert. Der Kosinus wird zur Berechnung der Änderung in x-Richtung verwendet, der Sinus zur Berechnung der Änderung in y-Richtung.

$$x_{Pos\ neu} = x_{Pos\ alt} + \frac{d_{Rechts} + d_{Links}}{2} * cos\,\theta \tag{3.14}$$

$$y_{Pos\ neu} = y_{Pos\ alt} + \frac{d_{Rechts} + d_{Links}}{2} * sin\,\theta \tag{3.15}$$

Diese Schritte werden für alle Zeitintervalle wiederholt, wodurch eine kontinuierliche Trajektorie des Fahrzeugs über die Zeit hinweg entsteht. Abbildung 3.16 stellt die Trajektorie des Testfahrzeugs für eine Messreihe grafisch dar.

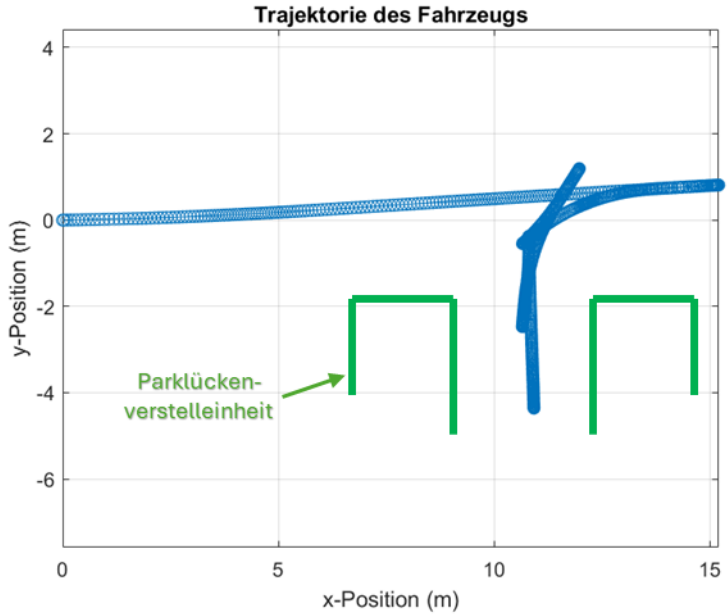


Abbildung 3.16 Berechnete Trajektorie des VUT

In der Grafik ist der Fahrtverlauf zu erkennen, welcher einen leichten Drift in die positive y-Richtung aufweist. Für die Nachvollziehbarkeit der Trajektorie sind die PLVs nachträglich eingezeichnet worden.

3.5.4 Synchronisierung von LiDAR und Trajektorie

Da die Zeitstempel der Raddrehzahlen bei null Sekunden beginnen, während die Zeitstempel der LiDAR-Daten der Windows-Systemzeit zum Aufnahmezeitpunkt entsprechen, müssen beide Datensätze auf eine gemeinsame Zeitskala gebracht werden. Hierfür wird auf die Zeitstempel der Raddrehzahlen ein festgelegtes Zeit-Offset aufaddiert. Dieses Offset entspricht dem Startzeitpunkt der Datenaufnahme, der aus den Logdaten von ecu.test entnommen werden kann und ebenfalls der Windows-Systemzeit entspricht.

Die Zeitstempel der LiDAR-Daten werden direkt aus den Dateinamen extrahiert. ecu.test speichert die LiDAR-Aufnahmen so ab, dass der letzte Teil des Dateinamens dem Aufnahmezeitpunkt entspricht.

Zur Synchronisation der LiDAR-Daten mit der Fahrzeugtrajektorie wird jeder Zeitstempel der berechneten Trajektorie mit den Zeitstempeln der LiDAR-Daten verglichen. Der LiDAR-Zeitstempel, der dem aktuellen Trajektorie-Zeitpunkt am nächsten liegt, wird identifiziert. Anschließend wird überprüft, ob dieser innerhalb

eines definierten Toleranzfensters liegt. Ist das der Fall, werden die entsprechenden LiDAR-Daten mit der aktuellen Position und Orientierung der Trajektorie kombiniert und gespeichert.

3.5.5 Kartierung der Parklücke

Zu jedem LiDAR-Bild ist nun die Position und Orientierung des VUT bekannt. Um die einzelnen Bilder zusammenzusetzen, muss die Koordinatentransformation vom Sensorkoordinatensystem in das Fahrzeugkoordinatensystem und schließlich ins Weltkoordinatensystem erfolgen. Dabei konnte auf den Code aus der Simulationsumgebung zurückgegriffen werden.

Für die Überführung vom Sensorkoordinatensystem ins Fahrzeugkoordinaten-system wird die relative Position des Sensors zum Mittelpunkt der Hinterachse benötigt. Diese wird mithilfe eines Gliedermaßstabs bestimmt, was sich als schwierig erweisen kann und zwangsläufig zu Ungenauigkeiten führt.

Zudem sind genaue Messungen der Sensorposition und -ausrichtung erforderlich, um eine korrekte Ausrichtung der Punktwolken zu gewährleisten. Ohne präzise Kenntnis dieser Parameter würden die aufgenommenen Punktwolken aufgrund der Neigung der Windschutzscheibe schief abgespeichert, was die Weiterverarbeitung erschweren würde.

Die Rotationsmatrix der Sensororientierung wird mit der Sensortranslation und des Befehls *rigidtform3d* zu einer Transformationsmatrix kombiniert. Diese Matrix ermöglicht die Transformation der Punktwolken vom Sensorkoordinatensystem in das Fahrzeugkoordinatensystem.

Das gleiche Verfahren wird für die Transformation der Punktwolken vom Fahrzeugkoordinatensystem in das Weltkoordinatensystem verwendet. Eine Rotationsmatrix wird basierend auf der Fahrzeugausrichtung bestimmt, und eine Translation erfolgt auf Grundlage der x- und y-Positionen des Fahrzeugs.

Während des Transformationsprozesses wird der Boden aus der Punktwolke entfernt, um eine klarere Darstellung der relevanten Daten zu ermöglichen. Dies wird durch die Verwendung der *pcfitplane*-Funktion in MATLAB erreicht. Diese Funktion nutzt den MSAC-Algorithmus (M-estimator Sample Consensus). Der Algorithmus wählt zufällige Teilmengen der Punktwolke aus, berechnet mögliche

Ebenen und bewertet, wie gut diese Ebenen die Punkte in der Wolke beschreiben, basierend auf einem maximal zulässigen Abstand eines Punktes von der Ebene. Die Ebene mit dem besten Score, also der besten Übereinstimmung mit den Daten, wird schließlich als das Modell ausgewählt, das die Ebene repräsentiert. Damit ist es möglich die Bodenebene zu identifizieren [52–54].

Wie in der Simulation werden die transformierten Punktwolken mit dem Befehl *pcmerge* in MATLAB chronologisch zusammengefasst.

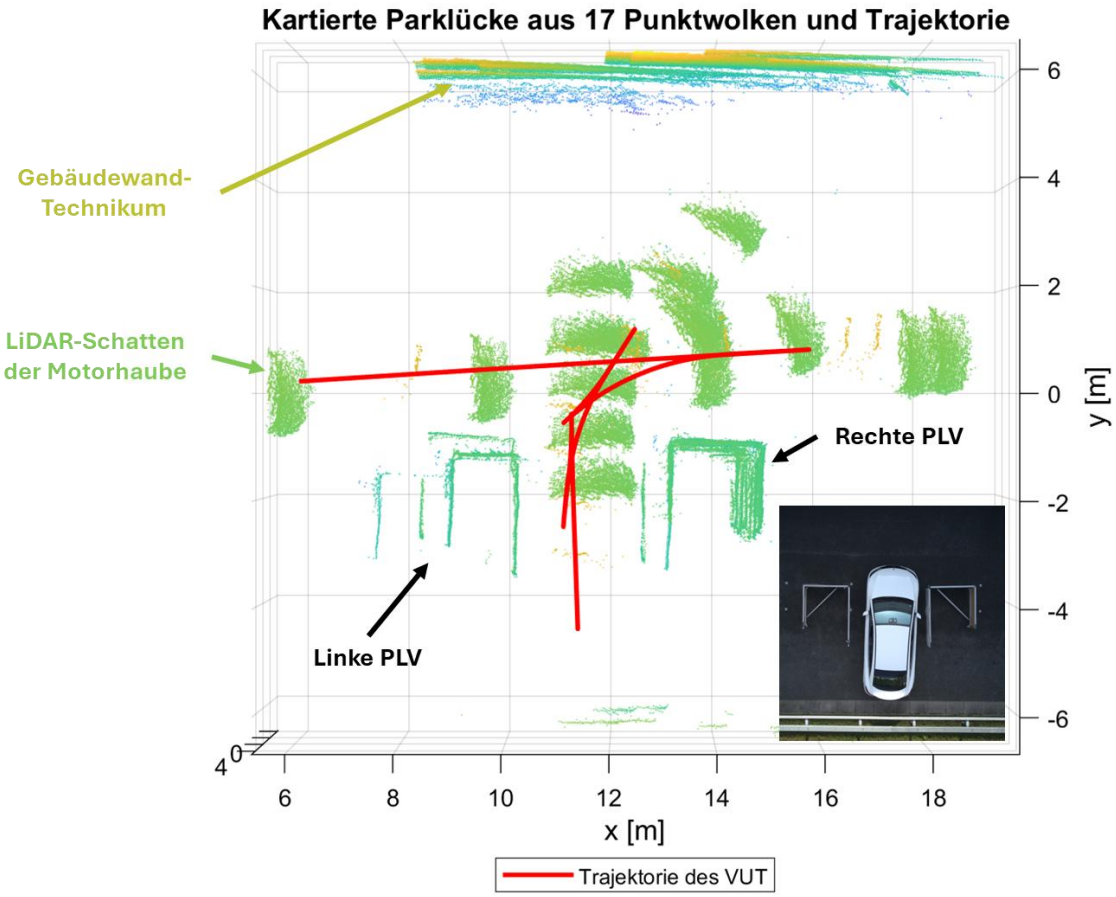


Abbildung 3.17 Vogelperspektive der Parklücke mit 17 Punktwolken

Abbildung 3.17 zeigt die Parklücke mit 17 zusammengesetzten Punktwolken, der berechneten Trajektorie und einem realen Foto aus der Vogelperspektive (Foto nachgestellt, Fahrzeug entspricht nicht dem im Test genutzten Fahrzeug). Die PLVs werden sehr ungenau abgebildet. Gut zu erkennen ist die leichte Abweichung der Trajektorie vom Fahrzeugmittelpunkt. Diese Abweichung kommt durch Ungenauigkeiten bei der Vermessung der relativen Position des Sensors

zur Hinterachse zu Stande. Neben den Punktwolken ist auch der LiDAR-Schatten der Motorhaube des VUT in der erstellten Karte erkennbar. Das Ergebnis ist jedoch unzureichend und kann nicht als Basis für weitere Berechnungen dienen, weshalb ein neues Konzept erforderlich ist. Im nächsten Kapitel werden die Probleme dieses Ansatzes detaillierter erläutert.

3.5.6 Nachteile der ersten Umsetzung

Das vorgestellte Konzept führt aus mehreren Gründen zu einer sehr ungenau kartierten Parklücke, die für weitere Arbeiten ungeeignet ist.

Die zeitliche Synchronisation zwischen LiDAR und Raddrehzahlen stellt ein erhebliches Problem dar. Die Genauigkeit dieser Methode konnte nicht überprüft werden. Eine Lösung des Problems könnte die Implementierung des Precision Time Protocol (PTP) sein. In einem PTP-System gibt es einen Master, der als primäre Zeitquelle fungiert, und mehrere Slaves (LiDAR und Radrehzahlsensor), die ihre Zeit mit der des Masters synchronisieren, um eine hohe Genauigkeit zu gewährleisten.

Die nächste Ungenauigkeit entsteht bei der Bestimmung der relativen Position des LiDAR-Sensors zur Mitte der Fahrzeughinterachse. Abweichungen in der Positionierung können zu Fehlern in den Transformationsmatrizen führen. Da die Position nur mit einem Gliedermaßstab bestimmt wurde, führt dies zwangsläufig zu Ungenauigkeiten. Ebenso führt die Bestimmung der Rotationswinkel des Sensors zu Fehlern.

Die Messung des Radumfangs ist eine weitere potenzielle Fehlerquelle. Ungenauigkeiten bei der Bestimmung des Radumfangs sowie Einflüsse von Temperatur, Alterung und Witterung können die Berechnung der zurückgelegten Strecke und damit die gesamte Trajektorie beeinflussen.

Schlupf an der Hinterachse könnte ebenfalls zu Abweichungen führen.

Diese Fehlerquellen summieren sich und führen zu einem ungenauen 3D-Bild der Parklücke, das für weitere Analysen nicht geeignet ist. Daher wurde beschlossen eine alternativen Methode zu entwickeln, welche im nächsten Kapitel ausführlich erläutert wird.

3.6 Umsetzung 2 – Kartierung der Parklücke mittels SLAM

3.6.1 SLAM

SLAM in MATLAB besteht aus einer Reihe von Schritten. Zunächst werden zwei Punktwolken (eine feste Punktwolke und eine bewegliche Punktwolke), die von unterschiedlichen Standorten aufgenommen wurden, mittels der Funktion *pcdownsample* dezimiert. Dieses Downsampling reduziert die Datenmenge und erhöht die Verarbeitungsgeschwindigkeit.

Anschließend wird die Funktion *pcregistericp* eingesetzt, um die bewegliche Punktwolke an die feste Punktwolke anzupassen. Hierbei kommt der Iterative Closest Point-Algorithmus mit der Point-to-Plane-Metrik zum Einsatz (Erläuterung siehe Kapitel 2.5). Die durch den ICP-Algorithmus ermittelte Transformationsmatrix wird mithilfe der Funktion *pctransform* auf die bewegliche Punktwolke angewendet. Dadurch wird die bewegliche Punktwolke so transformiert, dass sie bestmöglich mit der festen Punktwolke übereinstimmt. Dieser Schritt ist entscheidend, um die beiden Punktwolken in einem gemeinsamen Koordinatensystem auszurichten.

Im nächsten Schritt werden die transformierte bewegliche Punktwolke und die feste Punktwolke mithilfe der Funktion *pcmerge* zu einer einzigen, kombinierten Punktwolke zusammengeführt. Das Resultat ist eine kohärente dreidimensionale Darstellung der Umgebung, die die Informationen beider Punktwolken integriert und somit eine verbesserte Kartierung ermöglicht.

Dieser Prozess wird in einer Schleife über alle aufgenommenen Punktwolken implementiert. Durch die Anwendung des beschriebenen Verfahrens entsteht am Ende ein vollständiges 3D-Abbild der Umgebung. Jede neue Punktwolke wird dabei an die bereits kombinierte Punktwolke angepasst und integriert. Das Vorgehen ermöglicht eine sukzessive Erweiterung der Karte und eine stetige Verbesserung der Genauigkeit der dargestellten Umgebung.

In Abbildung 3.18 ist das Ergebnis einer Messung dargestellt. Während der Vorbeifahrt wurden insgesamt 14 Punktwolken aufgenommen und mittels SLAM zusammengeführt. Die beiden PLVs sind dabei deutlich erkennbar. Aufgrund der schrägen Positionierung des LiDAR-Sensors auf der Windschutzscheibe ist die

resultierende SLAM-Punktwolke ebenfalls schief angeordnet. In der Grafik kann dies nur anhand des Farbverlaufs der z-Achse erkannt werden.

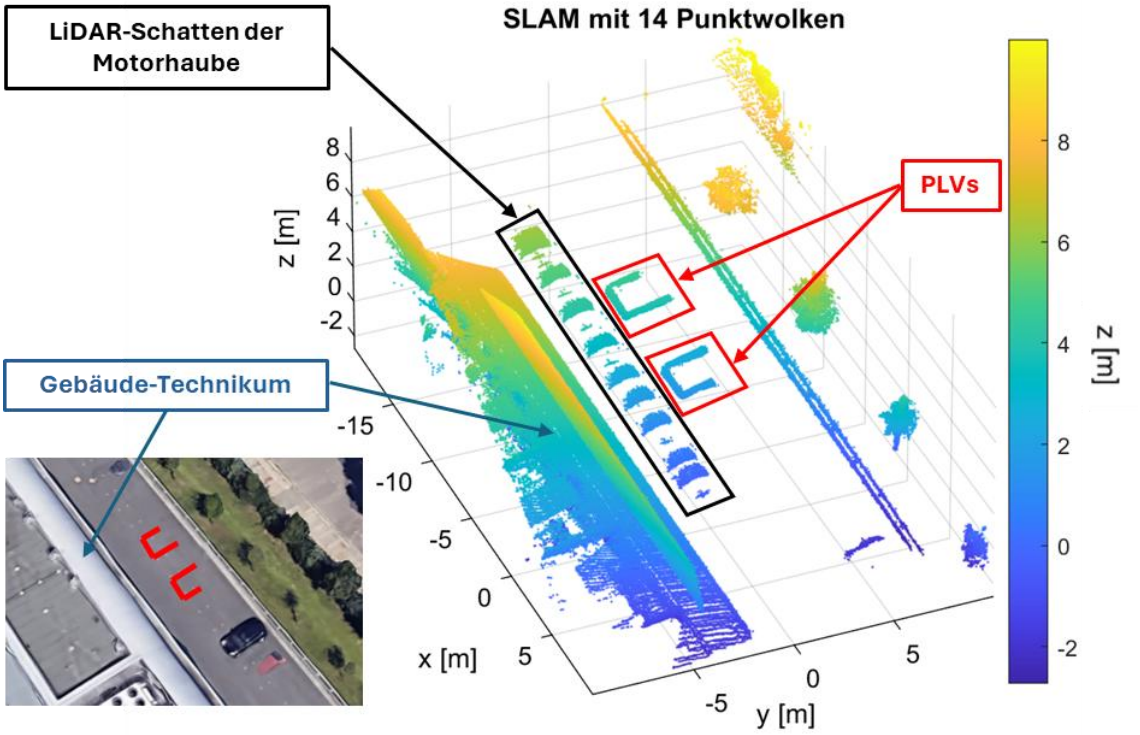


Abbildung 3.18 SLAM der Umgebung mit 14 Punktwolken [30]

Um die Erkennbarkeit der relevanten Strukturen zu verbessern, wurde in der Darstellung bereits die Bodenebene herausgefiltert.

Für die weitere Verwendung der SLAM-Punktwolke ist es notwendig, diese entsprechend zu rotieren und auf den relevanten Bereich zuzuschneiden, um eine effiziente Weiterverarbeitung zu ermöglichen.

3.6.2 Berechnung der Parklückenbreite

Das Ziel besteht darin, die SLAM-Punktwolke durch eine geeignete Rotation so zu transformieren, dass der Boden nivelliert wird. Zudem sollen die Bodenpunkte entfernt werden, um die relevanten Punkte der Parklückenverstelleinheiten zu extrahieren. Auf dieser Grundlage kann anschließend die Breite der Parklücke berechnet werden.

Zunächst werden Begrenzungen in den x-, y- und z-Richtungen festgelegt, um die SLAM-Punktwolke auf den relevanten Bereich mit den Parklücken-verstelleinheiten zu beschränken.

Die innerhalb dieser festgelegten Begrenzungen liegenden Punkte werden extrahiert. Anschließend wird ein Medianfilter auf die Punktwolke angewendet, um Rauschen zu reduzieren und die Punkte zu glätten. Dies führt zu einer saubereren Punktwolke, die weniger durch zufällige Schwankungen beeinflusst wird. In der betrachteten Punktwolke ist der Einfluss des Medianfilters jedoch gering, da die SLAM-Punktwolke im ausgewählten Bereich bereits eine geringe Rauschintensität aufweist. Daher ist die Notwendigkeit, Ausreißer mittels Medianfilter zu glätten, weniger relevant.

Im nächsten Schritt wird die Bodenebene ermittelt. Wie bereits im ersten Konzept beschrieben, wird hierfür die Funktion *pcfitplane* verwendet. Diese Funktion passt eine Ebene an die Punktwolke an und identifiziert somit die Bodenpunkte, die anschließend entfernt werden können.

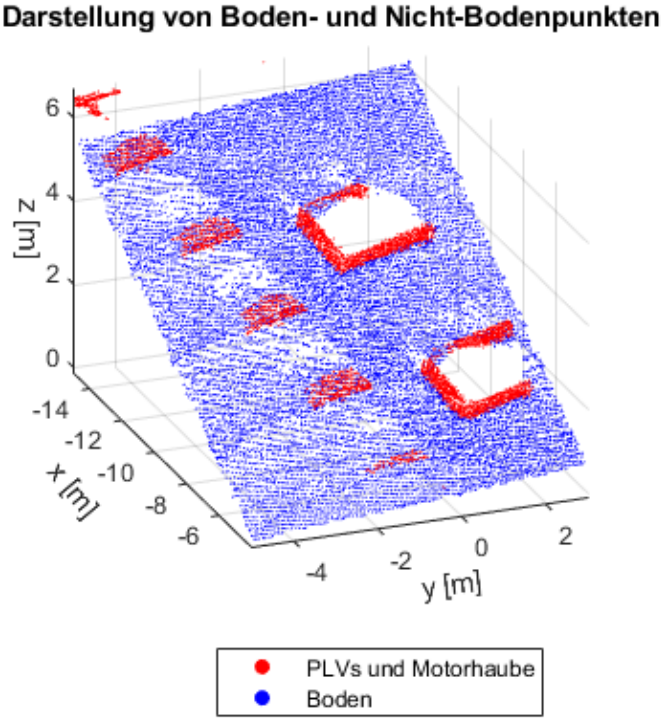


Abbildung 3.19 Identifizierte Bodenebene

Die roten Punkte in Abbildung 3.19 repräsentieren die PLVs sowie die Motorhaube des Fahrzeugs und bleiben für die weitere Analyse erhalten. Die

blauen Punkte gehören zur ermittelten Bodenebene und werden entfernt. Die Ebenennormale der Bodenebene wird jedoch gespeichert und für den nächsten Schritt verwendet.

Um die Punktwolke neu auszurichten und korrekt zu drehen, wird die Rodrigues-Rotationsformel verwendet. Diese Methode ermöglicht es, eine Rotationsmatrix zu erstellen, die die Punktwolke so transformiert, dass die Normale der ermittelten Ebene mit der z-Achse ausgerichtet wird. Dadurch wird die Schieflage des Sensors auf der Windschutzscheibe korrigiert, ohne dass eine direkte Messung der Rotationswinkel des Sensors erforderlich ist [55, 56].

Zunächst wird die Rotationsachse k berechnet, indem das Kreuzprodukt der Ebenennormale n und der z-Achse z gebildet.

$$k = n \times z \tag{3.16}$$

Für den Rotationswinkel θ wird ebenfalls die Normale der gefundenen Ebene $n = [n_x, n_y, n_z]^T$ und des Vektors der z-Achse $z = [0,0,1]^T$ benötigt. Die Berechnung erfolgt mittels:

$$\theta = cos^{-1}\left(\frac{n * z}{|n| \circ |z|}\right) \tag{3.17}$$

Hierbei gibt θ den Winkel an, um den die Punktwolke gedreht werden muss, damit die Ebenennormale mit der z-Achse ausgerichtet wird (Winkel zwischen zwei Vektoren).

Die Rodrigues-Formel berechnet die Rotationsmatrix R_O für eine Drehung um die Achse k mit dem Winkel θ:

$$R_O = I + sin(\theta)\, K + (1 - cos(\theta)K^2 \tag{3.18}$$

Dabei ist I die Einheitsmatrix und K die Kreuzprodukt-Matrix von k:

$$K = \begin{pmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{pmatrix} \tag{3.19}$$

Die Rotationsmatrix R_O wird auf jeden Punkt p der Punktwolke angewendet, um den transformierten Punkt p' zu erhalten:

$$p' = R_O * p \tag{3.20}$$

Abbildung 3.20 zeigt die neu ausgerichtete Punktwolke.

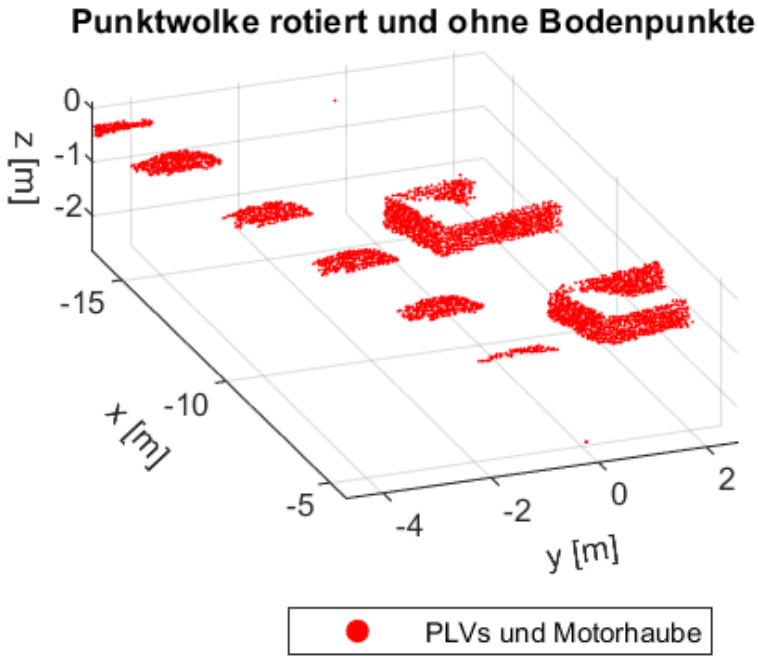


Abbildung 3.20 Rotierte Punktwolke und ohne Bodenpunkte

Im nächsten Schritt werden die Punkte mithilfe des DBSCAN-Algorithmus (Density-Based Spatial Clustering of Applications with Noise) in verschiedene Cluster gruppiert. Dies ermöglicht die Unterscheidung zwischen den beiden Parklückenverstelleinheiten und der Motorhaube. DBSCAN identifiziert Cluster von Punkten, die nahe beieinander liegen, und ignoriert dabei Ausreißer. Dadurch können Gruppen zusammenhängender Punkte, die relevante Objekte in der Umgebung darstellen, klar identifiziert werden.

Der Algorithmus basiert auf zwei Parametern: der maximalen Distanz zwischen zwei Punkten, damit sie als Teil desselben Clusters betrachtet werden, und der minimalen Anzahl von Punkten, die erforderlich ist, um einen Cluster zu bilden. In Abbildung 3.21 wird das Ergebnis der Clusteranalyse visualisiert, wobei Punkte der gleichen Farbe einem Cluster zugeordnet sind. Die Punkte der Motorhaube konnten dabei erfolgreich entfernt werden, indem nur die zwei Cluster erhalten bleiben, deren Mittelpunkte am weitesten in positive y-Richtung liegen. [57]

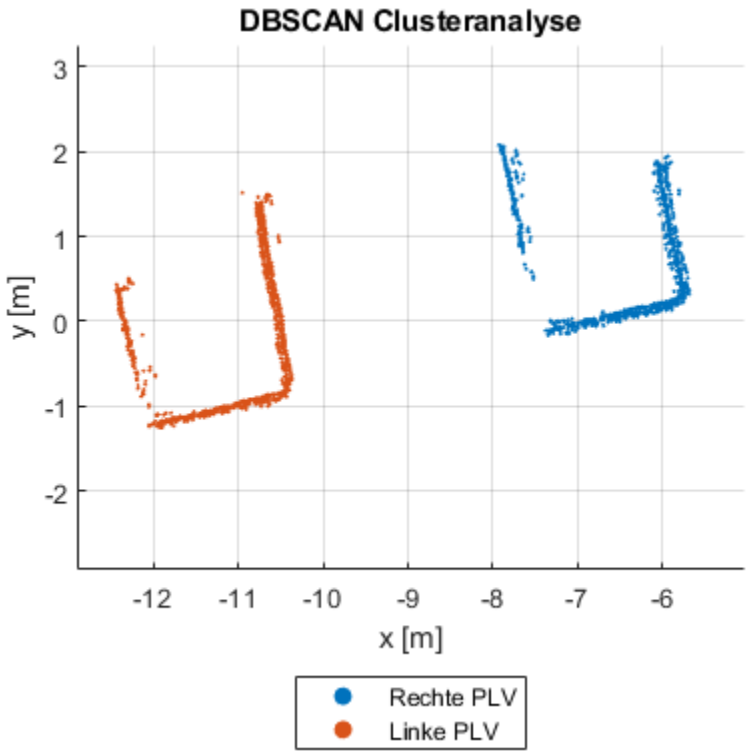


Abbildung 3.21 Unterteilung der linken und rechten PLV in zwei Cluster

Der nächste Schritt besteht darin, die Eckpunkte der Innenseiten der Parklückenverstelleinheiten zu identifizieren. Die relevanten Punkte für die Ecken werden basierend auf ihren x- und y-Koordinaten ausgewählt. Aus jeweils vier ausgewählten Punkten pro Ecke wird ein gemittelter Eckpunkt gebildet.

Durch lineare Regression werden die Gleichungen der Geraden ermittelt, die durch die Eckpunkte der linken und rechten PLV verlaufen. MATLAB bietet hierfür die Funktion *polyfit*, die ein Polynom beliebigen Grades (in diesem Fall ersten Grades) an die Daten anpasst. Dabei werden die Parameter m und n

bereitgestellt, die für die Definition einer Geraden nach $y = mx + n$ benötigt werden.

Zur Berechnung der Breite des Parkplatzes werden die Geraden in jeweils 100 äquidistante Punkte unterteilt. Die Wahl von 100 Punkten ist dabei ein Kompromiss zwischen Rechenaufwand und Genauigkeit.

Anschließend werden die senkrechten Abstände der Punkte auf der linken Geraden zur rechten Geraden und umgekehrt bestimmt. Der Abstand eines Punktes (x_0, y_0) zur Geraden wird mit der folgenden Formel berechnet:

$$d = \frac{|mx_0 - y_0 + n|}{\sqrt{m^2 + 1}} \qquad (3.21)$$

Aus den insgesamt 200 berechneten Abständen wird der Mittelwert gebildet. Die Bildung des Mittelwerts über eine große Anzahl von Messungen trägt dazu bei, Ausreißer und Messrauschen zu reduzieren. Dadurch wird eine robuste Schätzung der durchschnittlichen Breite der Parklücke erzielt.

In Abbildung 3.22 sind die beiden Regressionsgeraden eingezeichnet. Die markierten Kreise zeigen die Punkte an, die zur Bestimmung der Eckpunkte und für die Berechnung der Geraden ausgewählt wurden.

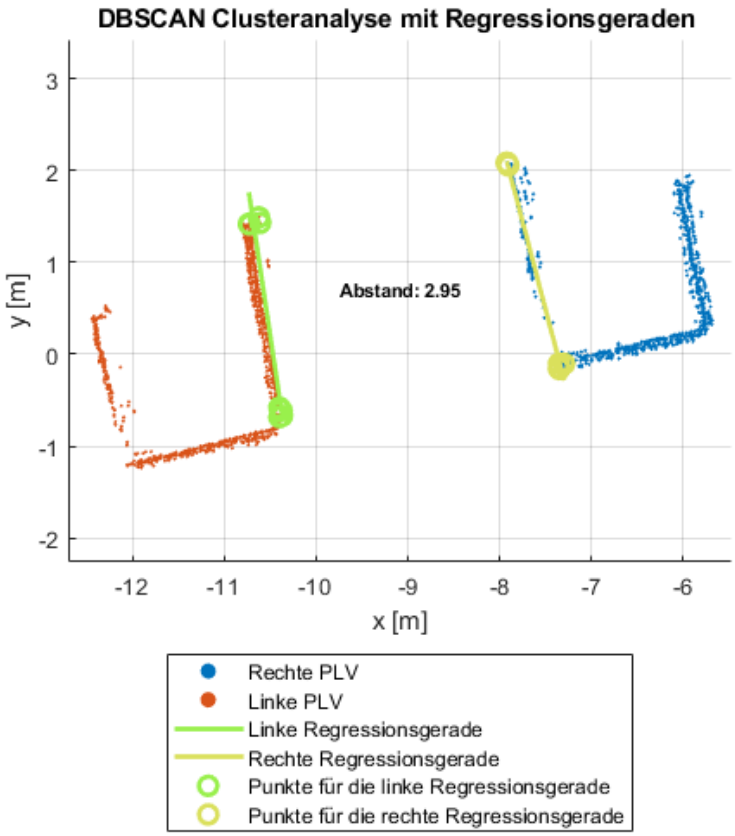


Abbildung 3.22 Regressionsgeraden für die Bestimmung der Parklückenbreite

3.6.3 Berechnung des Vorbeifahr- und Seitenabstand

Für die Auswertung ist es für den Prüfingenieur von Bedeutung, den Abstand zu ermitteln, mit dem das VUT an der Parklücke vorbeigefahren ist. Ziel dieser Analyse ist es, aus den während der Vorbeifahrt erfassten Punktwolken jene auszuwählen, die die präziseste Ableitung des Abstands zwischen dem Fahrzeug und der Parklückenverstelleinheit ermöglicht. Dieser Prozess umfasst mehrere Schritte, die im Folgenden betrachtet werden.

Die Punktwolken werden, wie bereits in Kapitel 3.6.2 beschrieben, entsprechend rotiert, auf den relevanten Bereich zugeschnitten und die Bodenebene entfernt. Um die aussagekräftigste Punktwolke zu ermitteln, erfolgt eine Bewertung der Punktwolken basierend auf ihrer Punktanzahl. Die Annahme hierbei ist, dass eine höhere Punktdichte eine bessere Erkennung der Parklückenverstelleinheit ermöglicht, da die Konturen vom LiDAR-Sensor besser erfasst werden. Je mehr Punkte eine Punktwolke enthält, desto höher wird sie daher bewertet. In

Abbildung 3.23 sind drei Punktwolken dargestellt, die während der Vorbeifahrt an der Parklücke aufgenommen wurden.

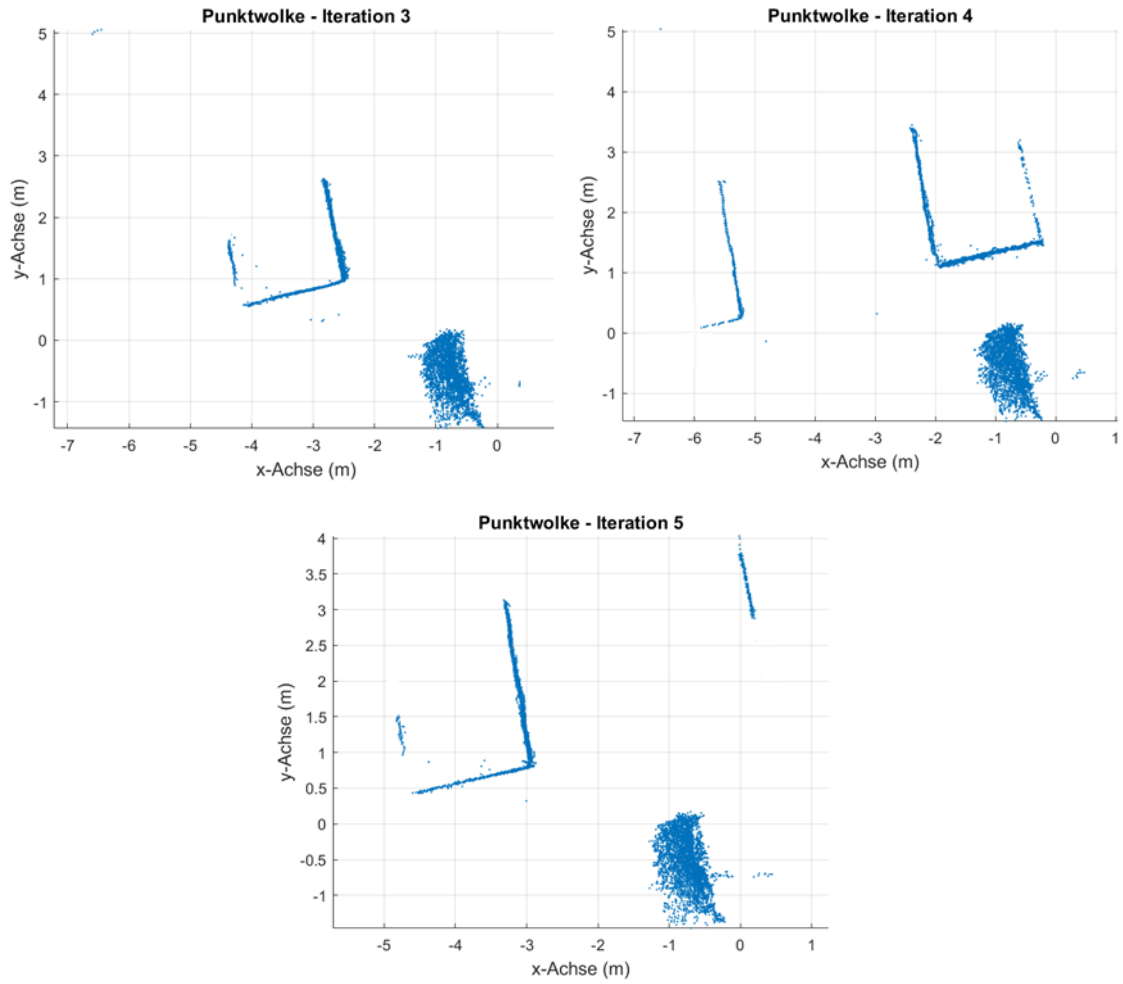


Abbildung 3.23 Aufgenommene Punktwolken während der Vorbeifahrt

Der Algorithmus hat Punktwolke 4 als beste Punktwolke bewertet, um den Vorbeifahrtsabstand zu berechnen. Anschließend wird eine Clusteranalyse mit dem DBSCAN-Algorithmus durchgeführt, um zwischen der Motorhaube und den Parklückenverstelleinheiten zu unterscheiden. Basierend darauf werden die relevanten Eckpunkte der dem Fahrzeug zugewandten Seite der Parklückenverstelleinheit ermittelt, um diese Seite durch eine Regressionsgerade darzustellen.

Um den Abstand zwischen dem VUT und der PLV zu berechnen, kann Gleichung (3.21) verwendet werden, die den Abstand zwischen einer Geraden und einem Punkt bestimmt. In diesem Fall ist der Punkt der Ursprung des Koordinatensystems, der sich im Sensor befindet. Die Regressionsgerade

repräsentiert die Wand der Parklückenverstelleinheit. In Abbildung 3.24 ist diese Auswertung dargestellt. Die rote Linie stellt die Regressionsgerade dar, welche die Wand der PLV abbildet, während die blau gestrichelte Linie den berechneten Abstand repräsentiert.

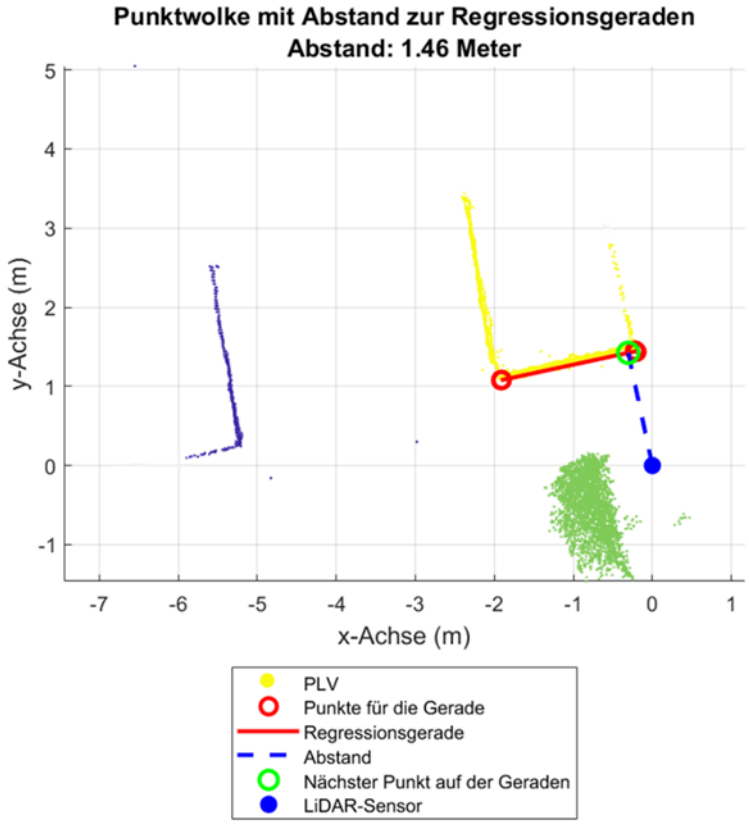


Abbildung 3.24 Abstand mit dem das VUT an der Parklücke vorbeifährt

Um den tatsächlichen Abstand zwischen dem Fahrzeug und der Parklückenverstelleinheit zu erhalten, muss der Abstand zwischen dem Sensor und der rechten Fahrzeugseite des VUT subtrahiert werden. Dieser Abstand ist bekannt und ergibt sich aus der Position des Sensors am Fahrzeug.

Es ist anzumerken, dass dieser Abstand beim Vorbeifahren nicht zwangsläufig konstant eingehalten wird, da der Fahrer möglicherweise leichte Abweichungen in der Spurführung verursacht. Daher dient der ermittelte Wert primär als Orientierungshilfe und für die Dokumentation.

Nachdem das VUT den automatisierten Einparkvorgang abgeschlossen hat, muss der Abstand zur rechten Parklückenverstelleinheit gemessen werden. Dies erfolgt nahezu analog zur Berechnung der Distanz beim Vorbeifahren. Nach

Abschluss des Parkvorgangs wird eine weitere LiDAR-Aufnahme durchgeführt. In dieser Punktwolke ist die Parklückenverstelleinheit deutlich erkennbar.

Durch eine erneute Clusteranalyse wird zwischen der Motorhaube und der Parklückenverstelleinheit unterschieden. Anschließend wird eine Regressionsgerade für die relevante Seite der Verstelleinheit erstellt, und der Abstand zum Sensormittelpunkt wird berechnet. Auf diese Weise wird der finale Abstand zwischen dem Fahrzeug und der Verstelleinheit bestimmt. Abbildung 3.25 zeigt den gemessenen Abstand zwischen dem Fahrzeug und der rechten Parklückenverstelleinheit am Ende des Parkvorgangs. Die rote Linie repräsentiert die Regressionsgerade, die entlang der dem Fahrzeug zugewandten Seite der Parklückenverstelleinheit gebildet wurde. Der grüne Kreis markiert den Punkt auf der Regressionsgerade, der dem Fahrzeug am nächsten liegt.

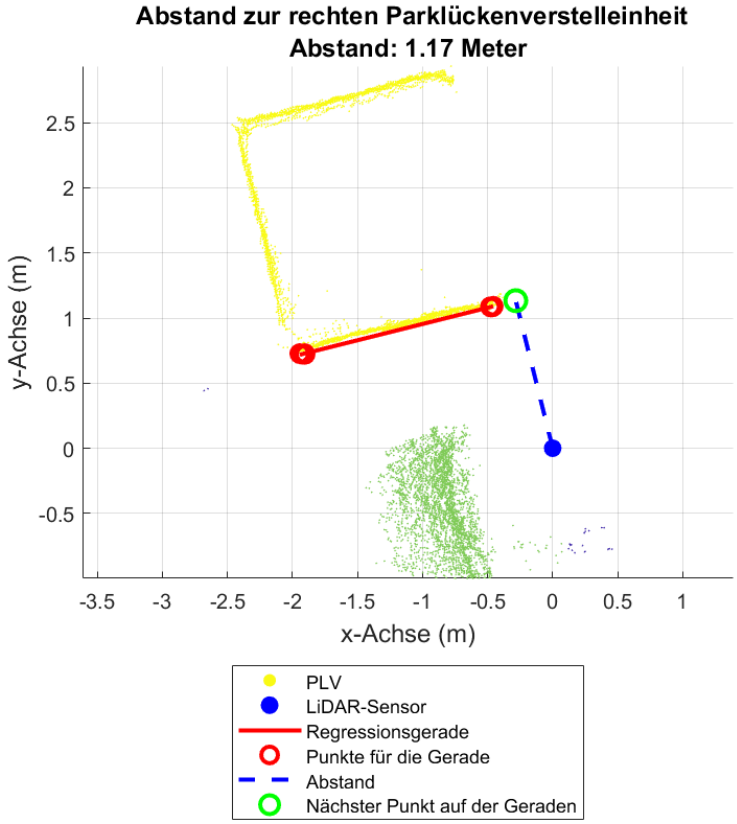


Abbildung 3.25 Abstand zwischen VUT und PLV am Ende des Parkvorgangs

Auch bei dieser Berechnung muss der Abstand zwischen dem Sensor und der rechten Fahrzeugseite des VUT subtrahiert werden, um den tatsächlichen seitlichen Abstand zu ermitteln.

Da nun alle relevanten Abstände bestimmt wurden, kann auch der Abstand zur linken Parklückenverstelleinheit berechnet werden. Dies ermöglicht eine Bewertung, ob das VUT zentriert oder asymmetrisch in der Parklücke positioniert ist. Die Berechnung und Bewertung erfolgen direkt in der Testumgebung ecu.test.

3.7 Implementierung in ecu.test

3.7.1 Technische Umsetzung der Implementierung

In Kapitel 2.2 wurde bereits erläutert, dass es sich bei ecu.test um eine auf Python basierende Testsoftware handelt. Dementsprechend ist die Implementierung einer Datenauswertung mittels Python gut integrierbar. In dieser Diplomarbeit wurde der Algorithmus zur Auswertung der LiDAR-Daten jedoch in MATLAB entwickelt. Der Grund dafür liegt in der großen Auswahl an Toolboxen, die die Erstellung des Algorithmus erheblich vereinfachen. Ebenso wurde in Kapitel 2.2 die Einbindung von MATLAB in ecu.test erläutert, die sich hauptsächlich auf die Verwendung von Simulink konzentriert. Um die benötigten Variablen, wie z.B. die Parklückenbreite von MATLAB nach ecu.test zu übertragen wurde ein Workaround geschaffen. Dieser Workaround basiert auf einem Python-Skript. Ziel ist es, dass der Prüfingenieur während des Tests der Einparkautomatik nicht zwischen ecu.test und MATLAB wechseln muss.

Das Python-Skript startet MATLAB im Batch-Modus. In diesem Modus werden Skripte oder Funktionen im Hintergrund ausgeführt, ohne dass eine aktive Sitzung erforderlich ist. Wenn der Testfall "Parking-Assistant" so weit fortgeschritten ist, dass die LiDAR-Daten ausgewertet werden können, wird das Python-Skript aufgerufen. Innerhalb der Python-Umgebung wird das entwickelte MATLAB-Skript ausgeführt und wertet die LiDAR-Daten aus. MATLAB schreibt dann die benötigten Variablen, wie z.B. die Breite der Parklücke oder den Abstand bei seitlicher Vorbeifahrt, in ein Textdokument, welches vom Python-Skript ausgelesen wird und die entsprechenden Variablen an ecu.test übergibt.

Abbildung 3.26 stellt den Berechnungsschritt in ecu.test dar. Unter „Parameter“ wird das Python-Skript mit dem Namen „run_matlab_script_part1“ aufgerufen. Sobald die Berechnungen abgeschlossen sind, werden die Ergebnisse in den

Variablen „Width_Parking_Space“ und „Distance_drive_by“ im ecu.test Workspace abgespeichert.

#	Aktion / Name	Parameter	Erwartung / Wert
1	Berechnung	user.run_matlab_script_part1.execute_matlab_script()	|-> (Width_Parking_Space, Distance_drive_by)

Abbildung 3.26 Aufruf des Python-Skriptes in ecu.test

Für ein besseres Verständnis stellt Abbildung 3.27 die Toolchain dar.

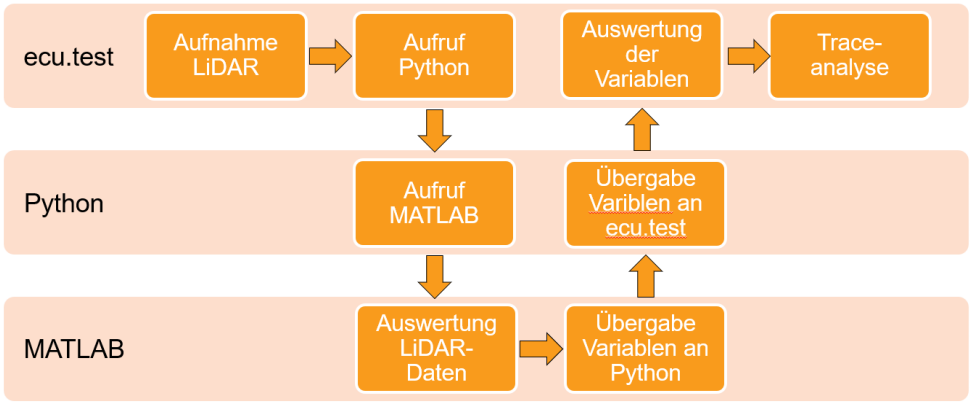


Abbildung 3.27 Toolchain

Da zur korrekten Ausführung des MATLAB-Skripts zunächst alle LiDAR-Daten, d.h. die Punktwolken während der Vorbeifahrt an der Parklücke sowie die Punktwolke nach Beendigung des Parkvorgangs aufgenommen werden müssen, kann der Aufruf von MATLAB im Batch-Modus erst am Ende des Testfalls erfolgen. Dadurch entsteht zwangsläufig eine Leerlaufzeit, in der der Prüfingenieur auf die Berechnung warten muss, da der SLAM-Prozess je nach Anzahl der Punktwolken einige Zeit in Anspruch nehmen kann. Um Leerlaufzeiten zu vermeiden, wird die MATLAB-Auswertung in zwei Teile aufgeteilt. Der erste Teil umfasst die Berechnung der Parklückenbreite und die Berechnung der Vorbeifahrtdistanz. Der zweite Teil beinhaltet die Berechnung des Abstands zur rechten PLV.

Die groben Programmablaufpläne für die beiden separaten MATLAB-Skripte können Abbildung 3.28 entnommen werden. Im Anhang sind exemplarisch die Ablaufpläne für die SLAM-Funktion (Anlage B-2) und die Funktion zum Extrahieren der Punktwolken (Anlage B-1) einsehbar.

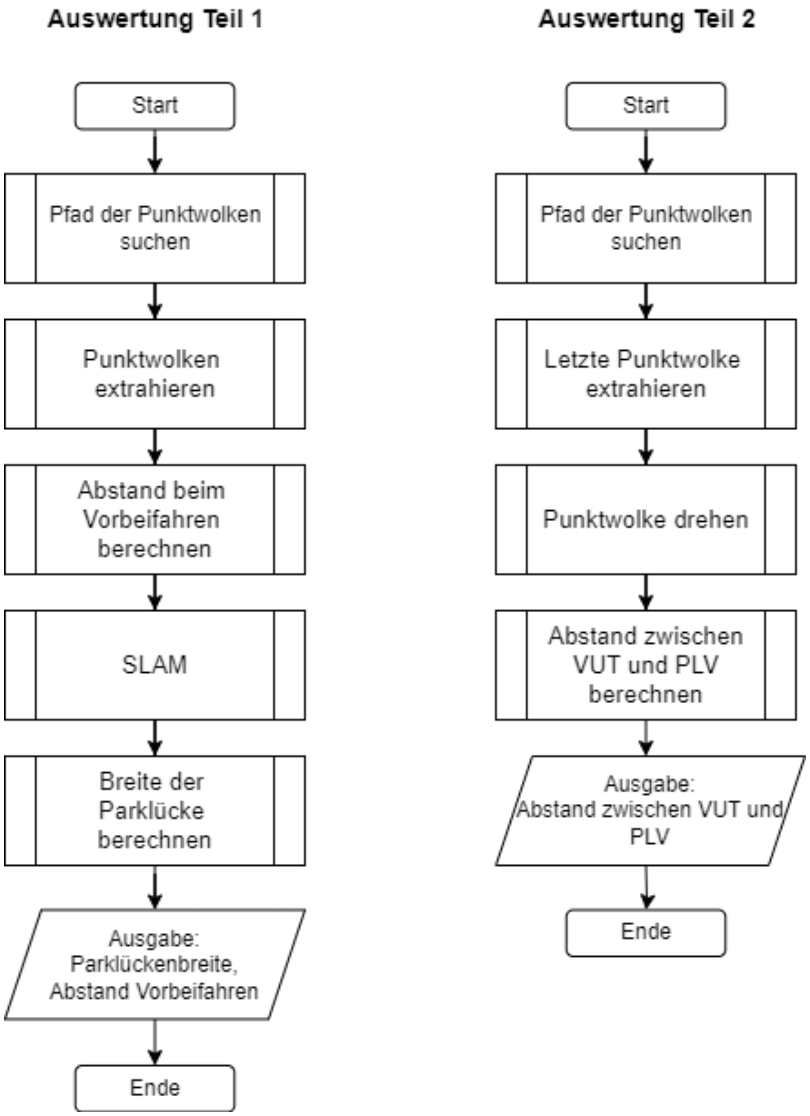


Abbildung 3.28 Programmablaufpläne für beide MATLAB-Skripte

Der Vorteil dieses Splits ist, dass der erste Teil während des laufenden Testfalls durchgeführt werden kann. Da die benötigten LiDAR-Punktwolken für den SLAM, Parklückenbreite und Vorbeifahrdistanz bereits vorliegen, wenn das VUT an der Parklücke vorbeigefahren ist. Die Auswertung kann also bereits erfolgen, während das VUT den automatisierten Einparkvorgang durchführt. Für jeden Teil des MATLAB-Skriptes wird ein eigenes Python-Skript benötigt. Die beiden Python-Skripte sind identisch, abgesehen vom Namen des aufgerufenen MATLAB-Skriptes.

Normalerweise wird ein Testfall von ecu.test sequenziell abgearbeitet. Dies würde bedeuten, dass beim Aufruf des ersten MATLAB-Skripts der Testfall nach dem Passieren der Parklücke pausieren müsste, bis das Skript abgearbeitet ist.

Dies würde jedoch erneut zu Leerlaufzeiten führen, da die Berechnung aufgrund der Komplexität der Punktwolken etwas Zeit in Anspruch nimmt.

Die Lösung für dieses Problem ist eine in ecu.test integrierte Funktion, mit der innerhalb eines Hauptpackages ein paralleles Package gestartet werden kann. In diesem parallelen Package („Calculation_Part1“) wird der erste Teil des MATLAB-Skripts durch das Python-Skript aufgerufen, während das primäre Package weiterhin abgearbeitet wird. Damit kann die Leerlaufzeit für den SLAM-Algorithmus vermieden werden.

Die Aufzeichnung der LiDAR-Punktwolken während der Vorbeifahrt an der Parklücke wird ebenfalls in ein paralleles Package („LiDAR_Capture“) ausgelagert. Der Grund hierfür liegt darin, dass ohne diese Parallelisierung während der Vorbeifahrt ausschließlich die LiDAR-Daten in ecu.test erfasst werden könnten. Durch die Verwendung eines parallelen Packages ist es hingegen möglich, zusätzlich weitere Testschritte im Hauptpackage simultan durchzuführen, wodurch eine effizientere Testausführung gewährleistet wird.

Für ein besseres Verständnis der zeitlichen Abfolge, können die Startzeitpunkte der drei Packages aus Abbildung 3.29 entnommen werden.

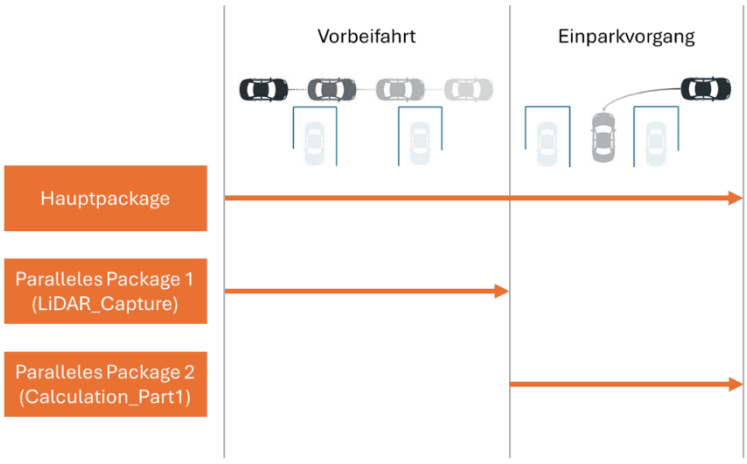


Abbildung 3.29 Sequenzierung der Packages

Für den zweiten Teil der MATLAB-Auswertung, der die Berechnung des seitlichen Abstands zur PLV umfasst, ist mit einer kurzen Leerlaufzeit zu rechnen.

Der gesamte Testfall kann in Anlage A-1 eingesehen werden. Die Programmablaufpläne sind den Anlage B-3, Anlage B-4 und Anlage B-5 zu entnehmen.

Nach Abschluss der Auswertung der LiDAR-Daten, liegen in ecu.test drei Variablen vor. Die Breite der Parklücke, der Abstand mit dem das VUT an der Parklücke vorbeigefahren ist und der seitliche Abstand zur rechten PLV nach dem Einparken. Um nun auf den Abstand zur linken PLV zu schließen, wird noch der Abstand zwischen LiDAR-Sensormittelpunkt und der rechten Fahrzeugseite (Kotflügel) benötigt sowie die Fahrzeugbreite ohne Außenspiegel. Durch eine Benutzereingabe nach Start des Testfalls werden die fehlenden Variablen abgefragt. In Abbildung 3.30 ist beispielhaft die Eingabe der Fahrzeugbreite abgebildet.

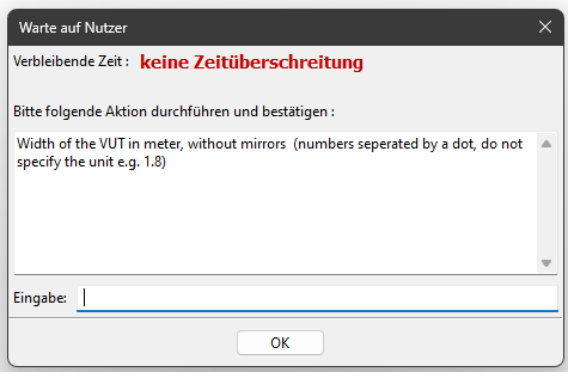


Abbildung 3.30 Benutzereingabe zu Beginn des Testfalls

Nachdem nun alle benötigten Größen bekannt sind, kann die Berechnung der linken Distanz zur PLV durchgeführt werden. Dadurch kann festgestellt werden, ob das Fahrzeug mittig oder versetzt in der Parklücke steht. Abbildung 3.31 zeigt die Berechnungsschritte in ecu.test.

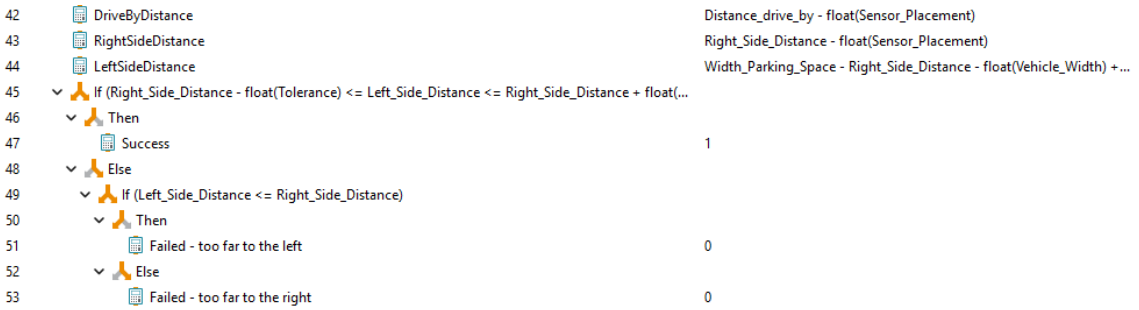


Abbildung 3.31 Berechnungsschritte für den Abstand zur linken PLV

3.7.2 Datei- und Ordnerstruktur im Workspace

Für die Erweiterung des Testfalls „Parking-Assistant“ soll nun übersichtlich dargestellt werden, welche Dateien und Skripte im ecu.test Workspace benötigt

werden. Anlage A-2 zeigt die Ordnerstruktur eines neuen Workspace. Für die Implementierung der beiden MATLAB- und Python-Skripte sind die zwei Ordner „TestReports“ und „UserPyModules“ wichtig.

In den Ordner „TestReports“ müssen die beiden MATLAB-Skripte abgelegt werden. Eine weitere Anpassung dieser Skripte ist nicht notwendig.

Der Ordner „UserPyModules“ ist der Ablageort für die zwei Python-Skripte. Die Python-Skripte passen sich automatisch der Struktur des Dateisystems an, solange die relative Struktur des Workspace erhalten bleibt. Dies stellt sicher, dass keine weiteren Anpassungen der im Python-Skript hinterlegten Pfade notwendig sind, wenn das Projekt auf verschiedenen Rechnern oder in unterschiedlichen Umgebungen verwendet wird.

Im Ordner „Packages“ müssen das Hauptpackage (Inspection_Parking_Assistant_Extended) und die beiden parallel ablaufenden Packages (LiDAR_Capture, Calculation_Part1) abgelegt werden.

Tabelle 3.1 stellt zusammenfassend übersichtlich dar, welche Dateien für einen Ablauf benötigt werden.

Tabelle 3.1 Übersicht der Dateien

Skript	Dateiname	Ablageordner
MATLAB-Skript 1	Parking_Assistant_Part1.m	„TestReports“
MATLAB-Skript 2	Parking_Assistant_Part2.m	„TestReports“
Python-Skript 1	run_matlab_script_part1.py	„UserPyModules“
Python-Skript 2	run_matlab_script_part2.py	„UserPyModules“
Hauptpackage	Inspection_Parking-Assistant_Extended.pkg	„Packages“
Paralleles Package 1	LiDAR_Capture.pkg	„Packages“
Paralleles Package 2	Calculation_Part1.pkg	„Packages“

Die MATLAB und Python-Skripte, sowie die Packages sind auf dem Datenträger hinterlegt. Die Struktur des Datenträgers kann Anlage D-1 entnommen werden.

4 Test und Genauigkeit des Verfahrens

In diesem Kapitel wird die Genauigkeit des entwickelten Testverfahrens zur Kartierung und Positionsbestimmung des Fahrzeugs in der Parklücke untersucht.

Zunächst wird die Vorgehensweise zur Bestimmung der Genauigkeit vorgestellt. Anschließend erfolgt eine Bewertung der Abweichungen zwischen den berechneten und tatsächlichen Messwerten.

4.1.1 Genauigkeit der Berechnung der Parklückenbreite

Um den Algorithmus zu testen und die Genauigkeit zu überprüfen, wurden zwölf Messungen durchgeführt. Die ersten sechs Messungen erfolgten bei einer Parklückenbreite von 3,00 m, die Messungen sieben bis zwölf bei einer Breite von 3,30 m. Zur Validierung des Abstands zwischen den beiden PLVs wurde ein Laserentfernungsmesser (Genauigkeit ±1,5 mm) eingesetzt. Theoretisch erkennt der Parklenkassistent Parklücken, die mindestens der Fahrzeugbreite (in diesem Fall 1,83 m) plus 0,8 m entsprechen. Doch Vorversuche haben gezeigt, dass die Erkennung bei Parklücken unter 2,90 m Breite nicht zuverlässig funktioniert. Um verlässliche Ergebnisse zu erzielen und mögliche Fehlmessungen zu vermeiden, wurde daher eine Parklückenbreite von 3,00 m als untere Grenze gewählt. Dies stellt sicher, dass die Parklücke verlässlich erkannt wird. Die Breite von 3,30 m wurde hinzugefügt, um zu untersuchen, wie sich die Genauigkeit bei einer etwas größeren Lücke verhält.

Es ist hervorzuheben, dass alle gemessenen Breiten deutlich über der gesetzlichen Mindestbreite von 2,30 m für Parklücken liegen. Aufgrund der zunehmenden Größe moderner Fahrzeuge empfiehlt die Forschungsgesellschaft für Straßen- und Verkehrswesen, welche die Vorgaben in den „Richtlinien für die Anlage von Stadtstraßen“ festlegt, künftig eine Mindestbreite von 2,65 m für Parklücken [58].

Abbildung 4.1 zeigt die Messwerte für einen Abstand von 3,00 m. Auf der x-Achse sind die jeweiligen Messungsnummern dargestellt, während die y-Achse die ermittelte Breite der Parklücke abbildet. Die berechneten Werte (blaue Punkte) zeigen eine geringe Streuung um die tatsächliche Breite, wobei der Mittelwert (grüne Linie) bei 3,01 m liegt. Diese Ergebnisse deuten darauf hin, dass der

Algorithmus die Breite der Parklücke im Durchschnitt mit einer sehr geringen Abweichung bestimmt, was auf eine hohe Genauigkeit der Methode hinweist. Eine systematische Tendenz zur Über- oder Unterschätzung der Breite ist anhand der vorliegenden Daten nicht erkennbar. Stattdessen erscheinen die Schwankungen der Messwerte zufällig verteilt. Diese zufällige Verteilung der Abweichungen könnte auf die Berechnungsmethode durch den SLAM-Algorithmus zurückzuführen sein.

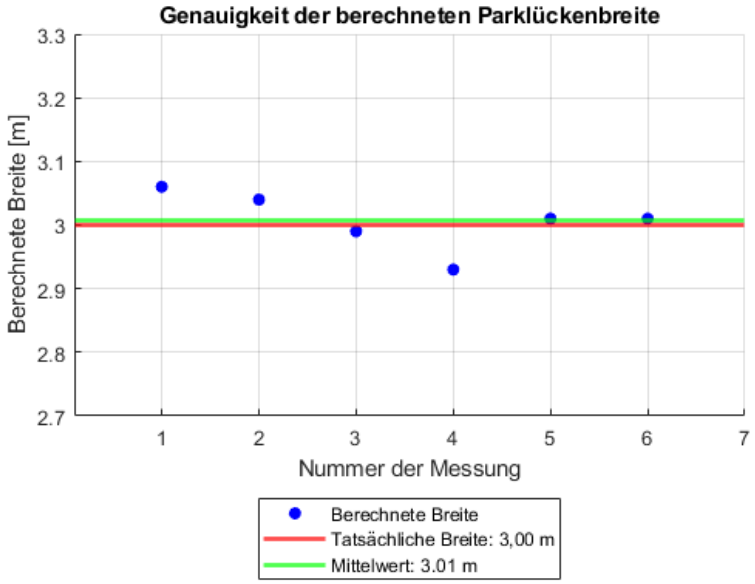


Abbildung 4.1 Genauigkeit bei einer Parklückenbreite von 3,00 m

Im Vergleich zur vorherigen Messreihe zeigt Abbildung 4.2 für die tatsächliche Breite von 3,30 m eine leichte Tendenz zur Unterschätzung der berechneten Breiten. Der Mittelwert der Messungen liegt bei 3,23 m, was eine durchschnittliche Abweichung von -0,07 m darstellt. Diese Abweichungen sind zwar größer als bei der 3,00 m-Messreihe.

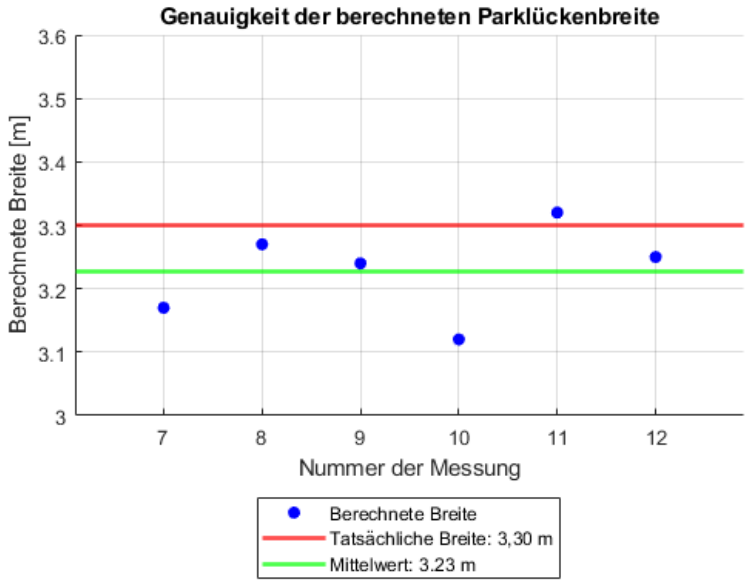


Abbildung 4.2 Genauigkeit bei einer Parklückenbreite von 3,30 m

4.1.2 Genauigkeit der Berechnung des rechten Abstandes

Nachdem das VUT den Parkvorgang erfolgreich durchgeführt hat, wird im Anschluss ein LiDAR-Bild von der rechten PLV aufgenommen, aus dessen Daten der rechte Abstand berechnet wird. Um diesen berechneten Abstand mit dem realen Abstand zu vergleichen, wurde dieser händisch mittels Laserentfernungsmesser vermessen. Als Referenzpunkt diente dabei der vordere Kotflügel des Fahrzeugs (siehe Anlage A-3).

Abbildung 4.3 zeigt das Diagramm, das den Vergleich zwischen der berechneten und der gemessenen Distanz veranschaulicht. Auf der Abszisse sind die berechneten Distanzen aus den LiDAR-Daten in Metern aufgeführt, während die Ordinate die händisch gemessenen Distanzen zeigt. Die gestrichelte rote Linie symbolisiert die Linie der perfekten Übereinstimmung, bei der die berechneten und gemessenen Distanzen identisch sind. Liegen die Datenpunkte unterhalb dieser Linie, wurde der Abstand größer berechnet, als er in Wirklichkeit ist; liegen sie oberhalb, hat der Algorithmus den Abstand zu kurz berechnet.

Die Datenpunkte im Diagramm sind in zwei Gruppen unterteilt: Blaue Punkte repräsentieren Messungen bei einer Parklückenbreite von 3,00 m, während rote Punkte Messungen bei einer Breite von 3,30 m darstellen. Die Nummerierung gibt die jeweiligen Messungsnummern an.

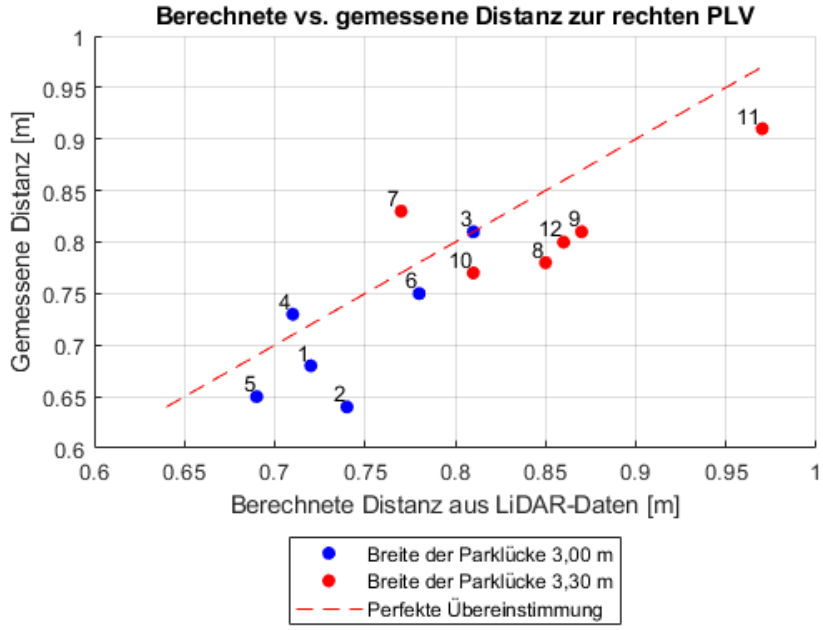


Abbildung 4.3 Gegenüberstellung der gemessenen und berechneten Abstände zur rechten PLV

Bei näherer Betrachtung fällt auf, dass die Punkte sowohl bei der Parklücke von 3,00 m als auch bei 3,30 m nicht immer exakt auf der Linie der perfekten Übereinstimmung liegen, sondern die berechneten Daten im Mittel um 3,5 cm zu groß ausfallen. Mögliche Ursachen dafür können Messungenauigkeiten des Ouster-LiDAR sein; im Datenblatt wird im Bereich von 0,3 bis 1 Meter eine Abweichung von ±0,7 cm angegeben. Ein weiterer Grund ist, dass vom berechneten Wert noch der Abstand vom Sensormittelpunkt zur Fahrzeugflanke abgezogen werden muss, der von Hand gemessen wird und somit zu Ungenauigkeiten führen kann.

Bei der händischen Messung des Abstandes zwischen PLV und VUT kommen zudem Messfehler des Laserentfernungsmessers.

Die Punkte der Messungen bei einer Parklückenbreite von 3,3 m weisen meist höhere x- und y-Werte auf, da die Parklücke breiter war und somit auch ein größerer Abstand zwischen Fahrzeug und PLV vorhanden sein kann**.** Die Genauigkeit dieser Berechnungsmethode kann als ausreichend bewertet werden, da die durchschnittliche Abweichung lediglich 3,5 cm beträgt.

4.1.3 Genauigkeit der Berechnung des linken Abstandes

Die Vorgehensweise zur Bestimmung der Genauigkeit des linken Abstands ist ähnlich wie im vorherigen Abschnitt beschrieben. Der Abstand zur linken PLV wurde händisch mit dem Laserentfernungsmesser gemessen. Die Berechnung erfolgt über die Breite der Parklücke abzüglich der Fahrzeugbreite und des rechten seitlichen Abstandes. In Abbildung 4.5 sind die Werte dargestellt. Alle Messdaten zeigen, dass der linke Abstand kürzer berechnet wird, als er tatsächlich ist. Im Mittel über alle zwölf Messungen beträgt die Abweichung 17,4 cm. Diese Abweichung resultiert aus den kumulierten Fehlern der vorangegangenen Berechnungen. Wird die Parklücke zu breit oder zu schmal berechnet, fließt dieser Fehler mit ein, ebenso wenn der rechte Abstand nicht genau berechnet wird.

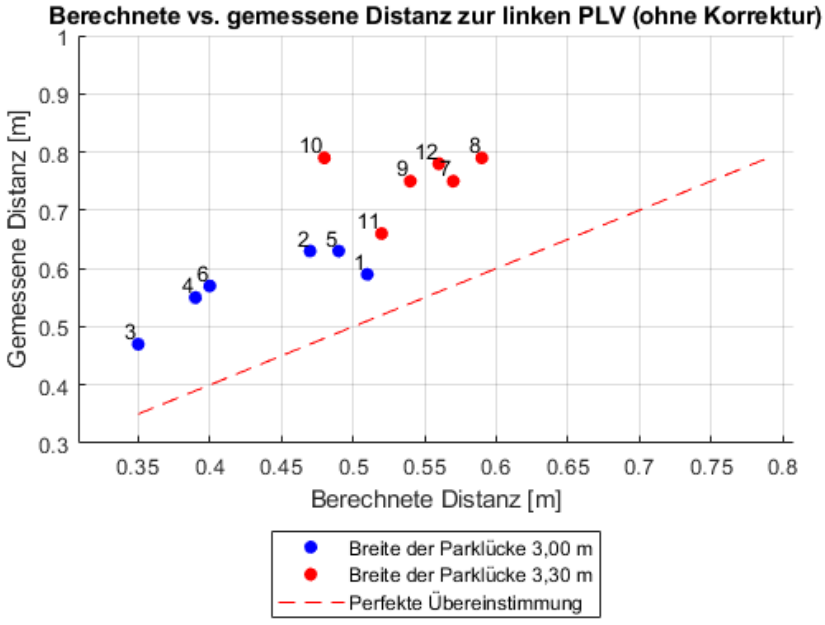


Abbildung 4.4 Gegenüberstellung der gemessenen und berechneten Abstände zur linken PLV

Ein Korrekturfaktor kann nun genutzt werden, um die Werte näher an die Linie der perfekten Übereinstimmung anzugleichen. Der Korrekturfaktor ist eine Zahl, die verwendet wird, um eine systematische Abweichung in einer Messung zu korrigieren. Er kann auf empirischen Daten oder Modellen basieren. Aus den zwölf Messwerten wurde als Mittelwert eine Abweichung von 17,4 cm bestimmt. Der Mittelwert wurde als Maß gewählt, da die vorliegenden Messwerte keine signifikanten Ausreißer oder großen Streuungen aufweisen. Dieser Wert wird nun auf die zu kurz berechneten Abstände addiert. Abbildung 4.5 zeigt die

Gegenüberstellung unter Berücksichtigung des Korrekturfaktors. Die Werte liegen nun näher an der perfekten Übereinstimmung.

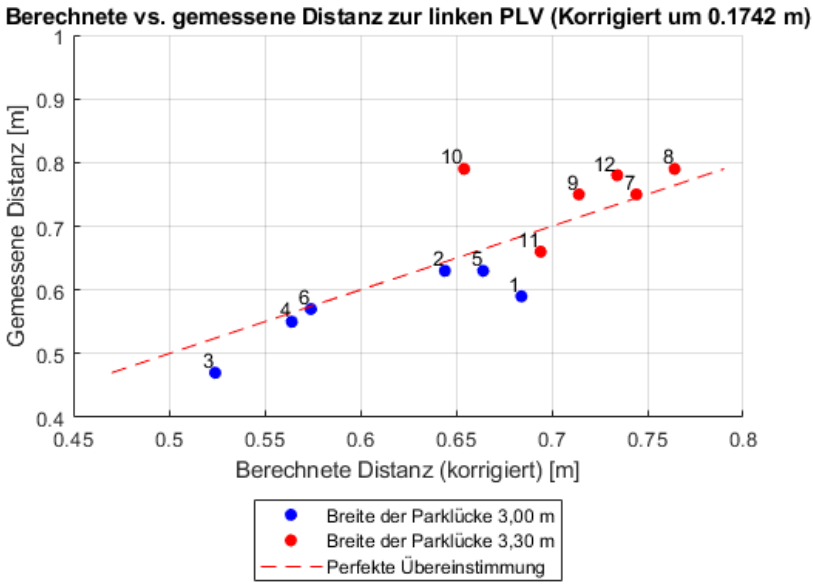


Abbildung 4.5 Gegenüberstellung der gemessenen und berechneten Abstände zur linken PLV mit Korrektur

Auf den berechneten Abstand zur rechten PLV wurde bewusst kein Korrekturfaktor angewendet, da der Korrekturfaktor für die linke Seite auf dem rechten Abstand basiert. Dies wäre methodisch ungünstig, da die Anwendung eines Korrekturfaktors auf der Grundlage eines bereits korrigierten Wertes potenzielle Fehler und Unsicherheiten verstärken könnte. Es würde zu einer Fehlerfortpflanzung kommen.

4.1.4 Toleranzgrenze

Als abschließende Darstellung sind in Abbildung 4.6 die gemessenen Werte (nicht ausgefüllte Punkte) im Vergleich zu den berechneten und korrigierten Werten (ausgefüllte Punkte) dargestellt. Auf der Abszisse ist der linke Abstand, auf der Ordinate der rechte Abstand aufgetragen. Liegt ein Punkt genau auf der Geraden der perfekten Übereinstimmung, bedeutet dies, dass der Parklenkassistent das Fahrzeug genau mittig positioniert hat (perfekt geparkt). Zusätzlich wurde eine Toleranzgrenze zur perfekten Übereinstimmung eingezeichnet, in diesem Fall ±0,1 m. Diese Grenze ist wichtig, da nicht davon ausgegangen werden kann, dass das VUT perfekt einparkt. Liegt der Messpunkt

innerhalb der Toleranz, gilt der Testfall als erfüllt; der Parklenkassistent hat das Fahrzeug ausreichend genau in die Parklücke manövriert.

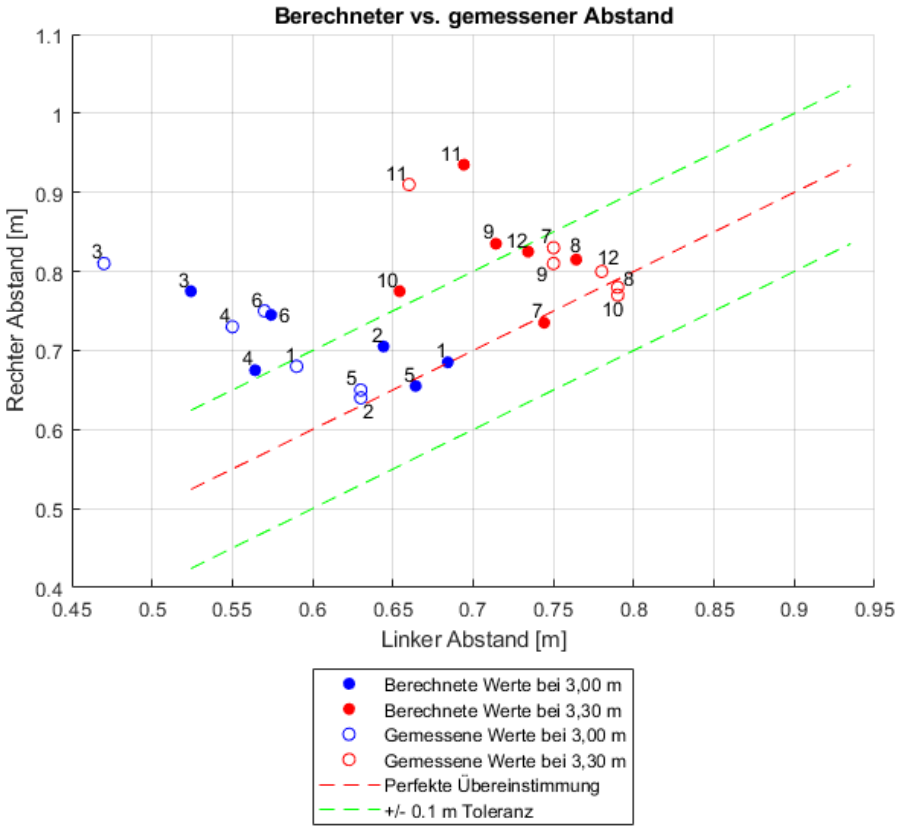


Abbildung 4.6 Vergleich der berechneten und gemessenen Werten

Beim Betrachten der Punkte ist zu sehen, dass nur bei Messung 9 und 10 der Testfall als unerfüllt klassifiziert wurde, obwohl in der Realität die Messungen innerhalb der Toleranz liegen würden. Dies deutet darauf hin, dass der Algorithmus in 10 von 12 Fällen zuverlässig arbeitet, jedoch in einigen Fällen aufgrund von Messungenauigkeiten oder systematischen Fehlern zu einer fehlerhaften Klassifizierung führen kann. Eine Anpassung der Toleranzgrenze oder eine Verbesserung der Messgenauigkeit könnte dazu beitragen, diese Fehlklassifizierungen zu reduzieren.

Die gewünschte Toleranzgrenze wird zu Beginn des Testfalls abgefragt und kann somit vom Prüfingenieur angepasst werden.

4.1.5 Ergänzungen zur Auswertung

Die Berechnung der Parklückenbreite erfolgt auf Grundlage der durch das SLAM-Verfahren erstellten Karte. Da SLAM teilweise auf Wahrscheinlichkeitsmodellen basiert, kann die kartierte Darstellung der Parklücke variieren und ist somit nicht immer identisch. Dies führt dazu, dass die berechnete Breite – bei erneuter Berechnung – schwanken kann, obwohl die zugrunde liegende LiDAR-Punktwolke unverändert bleibt.

In Anlage C-2 können alle Messwerte eingesehen werden. Das MATLAB-Skript für die Auswertung ist auf dem Datenträger hinterlegt.

5 Zusammenfassung und Ausblick

Die vorliegende Diplomarbeit widmete sich der Entwicklung und Implementierung neuer Testverfahren für automatisierte Fahrfunktionen, speziell für den Parklenkassistenten. Angesichts der steigenden Komplexität moderner Fahrzeuge und der fortschreitenden Automatisierung ist die Validierung solcher Systeme von Bedeutung, um Sicherheit und Zuverlässigkeit im Straßenverkehr zu gewährleisten.

Im Zuge dieser Arbeit wurde ein bestehender Testfall in der Testautomatisierungssoftware ecu.test erweitert und optimiert. Durch die Nutzung eines LiDAR-Sensors und die Anwendung des SLAM-Verfahrens gelang es, eine präzise Kartierung der Parklücke zu realisieren. Dies ermöglichte die genaue Berechnung der Parklückenbreite sowie der seitlichen Abstände des Versuchsfahrzeugs zu den benachbarten Parklückenverstelleinheiten. Basierend auf diesen Daten kann nun bewertet werden, ob das Fahrzeug durch den Parklenkassistenten mittig oder versetzt in der Parklücke positioniert wurde.

Die durchgeführten Tests haben gezeigt, dass die Genauigkeit des entwickelten Verfahrens im niedrigen Zentimeterbereich liegt, was für lokale Anwendungen auf Testflächen ausreichend ist. Die Berechnung der Parklückenbreite wies Abweichungen von weniger als 10 % auf, und auch die seitlichen Abstände konnten mit hoher Präzision bestimmt werden. Insbesondere durch die Einführung eines Korrekturfaktors wurde die Genauigkeit bei der Berechnung des linken seitlichen Abstands verbessert.

Zusammenfassend bietet die entwickelte Methodik eine solide Grundlage für die weitere Forschung und Optimierung im Bereich der Testverfahren für den Parklenkassistenten, da sie eine effiziente Bewertung ermöglicht und zur Erhöhung der Sicherheit sowie Zuverlässigkeit solcher Systeme beiträgt.

Für zukünftige Arbeiten ergeben sich mehrere Ansatzpunkte:

1. **Umsetzung in Python**: Die Implementierung des gesamten Algorithmus in Python würde nicht nur die Integration in ecu.test erleichtern, sondern auch die Abhängigkeit von externen Softwarelösungen wie MATLAB

reduzieren. Dies würde den Testprozess beschleunigen und vereinfachen, da alles in einer einheitlichen Umgebung abläuft.

2. **Berechnung der Schrägstellung**: Für die weitere Prüfung des Parklenkassistenten kann die Schrägstellung des VUT in der Parklücke analysiert werden. Dafür muss der Winkelversatz zwischen den PVLs und dem Testfahrzeug berechnet werden. Ein Ansatz wäre ein Normalvektor, der vom Sensor aus in Richtung der rechten PLV aufgespannt wird. Steht das VUT parallel zur PLV, spiegelt der Vektor die kürzeste Strecke wider. Steht das Fahrzeug schräg, ist der Normalvektor nicht mehr der Vektor mit dem kürzesten Abstand, und zwischen diesen beiden Vektoren (Normalvektor und Vektor des kürzesten Abstandes) kann der Winkel der Schrägstellung berechnet werden.

3. **Korrekturzüge**: Die Anzahl der benötigten Korrekturzüge kann für den Prüfingenieur von Interesse sein. Eine Methodik zum Zählen dieser ist im Testfall integriert, konnte aber noch nicht geprüft werden.

4. **Erweiterung der Testreihen**: Durch eine größere Anzahl an Messungen unter variierenden Bedingungen können die Ergebnisse weiter verifiziert und die Genauigkeit des Verfahrens sowie des Korrekturfaktors erhöht werden. Dies beinhaltet auch Tests mit unterschiedlichen Fahrzeugtypen.

5. **Anfälligkeit gegenüber Objekten in der Umgebung**: Befinden sich beispielsweise andere Fahrzeuge oder Personen in der Nähe der Parklückenverstelleinheiten, kann der Algorithmus gestört werden und fehlerhafte Ergebnisse liefern. Daher ist es wichtig, das Verfahren so anzupassen, dass es solche Störungen erkennt und kompensiert.

Diese Arbeit trägt zur Weiterentwicklung von Testmethoden für den Parklenkassistenten bei und schafft damit eine fundierte Grundlage zur Bewertung seiner Funktionalität und Effizienz. Gleichzeitig legt sie eine solide Basis für künftige Forschungsprojekte in diesem Bereich.

.

Literatur- und Quellenverzeichnis

[1] GESETZE IM INTERNET:
Verordnung zur Genehmigung und zum Betrieb von Kraftfahrzeugen mit autonomer Fahrfunktion in festgelegten Betriebsbereichen [online];2022 [Zugriff am: 22. April 2024]. Verfügbar unter: https://www.gesetze-im-internet.de/afgbv/#

[2] Hakuli, S. und Krug, M.:
Virtuelle Integration; In: H. WINNER, S. HAKULI, F. LOTZ und C. SINGER, Hg. *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort.* 3., überarbeitete und ergänzte Auflage. Wiesbaden: Springer Vieweg, 2015, S. 125-138. ISBN 978-3-658-05734-3

[3] O. V.:
SIL and PIL Simulations [online];o. J. [Zugriff am: 26. Juli 2024]. Verfügbar unter: https://de.mathworks.com/help/ecoder/ug/about-sil-and-pil-simulations.html#

[4] Borgeest, K.:
Elektronik in der Fahrzeugtechnik; Hardware, Software, Systeme und Projektmanagement. 5. Auflage Wiesbaden: Springer Vieweg, 2023. ATZ/MTZ-Fachbuch. ISBN 978-3-658-41483-2

[5] O. V.:
What is software-in-the-loop testing? [online];2022 [Zugriff am: 25. Mai 2024]. Verfügbar unter: https://www.aptiv.com/en/insights/article/what-is-software-in-the-loop-testing#

[6] O. V.:
What is vehicle-in-the-loop testing? [online];2022 [Zugriff am: 26. Juli 2024]. Verfügbar unter: https://www.aptiv.com/en/insights/article/what-is-vehicle-in-the-loop-testing#

[7] TRACETRONIC GMBH:
ecu.test [online];2024 [Zugriff am: 22. April 2024]. Verfügbar unter: https://www.tracetronic.de/produkte/ecu-test/#

[8] TRACETRONIC GMBH.:
Anwenderhandbuch für ecu.test, 2024

[9] O. V.:
Absolute Pfade, relative Pfade, UNC-Pfade und URL-Pfade—ArcMap | Dokumentation [online];o. J. [Zugriff am: 27. Juli 2024]. Verfügbar unter: https://desktop.arcgis.com/de/arcmap/latest/tools/supplement/pathnames-explained-absolute-relative-unc-and-url.htm#

[10] Schrocke, M.:
Konzeption und Implementierung einer Kommunikations- Schnittstelle zwischen der Software ECU-TEST und einem Laserscanner zur synchronen Aufzeichnung von Referenzdaten. Masterarbeit Dresden, 2023

[11] O. V.:
ecu.test Schulung "advanced", 16. April 2024

[12] O. V.:
MATLAB [online];o. J. [Zugriff am: 27. Juli 2024]. Verfügbar unter: https://de.mathworks.com/products/matlab.html#

[13] Glöckler, M.:
Simulation mechatronischer Systeme; Grundlagen und Beispiele für MATLAB und Simulink. 3., überarbeitete und erweiterte Auflage Wiesbaden: Springer Vieweg, 2023. Lehrbuch. ISBN 978-3-658-42522-7

[14] O. V.:
MATLAB für Deep Learning [online];o. J. [Zugriff am: 27. Juli 2024]. Verfügbar unter: https://de.mathworks.com/solutions/deep-learning.html#

[15] O. V.:
Design driving scenarios, configure sensors, and generate synthetic data - MATLAB - MathWorks Deutschland [online] [Zugriff am: 29. Juli 2024]. Verfügbar unter: https://de.mathworks.com/help/driving/ref/drivingscenariodesigner-app.html#

[16] WINNER, H., S. HAKULI, F. LOTZ und C. SINGER, Hg.:
Handbuch Fahrerassistenzsysteme; Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort. 3., überarbeitete und ergänzte Auflage Wiesbaden: Springer Vieweg, 2015. ATZ / MTZ-Fachbuch. ISBN 978-3-658-05734-3

[17] Kamwa, A.:
Time-of-Flight- (ToF) -Verfahren mit Lidar-Systemen [online];2022 [Zugriff am: 2. Mai 2024]. Verfügbar unter: https://www.all-electronics.de/elektronik-entwicklung/time-of-flight-und-40tof-und-41-verfahren-mit-lidar-systemen-632.html#:~:text=Das%20grundlegende%20Prinzip%20ist%20einfach,gemessenen%20Laufzeit%20des%20Lichts%20errechnet.#

[18] Kernhof, J., Leuckfeld, J. und Tavano, G.:
LiDAR-Sensorsystem für automatisiertes und autonomes Fahren; In: T. TILLE, Hg. *Automobil-Sensorik 2.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, S. 29-54. ISBN 978-3-662-56309-0

[19] Bi, X.:
Environmental Perception Technology for Unmanned Systems Singapore: Springer Singapore, 2021. ISBN 978-981-15-8092-5

[20] Okunsky, M.V. und Nesterova, N.V.
Velodyne LIDAR method for sensor data decoding [online]. *IOP Conference Series: Materials Science and Engineering,* 2019, **516**, 12018. Verfügbar unter: doi:10.1088/1757-899X/516/1/012018#

[21] Gotzig, H. und Geduld, G.:
LIDAR-Sensorik; In: H. WINNER, S. HAKULI, F. LOTZ und C. SINGER, Hg. *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort.* 3., überarbeitete und ergänzte Auflage. Wiesbaden: Springer Vieweg, 2015, S. 317-334. ISBN 978-3-658-05734-3

[22] Voigt, E. und Kremsreiter, M.:
LiDAR in Anwendung [online];2020 [Zugriff am: 30. Juli 2024]. Verfügbar

unter: https://www.tu-chemnitz.de/physik/EXSE/ForPhySe/LiDAR_in_Anwendung.pdf#

[23] Mazzari, V.:
Was ist die LiDAR-Technologie? [online];2019 [Zugriff am: 26. August 2024]. Verfügbar unter: https://www.generationrobots.com/blog/de/was-ist-die-lidar-technologie/#

[24] Wagner, J.:
Von Licht zu Strom: Die faszinierende Welt der Photodetektoren [online];2023 [Zugriff am: 30. Juli 2024]. Verfügbar unter: https://ipdinstitute.at/photodetektoren/#

[25] TILLE, T., Hg.:
Automobil-Sensorik 2 Berlin, Heidelberg: Springer Berlin Heidelberg, 2018. ISBN 978-3-662-56309-0

[26] O. V.:
LiDAR - Lösungen für den Automotive-Bereich und industrielle Anwendungen [online];o. J. [Zugriff am: 23. Mai 2024]. Verfügbar unter: https://www.forschungsfabrik-mikroelektronik.de/de/unser-angebot/anwendungsbereiche/Transport_and_Smart_Mobility/lidar.html#

[27] Wang, D., Watkins, C. und Xie, H.
MEMS Mirrors for LiDAR: A review [online]. *Micromachines,* 2020, **11**(5). ISSN 2072-666X. Verfügbar unter: doi:10.3390/mi11050456#

[28] Sons, M., Theers, M. und Hofstetter, I.:
SLAM und kartenbasierte Lokalisierung; In: H. WINNER, K.C.J. DIETMAYER, L. ECKSTEIN, M. JIPP, M. MAURER und C. STILLER, Hg. *Handbuch Assistiertes und Automatisiertes Fahren. Grundlagen, Komponenten und Systeme für assistiertes und automatisiertes Fahren.* 4th ed. 2024. Wiesbaden: Springer Fachmedien Wiesbaden; Imprint Springer Vieweg, 2024, S. 511-566. ISBN 978-3-658-38485-2

[29] O. V.:
Was ist SLAM (Simultane Lokalisierung und Kartierung) – MATLAB & Simulink [online] [Zugriff am: 19. August 2024]. Verfügbar unter: https://de.mathworks.com/discovery/slam.html#

[30] O. V.:
Google Maps [online];2024 [Zugriff am: 21. September 2024]. Verfügbar unter: https://www.google.com/maps/@51.0326527,13.7395217,212a,35y,48.35t/data=!3m1!1e3?authuser=0&entry=ttu&g_ep=EgoyMDI0MDkxOC4xIKXMDSoASAFQAw%3D%3D#

[31] O. V.:
pcregistericp - Register two point clouds using ICP algorithm - MATLAB - MathWorks Deutschland [online];o. J. [Zugriff am: 15. September 2024]. Verfügbar unter: https://de.mathworks.com/help/vision/ref/pcregistericp.html#

[32] SIMSANGCHEOL:
Registration [online];2023 [Zugriff am: 15. September 2024]. Verfügbar unter: https://medium.com/@sim30217/registration-85e090a7a87a#

[33] Smistad, E., Falch, T.L., Bozorgi, M., Elster, A.C. und Lindseth, F.
Medical image segmentation on GPUs--a comprehensive review [online]. *Medical image analysis,* 2015, **20**(1), 1-18. Verfügbar unter: doi:10.1016/j.media.2014.10.012#

[34] O. V.:
OS1 Mid-Range High-Resolution Imaging Lidar Datasheet [online];2021 [Zugriff am: 30. Mai 2024]. Verfügbar unter: https://data.ouster.io/downloads/datasheets/datasheet-revd-v2p0-os1.pdf#

[35] O. V.:
Software User Manual [online]*Firmware v2.0.0 for all Ouster sensors;*2021 [Zugriff am: 26. August 2024]. Verfügbar unter: https://data.ouster.io/downloads/software-user-manual/software-user-manual-v2p0.pdf#

[36] VECTOR GMBH:
VN1600 - Netzwerk-Interfaces [online];o. J. [Zugriff am: 22. Juli 2024]. Verfügbar unter: https://www.vector.com/de/de/produkte/produkte-a-z/hardware/netzwerk-interfaces/vn16xx/##

[37] VECTOR GMBH:
VN1600 FactSheet [online]*Bus-Interfaces für CAN, CAN FD, CAN XL, LIN, K-Line, J1708 und IO;*2024

[38] Wieler, J.
Test Einparken auf Knopfdruck: So gut parken Autos automatisch ein [online]. *ADAC,* 30. Juni 2021 [Zugriff am: 14. September 2024]. Verfügbar unter: https://www.adac.de/rund-ums-fahrzeug/ausstattung-technik-zubehoer/assistenzsysteme/autonome-parkassistenten-vergleich/#

[39] VOLKSWAGEN.:
Betriebsanleitung Passat GTE. Parklenkassistent, 2015, S. 262-269

[40] VOLKSWAGEN NEWSROOM:
Parklenkassistent "Park Assist" [online] [Zugriff am: 31. Juli 2024]. Verfügbar unter: https://www.volkswagen-newsroom.com/de/parklenkassistent-park-assist-15396#

[41] Katzwinkel, R., Brosig, S., Schrove, F., Auer, R., Rohlfs, M., Eckert, G., Wuttke, U. und Schwitters, F.:
Einparkassistenz; In: H. WINNER, S. HAKULI, F. LOTZ und C. SINGER, Hg. *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort.* 3., überarbeitete und ergänzte Auflage. Wiesbaden: Springer Vieweg, 2015, S. 841-849. ISBN 978-3-658-05734-3

[42] VOLKSWAGEN NEWSROOM:
Parkassistent "Park Assist Plus" [online];o. J. [Zugriff am: 31. Juli 2024]. Verfügbar unter: https://www.volkswagen-newsroom.com/de/parkassistent-park-assist-plus-3669#

[43] Mendt, F.:
Nürtingen VDI Demos [online];2023 [Zugriff am: 30. Mai 2024]. Verfügbar unter: https://gitlab.com/htw_nives/tracetronic/ecu-test/nuertingen_demo#

[44] O. V.:
pcmerge [online];o. J. [Zugriff am: 8. Juli 2024]. Verfügbar unter: https://de.mathworks.com/help/vision/ref/pcmerge.html?searchHighlight=pcmerge&s_tid=srchtitle_support_results_1_pcmerge#

[45] Grünvogel, S.M.:
Einführung in die Computeranimation; Methoden, Algorithmen, Grundlagen Wiesbaden: Springer Vieweg, 2024. Lehrbuch. ISBN 978-3-658-41988-2

[46] Ngi, A.
Automotive: Wie eine spurgenaue Positionierung funktioniert [online]. *all-electronics.de,* 28. Oktober 2019 [Zugriff am: 12. August 2024]. Verfügbar unter: https://www.all-electronics.de/automotive-transportation/automotive-wie-eine-spurgenaue-positionierung-funktioniert.html#

[47] MAGICMAPS:
Präzise GPS-Messungen mit Hilfe von DGPS und RTK [online];o. J. [Zugriff am: 12. August 2024]. Verfügbar unter: https://www.magicmaps.de/gnss-wissen/praezise-gps-messungen-mit-hilfe-von-dgps-und-rtk/?L=0#

[48] STUDYSMARTER:
Trägheitsnavigation: Prinzip & Anwendungen | *StudySmarter* [online] [Zugriff am: 12. August 2024]. Verfügbar unter: https://www.studysmarter.de/studium/ingenieurwissenschaften/luft-und-raumfahrttechnik/traegheitsnavigation/#

[49] RICHARD DIXON.:
Trends in der Automobil-Sensorik; In: T. TILLE, Hg. *Automobil-Sensorik 2.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, S. 17-28. ISBN 978-3-662-56309-0

[50] Trautmann, T., Cao, S., Jin, Y. und Liu, K.
Design and Experiments of Autonomous Path Tracking Based on Dead Reckoning [online]. *Applied Sciences,* 2023, **13**(1), 317. Verfügbar unter: doi:10.3390/app13010317#

[51] Kiebler, J., Saljanin, M., Müller, S., Todorovic, S., Neubeck, J. und Wagner, A.:
Novel Approach for Vehicle-Self-Localization; In: M. BARGENDE, H.-C. REUSS und A. WAGNER, Hg. *22. Internationales Stuttgarter Symposium.* Wiesbaden: Springer Vieweg, 2022, S. 75-88. ISBN 978-3-658-37010-7

[52] O. V.:
pcfitplane - Fit plane to 3-D point cloud - MATLAB - MathWorks Deutschland [online] [Zugriff am: 16. September 2024]. Verfügbar unter: https://de.mathworks.com/help/vision/ref/pcfitplane.html#

[53] Fahrendholz, J.L. und Brell-Cokcan, S.:
Grundlagen zur automatisierten Baufortschrittsüberwachung mittels Deep Learning basierend auf Punktwolken und Bauinformationsmodellen; In: S. BRELL-ÇOKCAN und R. SCHMITT, Hg. *IoC - Internet of Construction. Informationsnetzwerke zur unternehmensübergreifenden Kollaboration in den Fertigungsketten des Bauwesens.* Wiesbaden: Springer Vieweg, 2024, S. 717-763. ISBN 978-3-658-42543-2

[54] SCHMIDT, J., A. EICHHORN und D. IWASZCZUK, Hg.:
Punkt- und ebenenbasierte Detektion von Ecken und Kanten in Innenraum-Punktwolken. Geschäftsstelle der DGPF, 2023

[55] CARLO TOMASI:
Vector Representation of Rotations [online];o. J. [Zugriff am: 16. September 2024]. Verfügbar unter: https://courses.cs.duke.edu/cps274/fall13/notes/rodrigues.pdf#

[56] GITHUB:
Articles/Rodrigues' rotation/Rodrigues' rotation.md at master · EgoMoose/Articles [online];o. J. [Zugriff am: 16. September 2024]. Verfügbar unter: https://github.com/EgoMoose/Articles/blob/master/Rodrigues'%20rotation/Rodrigues'%20rotation.md#

[57] O. V.:
DBSCAN - MATLAB & Simulink - MathWorks Deutschland [online] [Zugriff am: 16. September 2024]. Verfügbar unter: https://de.mathworks.com/help/stats/dbscan-clustering.html#

[58] FORSCHUNGSGESELLSCHAFT FÜR STRAßEN- UND VERKEHRSWESEN:
Richtlinien für die Anlage von Stadtstraßen; RASt 06. Ausg. 2006, korr. Nachdr. Mai 2012 Köln: FGSV-Verl., 2012. FGSV. 200. ISBN 9783939715214

Eidesstattliche Erklärung

Das vorliegende Dokument wurde an der Hochschule für Technik und Wirtschaft Dresden unter der Leitung von Prof. Dr. rer. nat. Toralf Trautmann und Dipl.-Ing (FH) Franziskus Mendt angefertigt.

Hiermit erkläre ich, dass ich die vorliegende Arbeit zum Thema

„Entwicklung und Implementierung von neuen Testverfahren für automatisierte Fahrfunktionen“

selbstständig und ohne Benutzung anderer Quellen und Hilfsmittel als angegeben angefertigt habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Ferner gestatte ich der Hochschule für Technik und Wirtschaft Dresden, die vorliegende Diplomarbeit unter Beachtung insbesondere urheber-, datenschutz- und wettbewerbsrechtlicher Vorschriften für Lehre und Forschung zu nutzen.

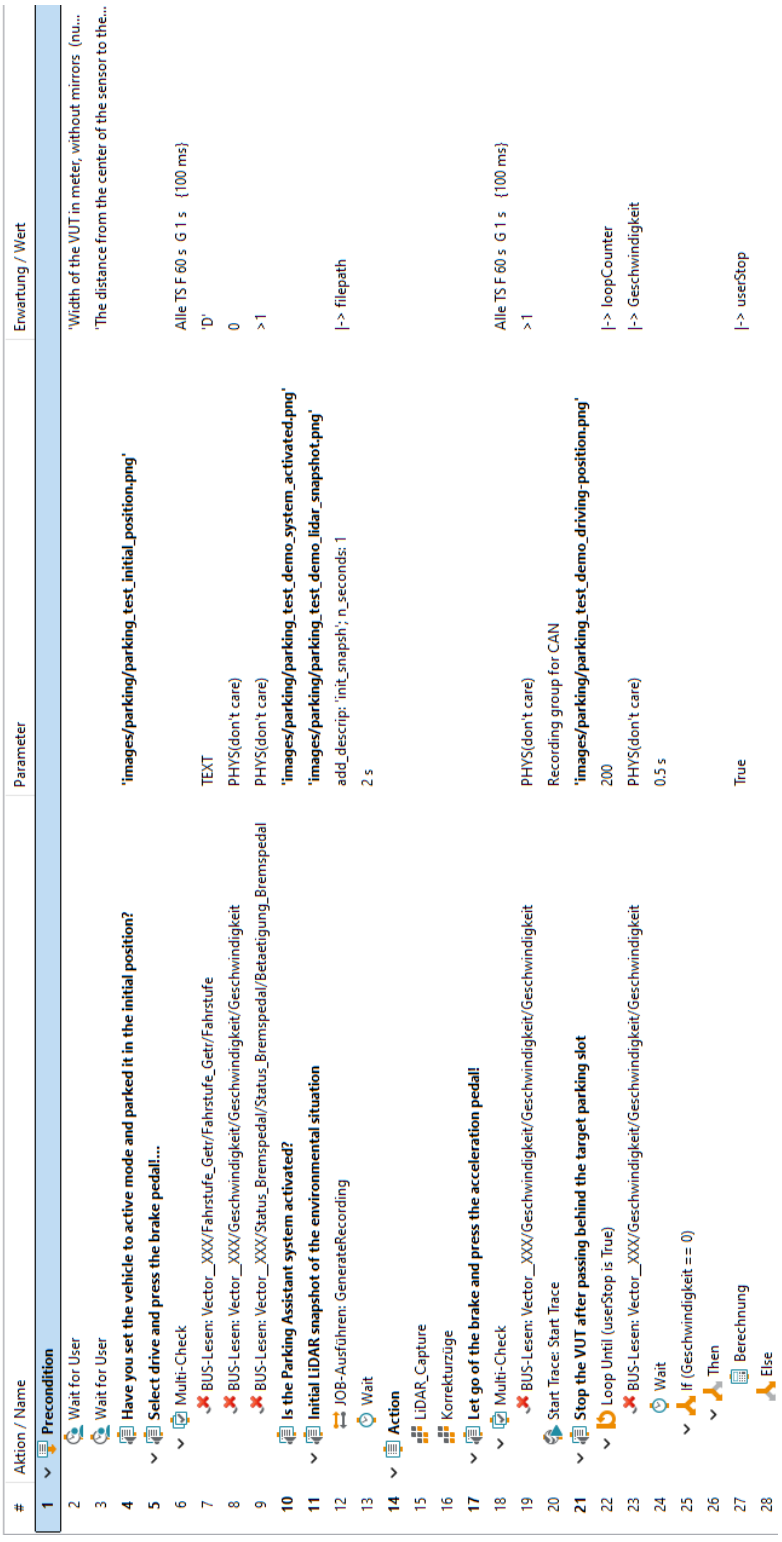
Ort, Datum

Unterschrift

Anlagenverzeichnis

A Abbildungen

Anlage A-1 Testfall in ecu.test



31	**Was the parking space detected by the VUT?**	**'images/parking/parking_test_demo_space-detected-query.png'**		
32	Berechnung_part1			
33	**Select reverse and confirm if the parking manouvre is finished!...**	**'images/parking/parking_test_demo_reverse-position.png'**		
34	**LiDAR snapshot from the final stop position**			
35	JOB-Ausführen: GenerateRecording	add_descrip: 'side'; n_seconds: 1	|-> filepath	
36	Wait	2 s		
37	Stop Trace: Stop Trace	Recording group for CAN		
38	**Postcondition**			
39	Berechnung	user.run_matlab_script_part2.execute_matlab_script()	|-> Right_Side_Distance	
40	Wait	10 s		
41	Aus Inbox lesen: INBOX_Part1		|-> (Width_Parking_Space, Distance_drive_by)	
42	DriveByDistance	Distance_drive_by - float(Sensor_Placement)	|-> Distance_drive_by	
43	RightSideDistance	Right_Side_Distance - float(Sensor_Placement)	|-> Right_Side_Distance	
44	LeftSideDistance	Width_Parking_Space - Right_Side_Distance - float(Vehicle_Width) + ...	|-> Left_Side_Distance	
45	If (Right_Side_Distance - float(Tolerance) <= Left_Side_Distance <= Right_Side_Distance + float(...			
46	Then			
47	Success	1	1 |-> Success	
48	Else			
49	If (Left_Side_Distance <= Right_Side_Distance)			
50	Then			
51	Failed - too far to the left	0	1 |-> Success	
52	Else			
53	Failed - too far to the right	0	1 |-> Success	
54	**Park safely and and apply the handbrake!**			
55	*Bildialog: apiGetSetting('workspacePath') + '\\Images\\LeaveCar.png'*			*'Leave car' - 'Please leave the car' --> CarLeft*
56	Aus Inbox lesen: INBOX_Korrekturzuege		|-> Korrekturzuege	
57	**Turn engine off**			
58	Multi-Check		Alle TS F 60 s G 5 s {100 ms}	
59	BUS-Lesen: Vector_XXX/Feststellbremse/Feststellbremse/Status_Feststellbremse	TEXT	'Bremse fest'	
60	BUS-Lesen: Vector_XXX/Fahrstufe_Getr/Fahrstufe_Getr/Fahrstufe	TEXT	'P'	

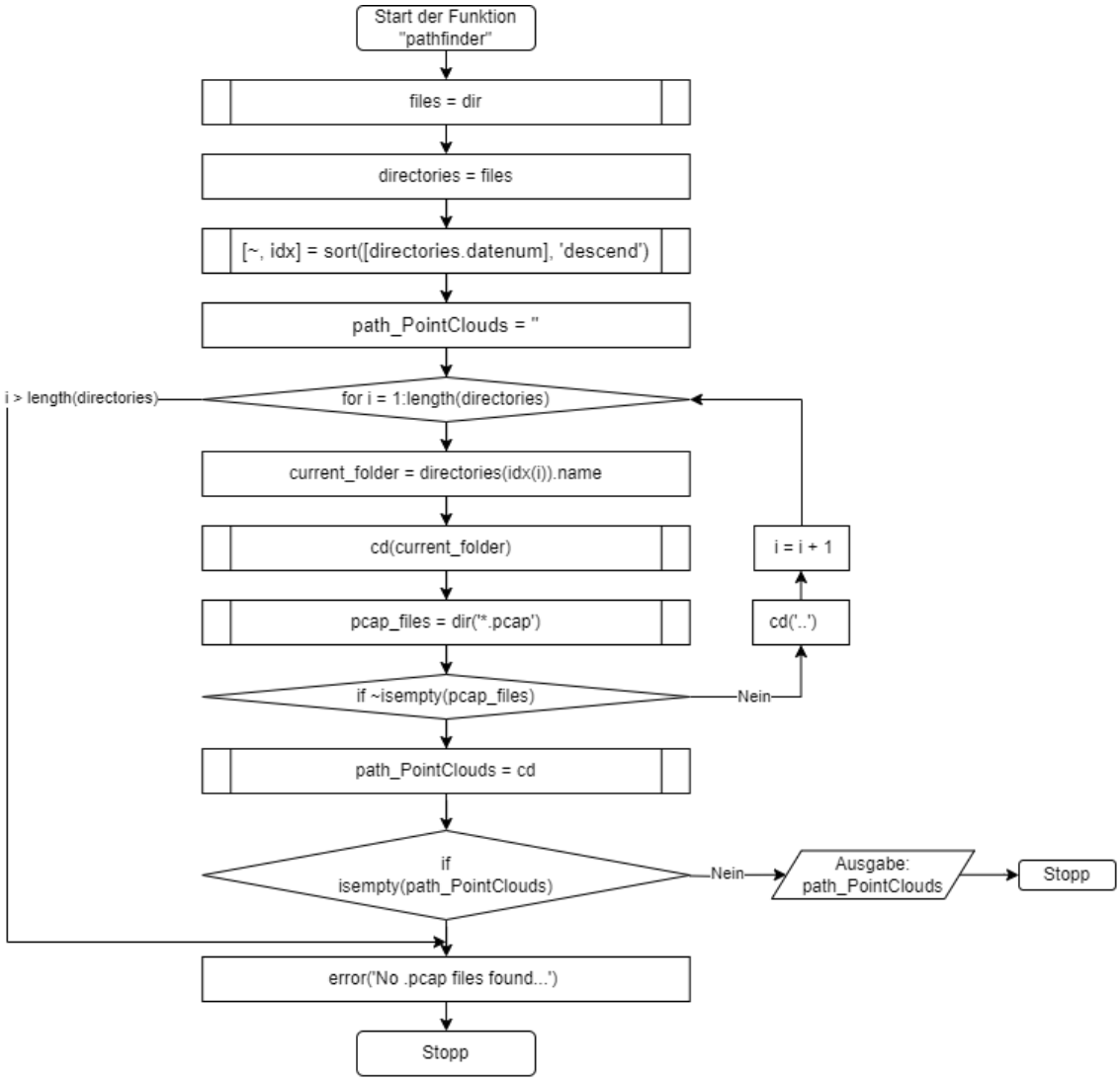
Anlage A-2 Ordnerstruktur des Workspace

.workspace	14.05.2024 13:19	Dateiordner
Configurations	14.05.2024 10:56	Dateiordner
Images	14.05.2024 10:56	Dateiordner
Offline-FIUs	14.05.2024 10:56	Dateiordner
Offline-Models	14.05.2024 10:56	Dateiordner
Offline-SGBDs	14.05.2024 10:56	Dateiordner
Packages	14.05.2024 10:56	Dateiordner
Parameters	14.05.2024 10:56	Dateiordner
ProjectGenerators	14.05.2024 10:56	Dateiordner
Templates	14.05.2024 10:56	Dateiordner
TestReports	14.05.2024 10:56	Dateiordner
TraceStepTemplates	14.05.2024 10:56	Dateiordner
UserPyModules	14.05.2024 10:56	Dateiordner
Utilities	14.05.2024 10:56	Dateiordner

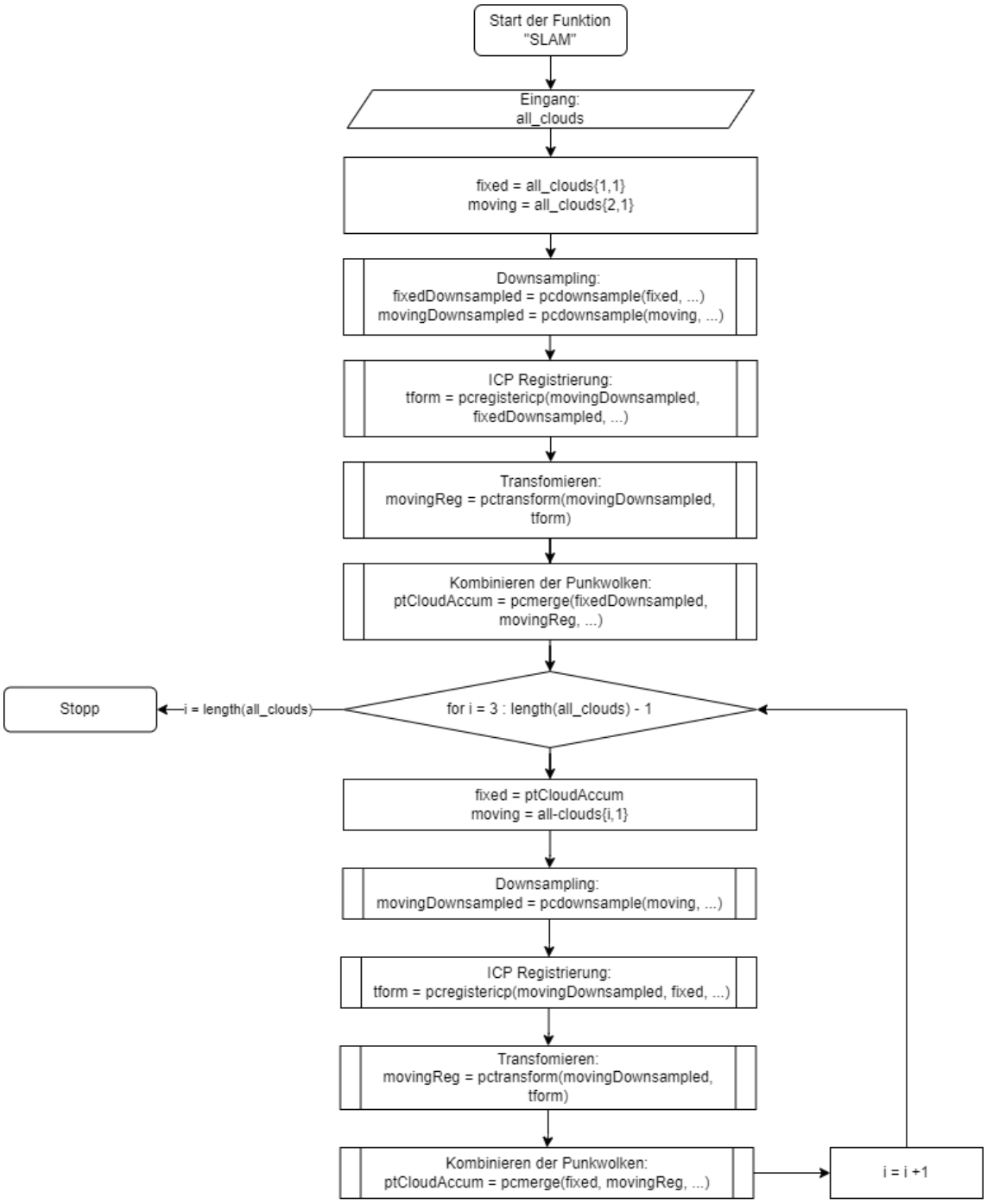
Anlage A-3 Referenzpunkt auf dem Kotflügel

B Programmablaufpläne

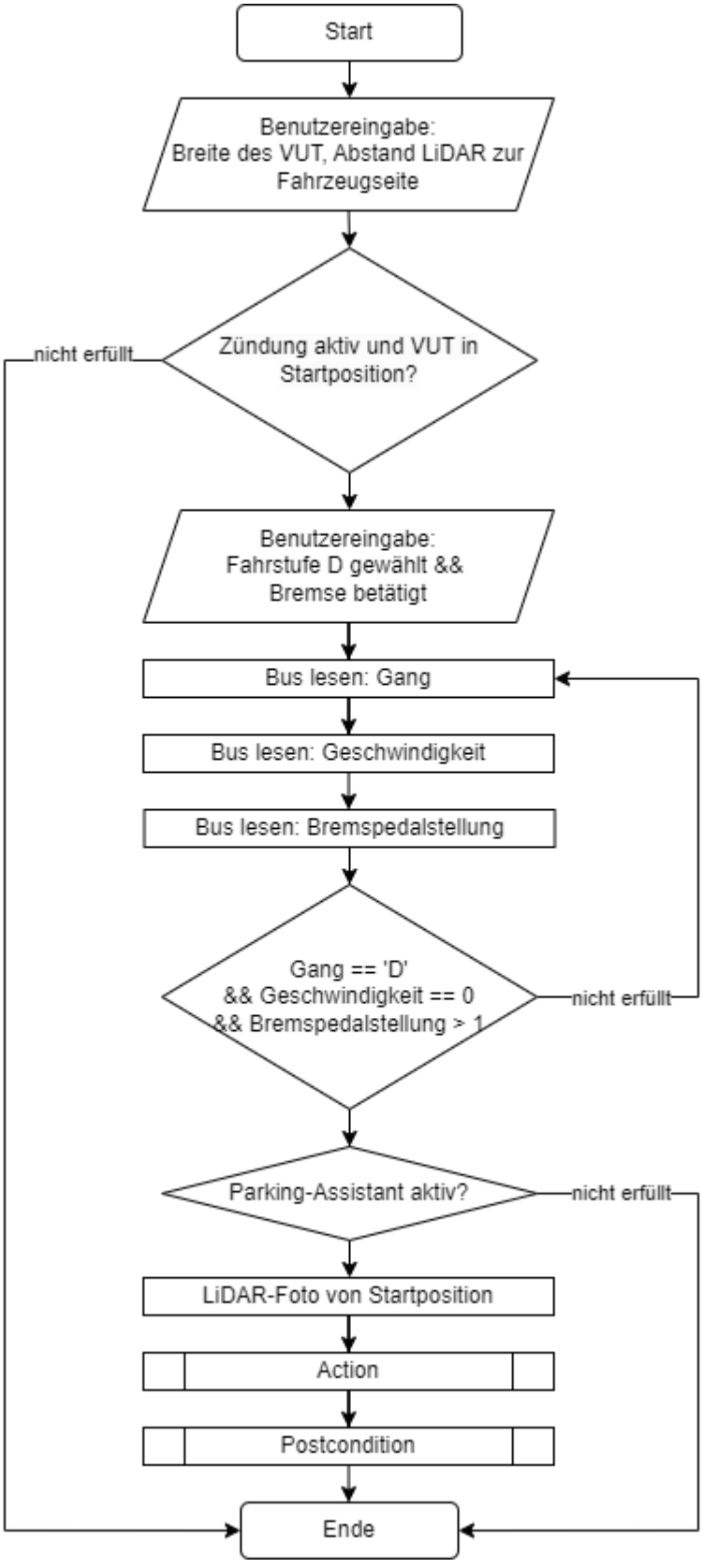
Anlage B-1 Funktion „pathfinder“ in MATLAB



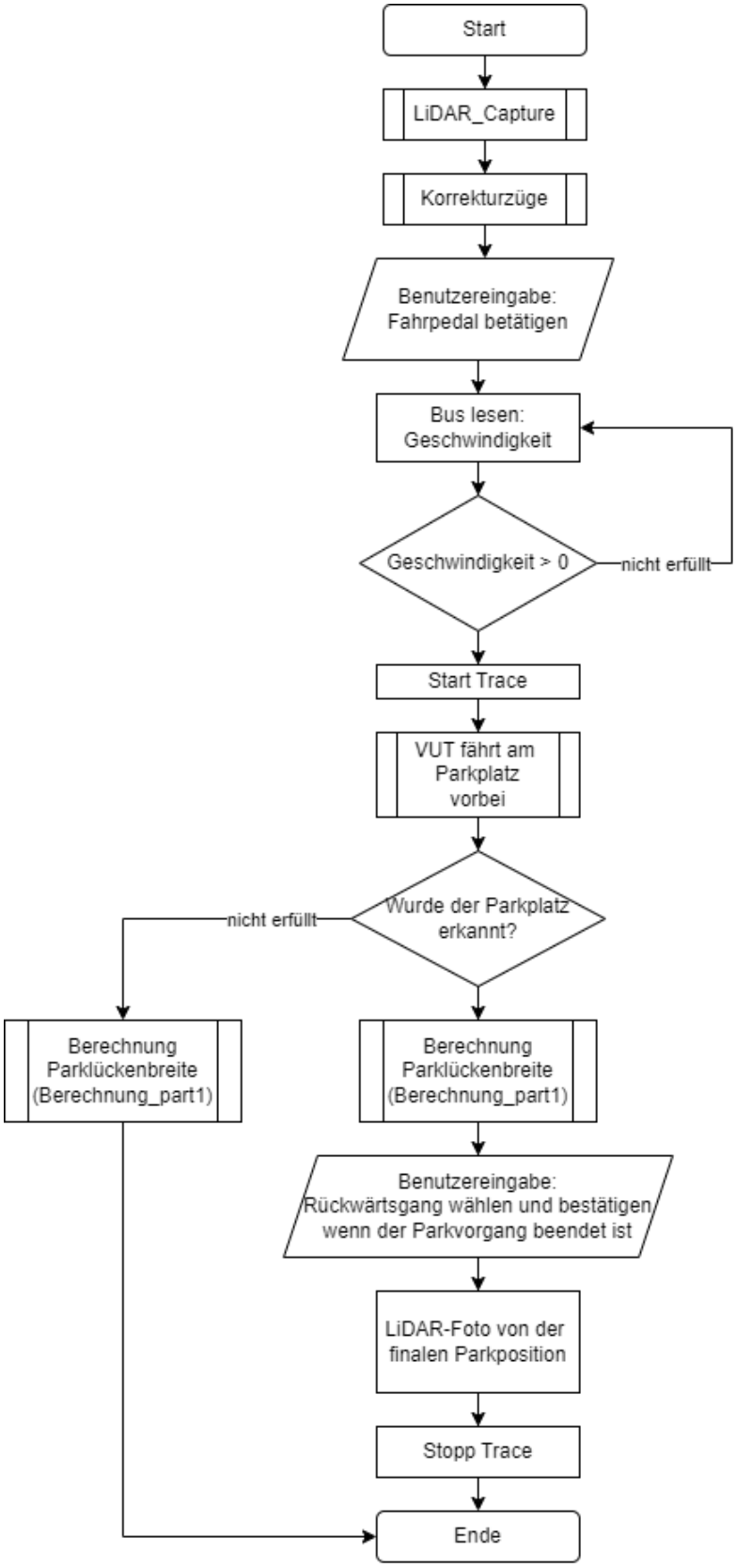
Anlage B-2 Funktion „SLAM“ in MATLAB



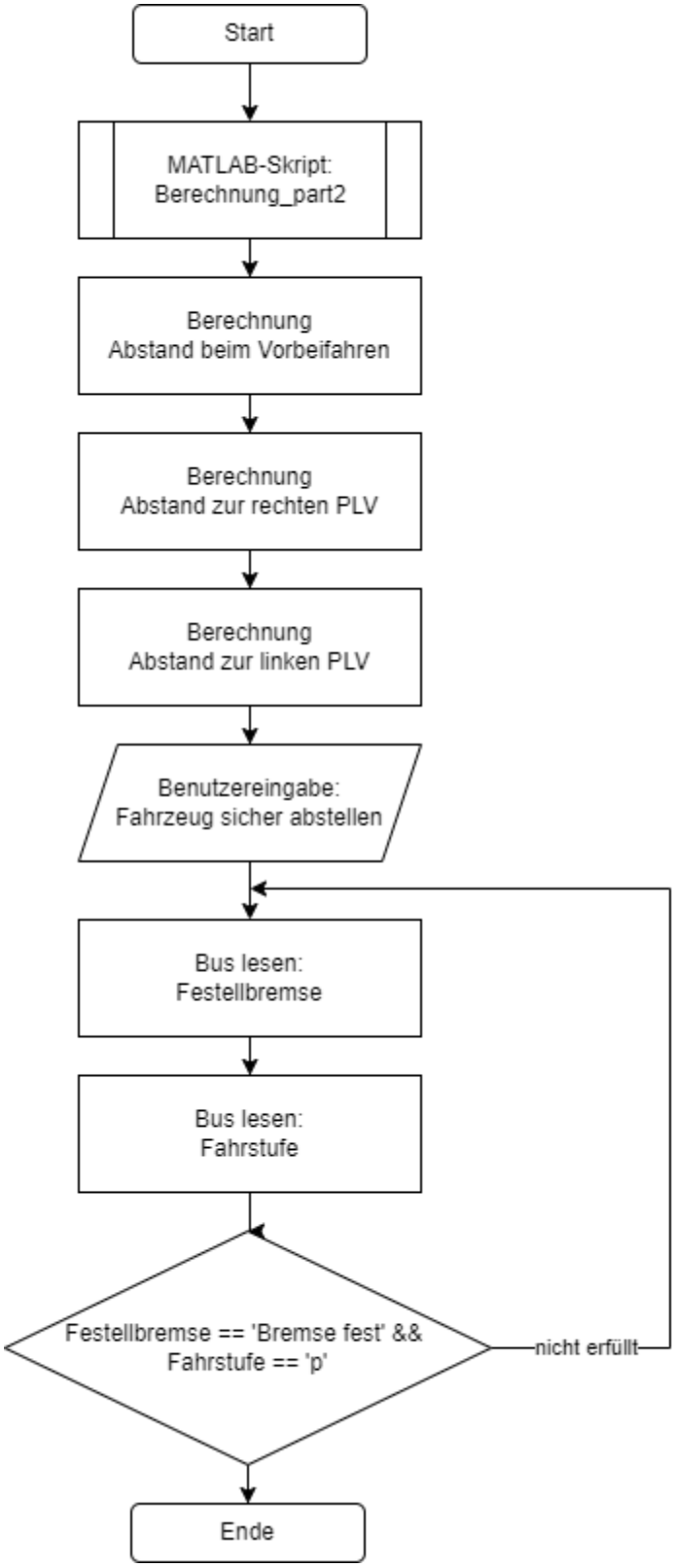
Anlage B-3 „Parking Assistant“ in ecu.test



Anlage B-4 „Action“- Unterprogramm in ecu.test



Anlage B-5 „Postcondition“ in ecu.test



C Tabellen

Verwendete Toolboxen
Automotive Toolbox
Image Processing and Computer Vision Toolbox
Robotics and Autonomous Systems Toolbox

Anlage C-2 Aufgenommene Messwerte

Measurement series 1a
Target A: 1,0 m Target B: 3,0 m

Measurement Number	LiDAR Distance (A) Target: 1,0 m	LiDAR Width Parkingspace (B) Target: 3,0 m	LiDAR Right Side Distance (R)	Measured Right Side Distance (R)	Corrections	Calculated Left Side Distance (L)	Measured Left Side Distance (L)
1	1,18	3,06	0,72	0,68	1	0,51	0,59
2	1,25	3,04	0,74	0,64	2	0,47	0,63
3	1,01	2,99	0,81	0,81	1	0,35	0,47

Measurement series 1b
Target A: 1,5 m Target B: 3,0 m

Measurement Number	LiDAR Distance (A) Target: 1,5 m	LiDAR Width Parkingspace (B) Target: 3,0 m	LiDAR Right Side Distance (R)	Measured Right Side Distance (R)	Corrections	Calculated Left Side Distance (L)	Measured Left Side Distance (L)
4	1,66	2,93	0,71	0,73	2	0,39	0,55
5	1,67	3,01	0,69	0,65	2	0,49	0,63
6	1,6	3,01	0,78	0,75	1	0,4	0,57

Measurement series 2a
Target A: 1,0 m Target B: 3,3 m

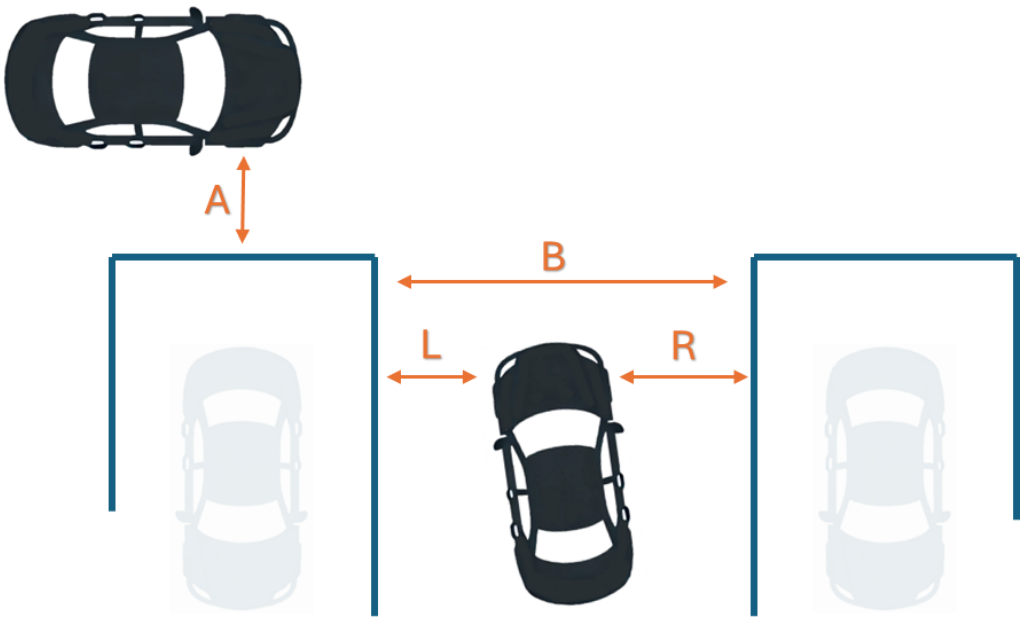
Measurement Number	LiDAR Distance (A) Target: 1,0 m	LiDAR Width Parkingspace (B) Target: 3,30 m	LiDAR Right Side Distance (R)	Measured Right Side Distance (R)	Corrections	Calculated Left Side Distance (L)	Measured Left Side Distance (L)
7	1,17	3,17	0,77	0,83	1	0,57	0,75
8	1,25	3,27	0,85	0,78	2	0,59	0,79
9	1,28	3,24	0,87	0,81	1	0,54	0,75

Measurement series 2b
Target A: 1,5 m Target B: 3,30 m

Measurement Number	LiDAR Distance (A) Target: 1,5 m	LiDAR Width Parkingspace (B) Target: 3,30 m	LiDAR Right Side Distance (R)	Measured Right Side Distance (R)	Corrections	Calculated Left Side Distance (L)	Measured Left Side Distance (L)
10	1,66	3,12	0,81	0,77	1	0,48	0,79
11	1,59	3,32	0,97	0,91	1	0,52	0,66
12	1,53	3,25	0,86	0,8	1	0,56	0,78

Auf der nachfolgenden Seite sind die (Kürzel) für die Daten erläutert

Erklärung der Kürzel



D Datenträger

Anlage D-1 Struktur des Datenträgers