Hochschule für Technik und Wirtschaft Dresden

Fakultät Maschinenbau

UNIVERSITY OF APPLIED SCIENCES

Aufgabenblatt für die Diplomarbeit

im Studiengang / Studienrichtung **Fahrzeugtechnik / Kraftfahrzeugtechnik**

Name des Diplomanden / Matr.-Nr.: **Tristan Weiland / 45470**

Thema: **Entwicklung eines Algorithmus zur Bestimmung der Fahrzeugtrajektorie und Kartierung der Umgebung mittels Lidar-Sensorik**

Aufgabe:

Durch den Einsatz von Lidar-Sensorik kann die Ermittlung der Fahrzeug-Eigenposition unter Nutzung von relevanten Infrastrukturinformationen deutlich verbessert werden. In der Diplomarbeit soll ein funktionsfähiges und erweiterbares Konzept für die Positionsbestimmung mit einer solchen Sensorik entwickelt und getestet werden.

Teilschritte:

- Einarbeitung und Literaturrecherche zur Thematik
- Konzeption und Aufbau eines flexiblen Systems zur Lidar-Messwertaufnahme
- Auswahl und Implementierung geeigneter Matlab-Erkennungsalgorithmen
- Bewertung anhand ausgewählter Testszenarien

Hochschulbetreuer: Prof. Dr. rer. nat. Toralf Trautmann
Dipl.-Ing. (FH) Dirk Engert

Ausgehändigt am: 23.01.2023 Einzureichen bis: 23.06.2023

.....
Prof. Dr. rer. nat. Toralf Trautmann
Verantwortlicher Hochschullehrer

.....
Prof. Dr.-Ing. Thomas Himmer
Prüfungsausschussvorsitzender

Hochschule für Technik und
Wirtschaft Dresden
University of Applied Sciences

Diplomarbeit

Entwicklung eines Algorithmus zur Bestimmung der Fahrzeugtrajektorie und Kartierung der Umgebung mittels LiDAR-Sensorik

vorgelegt von:	Tristan Weiland
Ort:	HTW Dresden
Fakultät:	Maschinenbau
Studiengang:	Fahrzeugtechnik
Betreuer:	Prof. Dr. rer. nat. Toralf Trautmann Dipl.-Ing. Dirk Engert
Abgabedatum:	23.06.2023

Inhaltsverzeichnis

Abkürzungsverzeichnis **III**

Verzeichnis der Formelzeichen und Symbole **V**

Abbildungsverzeichnis **VII**

Tabellenverzeichnis **X**

1 Einleitung **1**
1.1 Motivation 1
1.2 Zielstellung 2

2 Technische Grundlagen **3**
2.1 LiDAR - Light Detection and Ranging 3
2.1.1 Funktionsprinzip 3
2.1.2 Anwendungsbereiche 4
2.1.3 Sende- und Empfangszweig 5
2.1.4 Scanningmechanismen 6
2.2 Sensorik 10
2.2.1 Livox Horizon 10
2.2.2 Ouster OS1-64 12

3 Kartierung und Lokalisierung **13**
3.1 SLAM - Simultanes Lokalisieren und Kartieren 13
3.1.1 Funktionsprinzip LiDAR-SLAM 15
3.1.2 Graph-Optimierung durch Schleifenschluss-Erkennung 19
3.2 ICP - Iterativ-nächster-Punkt-Algorithmus 21
3.2.1 Funktionsprinzip 21
3.2.2 Punkt-zu-Punkt-Metrik 24
3.2.3 Punkt-zu-Fläche-Metrik 27
3.2.4 Generalisierter ICP-Algorithmus 29
3.3 NDT - Normalverteilungs-Transformation 31

4 Rahmenbedingungen und Vorbetrachtungen **34**
4.1 Matlab als Softwareumgebung 34
4.2 Erfassen der Messwerte 35
4.2.1 Einbindung LIVOX Horizon 35
4.2.2 Einbindung Ouster OS1 38
4.3 Versuchsaufbau 40

4.3.1 Versuchsträger 40
4.3.2 Versuchsumgebung 41
4.3.3 Testfälle 43

5 Implementierung der Konzepte und Algorithmen 47
5.1 Konzeptvergleich 47
5.2 Vorverarbeitung der Punktwolken 49
5.2.1 Entfernen der Bodenpunkte 49
5.2.2 Punktwolkenfilterung 53
5.3 Implementierung der Registrierungsprozesse 57
5.3.1 Frame-zu-Frame Registrierung 58
5.3.2 Graphoptimierung 62

6 Auswertung der Ergebnisse 65
6.1 Ermittlung der verwendeten Parameter 65
6.2 Methode der Versuchsauswertung 68
6.3 Vergleich und Bewertung der Algorithmen 72

7 Ausblick 77

Literatur- und Quellenverzeichnis 79

Eidesstattliche Erklärung 89

Anhang 89

Abkürzungsverzeichnis

LiDAR Light Detection and Ranging

SLAM Simultaneous Localization and Mapping

PKW Personenkraftwagen

RFA Rückfahrassistent

VP Valet Parking

GPS Global Positioning System

PLD Pulsed Laser Diode

APD Avalanche Photo Diode

PIND Positive Intrinsic Negative Diode

FoV Field of View

IMU Inertial Measurement Unit

PAP Programmablaufplan

VSLAM Visual Simultaneous Localization and Mapping

LOAM LiDAR Odometry and Mapping

KF Kalman-Filter

PF Partikelfilter

DSP digitaler Signalprozessor

IEEE Institute of Electrical and Electronics Engineers

KITTI Karlsruher Institut für Technologie & Toyota Technological Institute

LeGO Lightweight and Ground-Optimized

ICP Iterative Closest Point

PzP Punkt-zu-Punkt

PzF Punkt-zu-Fläche

FzF Fläche-zu-Fläche

SKD Scan-Kontext-Deskriptor

RMSE Root Mean Square Error

SWZ Singulärwertzerlegung

MKQ Methode der kleinsten Quadrate

GICP Generalized Iterative Closest Point

NDT Normal-Distributions-Transform

MdkG Modell der konstanten Geschwindigkeit

OSM OpenStreetMap

JOSM Java OpenStreetMap Editor

MSAC M-estimator Sample Consensus

DGNSS Differential Global Navigation Satellite System

HKA Hauptkomponentenanalyse

Verzeichnis der Formelzeichen und Symbole

Symbol	Einheit	Beschreibung
d	m, mm	Abstand zwischen Sensor und Objekt
c	m/s	Wellenausbreitungsgeschwindigkeit
t_{oF}	s, ms	Zeit vom Aussenden bis Empfangen des Lichtimpulses
R	-	Rotationsmatrix für drei- bzw. zweidimensionale Rotationen
t	-	Translationsvektor für drei- bzw. zweidimensionale Verschiebungen
Q, P	-	Fixierte und bewegte Punktwolke
q, p	-	Raumpunkte der Punktwolken Q, P
C	-	Menge der korrespondierenden Punkte aus Q, P
T	-	Transformation bestehend aus R und t
t_x, t_y, t_z	m, mm	Translationskomponenten von t für jede Raumachse
$R_x(\alpha)$	-	Rotationsmatrix um die x-Achse, abhängig vom Winkel α
$R_y(\beta)$	-	Rotationsmatrix um die y-Achse, abhängig vom Winkel β
$R_z(\gamma)$	-	Rotationsmatrix um die z-Achse, abhängig vom Winkel γ
e^2	m, mm	Summe der quadratierten Abstände, der Punkte aus Q und P
q_n, p_n	-	assoziiertes Punktepaar
ϕ	°, rad	Winkel einer zweidimensionalen Rotationsmatrix
q_0, p_0	-	gemittelte Punkte der Punktwolken Q und P
K	-	Kreuzkovarianzmatrix der Punktwolken Q und P
$\bar{p}_n$	-	durch R und t transformierter Punkt p
n_n	-	Normalvektor des Punktes q_n
$x(\phi, t_x, t_y)$	-	Vektor der Transformationsparameter der PzF-Metrik
$E(x)$	m, mm	Summe der quadrierten Abstände der PzF-Metrik
H	-	Hesse-Matrix
J	-	Jacobi-Matrix
Δx	-	Veränderung des Parametervektors x
$\hat{Q}, \hat{P}$	-	theoretische Punktwolken des GICP-Modells

C_n^Q, C_n^P	-	Kovarianzmatrizen der der theoretischen Normalverteilung von $\hat{Q}$ und $\hat{P}$
σ^2	-	Varianz
$\mathcal{N}(\hat{q}_n, C_n^Q)$	-	theoretische Normalverteilungen um den Erwartungswert $\hat{q}_n$ mit der entsprechenden Kovarianzmatrix
$\mathcal{N}(\hat{p}_n, C_n^P)$	-	theoretische Normalverteilungen um den Erwartungswert $\hat{p}_n$ mit der entsprechenden Kovarianzmatrix
d_n	m, mm	Abstand zwischen zwei Punkten des GICP-Algorithmus
T^*	-	optimale Transformation zweier Punkte des GICP-Algorithmus
ϵ	-	Kovarianz entlang der Flächennormale eines Punktes im GICP-Algorithmus
ν_n, μ_n	-	Normalvektoren zweier Punkte q_n, p_n im GICP-Algorithmus
R_{μ_n}, R_{ν_n}	-	Rotationsmatrizen für Rotationen entlang der Normalvektoren ν_n, μ_n
$\bar{q}_i$	-	gemittelter Punkt einer Zelle des NDT-Algorithmus
x_i	-	beliebiger Punkt in einer Zelle des NDT-Algorithmus
$p(x_i)$	-	Wahrscheinlichtkeit für das Existieren eines Punkte an der Stelle x_i in einer Zelle des NDT-Algorithmus
$p(\phi, t_x, t_y)$	-	Vektor der Transformationsparameter des NDT-Algorithmus
x_i'	-	durch $p(\phi, t_x, t_y)$ transformierter Punkt x_i des NDT-Algorithmus
$\texttt{score}(p)$	-	Summe der ausgewerteten Normalverteilungen des NDT-Algorithmus
g_1, g_2	-	dreidimensionale Geraden
u, v	-	Richtungsvektoren der Geraden g_1, g_2
E, F	-	Ebenen im dreidimensionalen Raum
n_E, n_F	-	Normalvektoren der Ebenen E, F

Abbildungsverzeichnis

Abbildung 2.1 Pulsantwort eines festen (links) und weichen Objekts (rechts) [3, S. 319-320] 4
Abbildung 2.2 Schematische Darstellung der LiDAR-Arten [2] 5
Abbildung 2.3 LiDAR-Sensor mit 40 Ebenen, PLD-Anordnung (links) und APD-Anordnung (rechts) [5] 6
Abbildung 2.4 Schematische Darstellung eines rotierenden LiDAR-Sensors mit 16 Ebenen [5] 7
Abbildung 2.5 Schematische Darstellung eines LiDAR-Sensors mit Risley-Prismen [5] 8
Abbildung 2.6 Vergleich der Scanning-Muster: (a) Livox-LiDAR und (b) rotierender LiDAR mit festen Ebenen [9] 9
Abbildung 2.7 Anordnung der Prismen des LIVOX Horizon (links) und erzeugtes Scanning-Muster (rechts) [5] 9
Abbildung 2.8 Vergleich der FoV-Abdeckung des LIVOX Horizon mit traditionellen LiDAR-Sensoren [10] 11

Abbildung 3.1 Allgemeingültiger PAP eines LiDAR-SLAM Algorithmus [27] 16
Abbildung 3.2 Vergleich zwischen nicht-berichtigter Punktwolke (links) und berichtigter Punktwolke (rechts) eines rotierenden LiDAR-Sensors [28] 18
Abbildung 3.3 Vergleich zwischen realer Positionen (blau) und geschätzten Positionen (grau) eines sich bewegenden LiDAR-Roboters 19
Abbildung 3.4 Graph vor Schleifen-Korrektur (oben) und Graph nach Korrektur (unten) 20
Abbildung 3.5 PAP eines allgemeinen ICP-Algorihmus 22
Abbildung 3.6 Direkte Lösung des Transformationsproblems durch eindeutige und korrekte Assoziation der Punkte aus beiden Kurven [39] 25
Abbildung 3.7 Mögliches Ergebnis nach einer Iteration des ICP-Algorithmus mit geschätzten Punktepaaren [39] 25
Abbildung 3.8 Visueller Vergleich von: (a) uniformes bzw. voxelisierendes Sampling, (b) Filtern nach geringen Normalschwankungen und (c) Filtern nach hohen Normalschwankungen [37] . . 27
Abbildung 3.9 Vergleich der Fehlervektoren zwischen PzP-Metrik (links) und PzF-Metrik (rechts) [37] 28
Abbildung 3.10 Vergleich eines Punktepaars zwischen PzP-Metrik (links) und PzF-Metrik (rechts) [37] 28

Abbildung 3.11 Grafische Darstellung von Punktepaarungen (blaue Linien) und Kovarianzstrukturen der einzelnen Punkte (schwarze Linien) [41] 30
Abbildung 3.12 Schematische Darstellung der Netzverteilung: Die weiße Zelle stellt das ursprüngliche Netz dar, die farblich markierten Zellen visualisieren die mit Versatz eingefügten Netze 32

Abbildung 4.1 Bildschirmausschnitt der Benutzeroberfläche des „Livox Viewer“ 36
Abbildung 4.2 Visualisierung zwei aufeinanderfolgender Pakete des Livox Horizon 37
Abbildung 4.3 Anbringung der Sensorik am Testfahrzeug 40
Abbildung 4.4 Aufbau und Vernetzung der Messtechnik im Fahrzeug . . 41
Abbildung 4.5 Das K-Gebäude der HTW Dresden und die anliegende Testfläche [61] 42
Abbildung 4.6 GPS-Punkte der Testfläche visualisiert in JOSM 42
Abbildung 4.7 Karte der Testfläche als Matlab Figure 43
Abbildung 4.8 Skizzierung der Testfälle auf der Testfläche 45

Abbildung 5.1 Frame des Livox Horizon und Frame des Ouster OS1 mit Bodenpunkten 50
Abbildung 5.2 Frame des Livox Horizon und Frame des Ouster OS1 nach Entfernung der Bodenpunkte 51
Abbildung 5.3 Vergleich der Transformationsgenauigkeit zwischen Punktwolken des Livox Horizon mit Bodenpunkten und Punktwolken ohne Bodenpunkte 52
Abbildung 5.4 Ungefilterter Frame des Ouster OS1 und gefilterter Frame 54
Abbildung 5.5 Vergleich der Transformationen zwischen zwei aufeinanderfolgenden ungefilterten und gefilterteten Punktwolken des Ouster OS1 56
Abbildung 5.6 Matlab Code zur Vorabdefinierung des benötigten Speicherplatzes 59
Abbildung 5.7 Übersicht der Struktur eines `pcviewset`-Objekts 60
Abbildung 5.8 Vereinfachter Programmablaufplan des Kartierungsalgorithmus ohne Schleifen-Optimierung 61
Abbildung 5.9 Eintrag einer erkannten Schleifenverbindung in der Tabelle `Connections` 63
Abbildung 5.10 Vereinfachter Programmablaufplan des Kartierungsalgorithmus mit Schleifen-Optimierung 64

Abbildung 6.1 Winkelbeziehungen zwischen Gerade-Gerade, Gerade-Ebene und Ebene-Ebene [85] 70

Abbildung 6.2 Vergleich zwischen unkorrigierten (oben) und korrigierten Punktwolken (unten) des zweiten Testfalls 71
Abbildung 6.3 Vergleich zwischen unkorrigierter und korrigierter LiDAR-Karte des dritten Testfalls (GICP) 76

Tabellenverzeichnis

Tabelle 4.1 Übersicht der Eigenschaften einer Punktwolke des Livox Horizon 38
Tabelle 4.2 Übersicht der Eigenschaften einer Punktwolke des Ouster OS1 39

Tabelle 5.1 Übersicht der Unterschiede und Gemeinsamkeiten der betrachteten Algorithmen 48
Tabelle 5.2 Übersicht der verfügbaren Sampling-Methoden in Matlab . . 53
Tabelle 5.3 Vergleich der Transformationswerte zwischen ungefilterten und gefilterten Frames 55
Tabelle 5.4 Übersicht der entwickelten Matlab Skripts 57
Tabelle 5.5 Übersicht der restlichen Parameter und deren Funktion 58
Tabelle 5.6 Übersicht der zusätzlichen Parameter, Objekte und Unterfunktionen für Skripte mit Schleifen-Optimierung 62

Tabelle 6.1 Übersicht der gewählten Parameterwerte 65
Tabelle 6.2 Ermittelte Parameterwerte für `gridSize` 67
Tabelle 6.3 Translationsfehler der Algorithmen und Sensoren 73
Tabelle 6.4 Gierwinkelfehler der Algorithmen und Sensoren 74

1. Einleitung

Assistenzfunktionen von Fahrzeugen im Straßenverkehr werden zunehmend komplexer und nehmen der fahrzeugführenden Person immer häufiger Aufgaben ab. Abläufe die früher manuell gesteuert werden mussten, werden automatisiert und die Verantwortung wird infolgedessen dem Fahrzeug übergeben. Während viele Assistenzfunktionen unterstützend arbeiten, gibt es vergleichsweise noch wenige, die gesamte Handlungsabläufe übernehmen. Im Bereich der Fahrzeugtechnik haben sich Funktionen wie die Einparkautomatik durchgesetzt. Ähnliche Funktionen, die im Vergleich dazu größere Handlungsstränge absolvieren, sind zum Beispiel Rückfahrassistenten (RFAs) oder das Valet Parking (VP). Es wird das Ziel verfolgt, der fahrzeugführenden Person Zeit zu sparen, indem beispielsweise das Fahrzeug am Parkhaus abgegeben wird und eigenständig nach einem Parkplatz sucht. RFAs hingegen erfüllen ihren Zweck, indem sie kompliziertere Fahrvorgänge, bei denen häufiger Auffahrunfälle entstehen, übernehmen. Ein Beispiel ist das Manövrieren aus einer schmalen Sackgasse.

Damit genannte Assistenzsysteme funktionieren können, benötigen sie Informationen über die Umgebung, sowie die Eigenposition und Orientierung innerhalb dieser. Jenes Problem wird häufig mit Simultaneous Localization and Mapping (SLAM) bzw. simultaner Lokalisierung und Kartierung, betitelt. Für Anwendungen dieser Art ist eine breit gestreute Menge an Informationen über die dreidimensionale Umgebung und der relativen Position notwendig. Mithilfe dieser Daten können Systeme Routen berechnen, um Hindernisse zu umfahren oder eigenständig zu einem vorher festgelegten Ziel navigieren.

1.1. Motivation

Sensorik wie Ultraschall, Radar, Global Positioning System (GPS) und Kameras sind fester Bestandteil der Grundausstattung vieler Personenkraftwagen (PKW). Mit den genannten Komponenten können bereits digitale Karten der Umwelt erfasst und modelliert werden. GPS-Daten werden für die Routenplanung und Navigation verwendet. Kameras können zur Erkennung und Identifizierung von Objekten sowie Abständen genutzt werden. Des Weiteren liefern Ultraschall- und Radarsensoren Distanzmessungen zu anderen Verkehrsteilnehmer*innen oder Hindernissen. Um jedoch eine dreidimensionale Darstellung der Umwelt erzeugen zu können, müssen diese Informationen häufig fusioniert werden. Da eine Abstandsmessung per Kamera ungenauer sein kann, als per Radar oder Ultraschall, werden häufig die Daten mehrerer verschiedener Sensoren fusioniert. Auch hier gibt es Ausnahmen, denn Tesla setzt bei seinem Autopiloten auf die Verwendung von Stereokamerasystemen, welche von neuronalen Netzwerken ausgewertet werden. Diese sollen das menschliche Sehen modellieren, um so Oberflächen, Formen und Farben und die zugehörigen Abstände zu erfassen [1]. Systeme, die

auf eine geringe Varietät der Sensorik setzen, könnten jedoch Defizite in bestimmten Situationen aufweisen. So ist das Sichtfeld von Kameras bei Dunkelheit stark beeinträchtigt, da ihre Funktionalität von externen Lichtquellen, wie dem Sonnenlicht, abhängig ist.

Ein neuer Ansatz für die Erfassung der Umgebung könnte demnach die Sensortechnologie: Light Detection and Ranging (LiDAR) sein. Durch Aussenden von Laserimpulsen und Empfangen dieser, können simultan Abstände, relative Position und Reflexionseigenschaften von Objekten erfasst werden. Das Messen mit Lasern bietet zudem den Vorteil, dass Objekte, wie Autos oder Fahrräder, erkannt und voneinander unterschieden werden können. Dies kann mit Radar- oder Ultraschallsensoren nicht allein bewerkstelligt werden [2]. In der Industrie und Robotik existieren bereits laserbasierte Messsysteme. So könnte auch im Bereich der Fahrzeugtechnik eine Umgebungserkennung durch laserbasierte Messdaten bewerkstelligt werden.

1.2. Zielstellung

Diese Arbeit soll einen Einblick in die Modellierung der Umwelt mit Lasermesssystemen für mobile Anwendungen geben. Dabei sind verschiedene Sensoren und deren Tauglichkeit in definierten Testfällen zu untersuchen und auszuwerten. Ziel ist es ein System für Messwertaufnahme und Auswertung in Matlab zu entwickeln. Für den Abgleich nacheinander aufgenommener Messdaten, um eine Nachbildung der abgefahrenen Umgebung zu erzeugen, sind verschiedene Algorithmen in Matlab zu implementieren. Um diese vergleichen und bewerten zu können sind eine Auswahl an Testszenarien zu treffen.

2. Technische Grundlagen

Zu Beginn dieser Arbeit sollen vorerst die technischen Besonderheiten von laserbasierten Messsystemen erläutert werden. Dazu gehören zum einen das Funktionsprinzip von LiDAR, sowie einige technische Umsetzungen und deren Anwendungsgebiete. Des Weiteren werden ausgewählte Sensoren und deren technische Merkmale genauer erklärt.

2.1. LiDAR - Light Detection and Ranging

LiDAR beschreibt das Erkennen und Messen mithilfe von Licht. Folglich gehört LiDAR zur Obergruppe der optischen Messverfahren. Die meistgenutzte Messmethodik im Bereich der Fahrzeugtechnik ist die sog. „Time of Flight“-Messung. Es gibt jedoch viele Unterschiede in der technischen Umsetzung der Sensoren [3].

2.1.1. Funktionsprinzip

LiDAR ähnelt dem Radarverfahren, bei welchem Mikrowellen emittiert werden. Bei LiDAR kommen jedoch Lichtwellen des Infrarot-, Ultraviolett- oder des sichtbaren Spektrums zum Einsatz. Es werden Lichtimpulse vom Sensor ausgesendet, welche von eventuell vorhandenen Objekten reflektiert werden und so zum Sensor zurückgelangen. Die im Voraus erwähnte „Time of Flight“-Messung nutzt die verstrichene Zeit zwischen Emittieren und Empfangen des Lichtimpulses, um so auf die zurückgelegte Strecke schließen zu können. Die benötigte Zeit, um zum Objekt und wieder zurück zum Sender zu gelangen, steht dementsprechend für die doppelte Strecke zwischen Sender und Objekt. Formel (2.1) beschreibt diesen Zusammenhang.[4], [3].

$$d = \frac{c \cdot t_{oF}}{2} \tag{2.1}$$

Dabei stellt d die berechnete Distanz zum Objekt dar. c ist die Ausbreitungsgeschwindigkeit der Wellen im jeweiligen Medium und t_{oF} die gemessene Zeit zwischen Emittieren und Empfangen des Lichtimpulses. Bedingungen für das Gelingen einer Messung sind, dass das bestrahlte Objekt reflektierende Eigenschaften aufweist und dass im Weg des Lichts keine Störungen auftreten. Befinden sich Regen, Nebel oder andere lichtstreuenden Medien in der Atmosphäre, kann es zur Absorption oder Streuung des Lichts kommen. Solche Beeinträchtigungen werden nach Winner als „weiche“ atmosphärische Störungen klassifiziert [3]. Im Falle dessen, dass es zur Streuung kommt, können jedoch noch Teile des Lichts reemittiert werden. Das Ergebnis wird dabei allerdings verfälscht. Pulsantworten die durch die Reflexion von Nebel oder ähnlichen Medien entstehen, weisen ein längeres, flacheres Empfangsecho auf. Feste Objekte hingegen erzeugen eine Pulsantwort, die der Form einer Gaußkurve entspricht. Die

Verflachung der Pulsantwort entsteht dadurch, dass „weiche“ Objekte Echos aus verschiedenen Tiefen erzeugen, was bedeutet, dass das Licht nicht zeitgleich im Ganzen reflektiert wird, sondern aufgeteilt über einen längeren Zeitraum hinweg. Eine grafische Darstellung dieses Verhaltens wird in der folgenden Abbildung (2.1) veranschaulicht. [3]

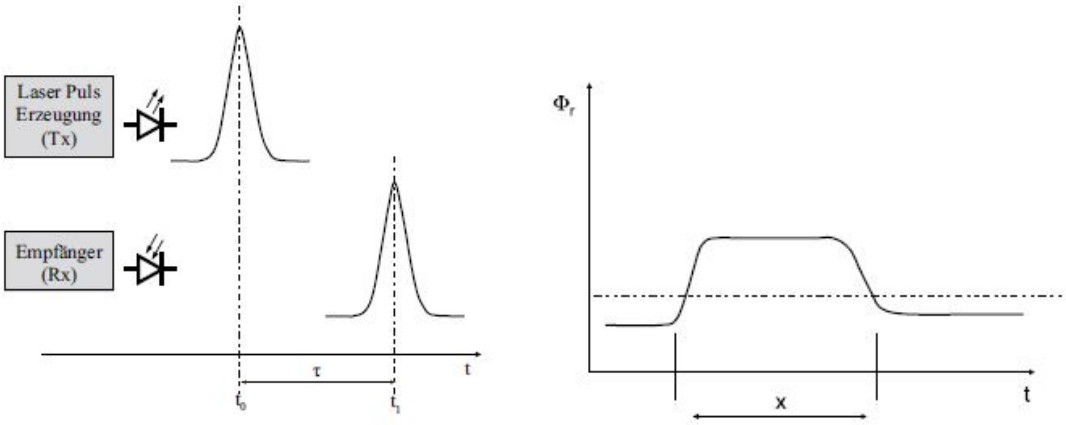


Abbildung 2.1.: Pulsantwort eines festen (links) und weichen Objekts (rechts) [3, S. 319-320]

Daraus lässt sich schließen, dass starker Regen und Nebel das Messen mit LiDAR erheblich beeinträchtigen oder gar unmöglich machen.[3].

Durch die Verwendung von Licht können LiDAR-Sensoren zusätzlich zur Abstandsmessung auch visuelle Eigenschaften erfassen. Da die Intensität des empfangenen Lichts messbar ist, können so die Reflexionseigenschaften des bestrahlten Objekts bestimmt werden. Aufgrund dessen können bestimmte Gegenstände, die stark reflektieren von weniger reflektierenden unterschieden werden. Verkehrszeichen weisen einen hohen Grad an Reflexion auf und können demnach, unter Verwendung geeigneter Detektionsalgorithmen, herausgefiltert werden.

2.1.2. Anwendungsbereiche

Das Messen mithilfe von Lasern ist bereits in vielen Wissenschaften Stand der Technik. Dazu gehören unter anderem die Geographie, Atmosphärenphysik und Vermessung. Heutzutage findet LiDAR vermehrt in moderneren Gebieten Einsatz, wie zum Beispiel in der Robotik und im Bereich des automatisierten Fahrens. Die technologischen Umsetzungen von LiDAR-Sensoren lassen sich entsprechend der Einsatzbedingungen kategorisieren, da der Zweck entscheidend für die Funktionsweise ist, siehe Abbildung 2.2. Für Anwendungen, die vom Boden losgelöst sind, wie die Verwendung aus der Luft oder Unterwasser, werden in der Regel sehr leistungsstarke Emitter benötigt, um die langen Strecken zum gemessenen Objekt erreichen zu können. [2]

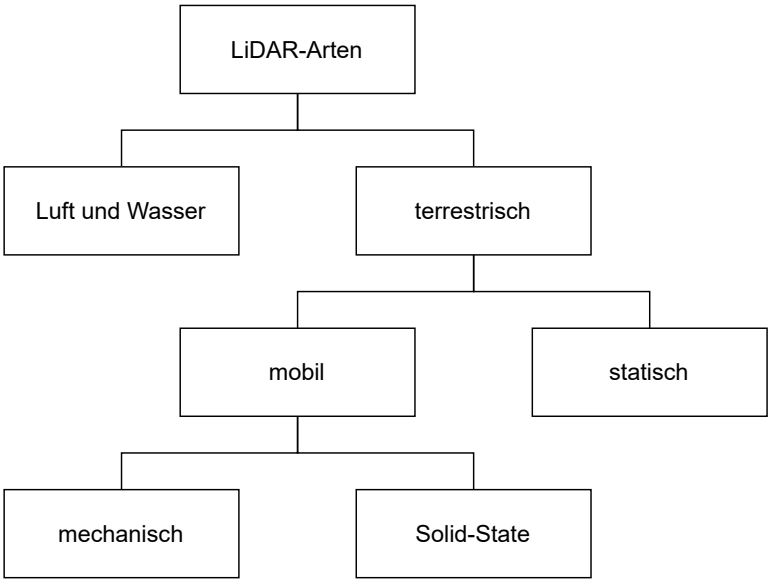


Abbildung 2.2.: Schematische Dartsellung der LiDAR-Arten [2]

Terrestrisch steht, wie der Name verrät, für alle Verwendungen die vom Boden aus geführt werden. LiDAR-Scanner weisen hier eine hohe Präzision und Punktwolkendichte auf, um so Objekte klar identifizieren zu können. Bei statischen Einsätzen kann die Messeinrichtung während eines Messvorgangs nicht bewegt werden. Aufnahmen aus mehreren Winkeln müssen getrennt voneinander aufgezeichnet werden, so wie es in der konventionellen Vermessung der Fall ist. [2]

Mobile Anwendungen stellen den für diese Arbeit wichtigen Teil von LiDAR-Sensoren dar. Sensoren dieser Klasse liefern bis zu Millionen Punkte pro Minute und haben üblicherweise eine hohe Datendichte. Sie stellen so den schnellsten Weg zur Koordinatenerfassung dar. Einsatzgebiete sind hier die moderne Kartierung sowie der Bereich des automatisierten Fahrens. [2]

2.1.3. Sende- und Empfangszweig

Auch die Kategorie der mobilen LiDAR-Sensoren kann in verschiedene Bauweisen unterteilt werden. Zum einen in mechanische und zum anderen in Solid-State-Sensoren. Letztere sind vergleichsweise neu und selten im Einsatz. Das Prinzip beruht auf der Verwendung von Halbleiter-Technologie und benötigt daher keine beweglichen Bauteile. Das deutsche Start-up-Unternehmen Blickfeld aus München entwickelt und produziert solche Sensoren für den Einsatz in der Automobiltechnik [2]. Da die für diese Arbeit verwendeten Sensoren jedoch mechanisch arbeiten, werden Solid-State-Sensoren nicht weiter betrachtet.

Die zwei Hauptkomponenten eines Laser-Messsystems sind der Sende- und Empfangszweig. Der Sendezweig besteht aus der Laserquelle, welche den Lichtstrahl aussendet.

Dafür werden oftmals Pulsed Laser Diodes (PLDs) verwendet, bzw. gepulste Laserdioden. Dabei liegen die Wellenlängen des ausgesendeten Lichts üblicherweise zwischen 850 nm bis 1 µm. Es wird angestrebt den Messimpuls so kurz wie möglich zu halten, unter der Beachtung, dass Spitzenleistungen der Laser bei 75 W und höher liegen können. Umso länger der Messimpuls, desto höher ist die anfallende Energie. Durch die Nutzung in urbanen Gebieten muss sichergestellt werden, dass durch die Laser keine Verletzungen hervorgerufen werden können.

Der Empfangszweig besteht aus Dioden, welche den reflektierten Impuls aufnehmen und so ein elektrisches Signal erzeugen können. Die Empfindlichkeit ist maßgebend für die Performance von LiDAR-Sensoren. Damit Messgenauigkeiten im cm-Bereich erzielt werden können, ist eine hohe Messgeschwindigkeit erforderlich. Bei einem typischen Messbereich von 0,1 m bis 150 m liegt die Lichtlaufzeit bei 0,1 ns bis 1,0 µs. Typische Dioden, die hier zum Einsatz kommen, sind Positive Intrinsic Negative Diodes (PINDs) und Avalanche Photo Diodes (APDs). [3]

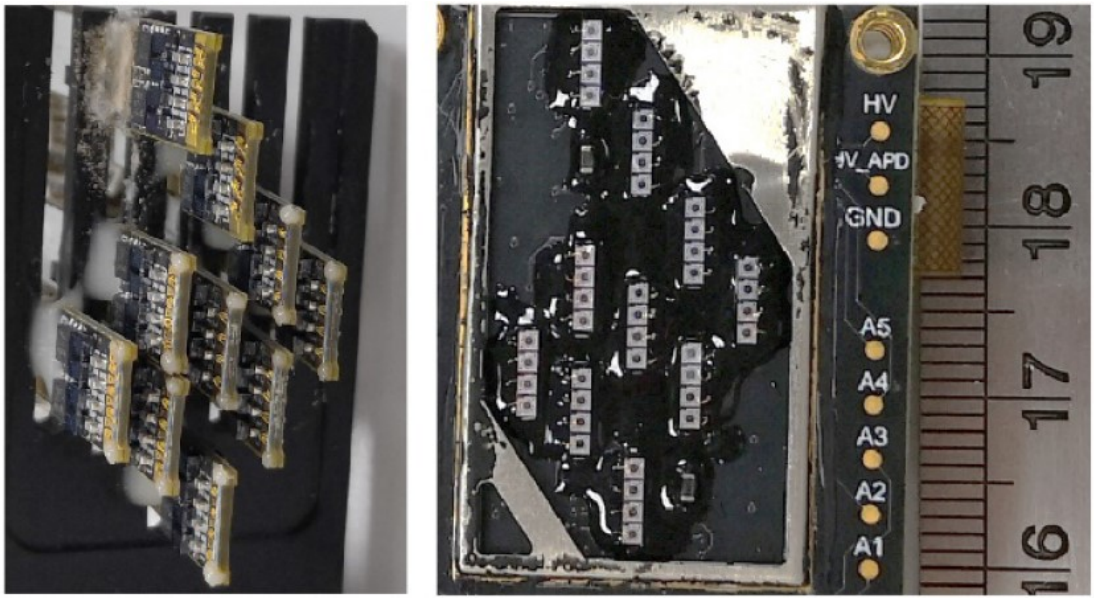


Abbildung 2.3.: LiDAR-Sensor mit 40 Ebenen, PLD-Anordnung (links) und APD-Anordnung (rechts) [5]

In Abbildung 2.3 ist eine typische Anordnung der Sende- und Empfangselemente in Ebenen zu sehen. Alle Komponenten müssen mit höchster Präzision verklebt werden, um eine hohe Messgenauigkeit zu garantieren [5].

2.1.4. Scanningmechanismen

Um eine sogenannte Punktwolke, welche eine Menge dreidimensionaler Koordinaten beschreibt, erzeugen zu können, muss der durch die Dioden erzeugte Laserstrahl durch den Raum bewegt werden. Anderenfalls wäre die Menge an Punkten auf die

Anzahl der verbauten Laserdioden begrenzt. Damit ein möglichst großer Bereich des dreidimensionalen Raumes abgetastet werden kann, wurden bereits einige verschiedene Methoden der Verteilung entwickelt, welche allgemein als Scanningmechanismen bezeichnet werden [4]. Im Folgenden wird auf zwei technische Umsetzungen eingegangen, die in dieser Arbeit Anwendung finden.

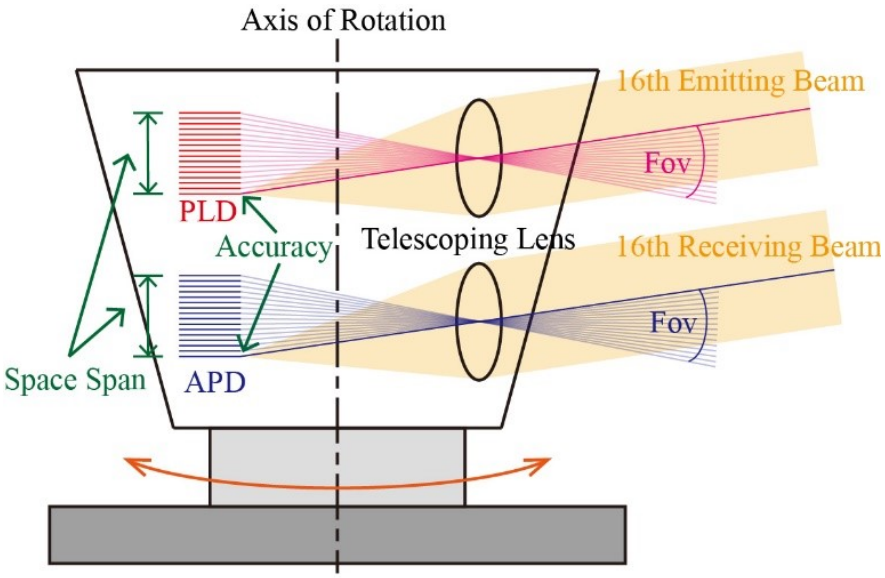


Abbildung 2.4.: Schematische Darstellung eines rotierenden LiDAR-Sensors mit 16 Ebenen [5]

Die konventionelle Methode, oft im Bereich der automatisierten Mobilität eingesetzt, basiert auf um die Höhenachse mechanisch rotierende Sensoren. Die Laserquellen und -empfänger werden dabei übereinander in Ebenen angeordnet, typischerweise ab 16 Ebenen aufwärts, bis hin zu über 100 Ebenen. Der Vorteil, der sich dadurch ergibt, ist eine räumliche Abdeckung von 360°. Die Laserquellen werden dabei in einem Objektiv gebündelt und in den meisten Fällen im gleichen Winkel zueinander durch den Raum verteilt. So entsteht ein vertikales Field of View (FoV) dessen Größe von der Ebenenanzahl und den Winkelabständen zwischen den Ebenen abhängig ist. Die Auflösung pro Grad wird ebenfalls durch die Winkelabstände definiert. Umso mehr Ebenen, bei gleichem vertikalen FoV, ein Sensor besitzt, desto höher ist die resultierende Auflösung [5]. Durch ihre Bauweise sind solche Sensoren häufig sehr teuer. Da viele PLDs und APDs mit hoher Präzision verbaut werden müssen, erschwert es den Automatisierungsprozess in der Produktion [5], [6]. Trotz allem bieten sie den Vorteil, gerade im Bereich der Fahrzeugtechnik, dass mit einem einzigen Sensor alle Richtungen abgedeckt werden können und so keine toten Winkel entstehen. Ein schematischer Aufbau eines solchen Sensors wird in Abbildung 2.4 dargestellt.

Es gibt jedoch auch Sensoren die auf ein 360° Sichtfeld verzichten und dabei hohe Auflösungen bei geringerem technischem Aufwand erzeugen können. Die Raumverteilung

der Laserquellen wird dann hauptsächlich durch den Einsatz von Spiegeln und/oder Prismen erzeugt [7].

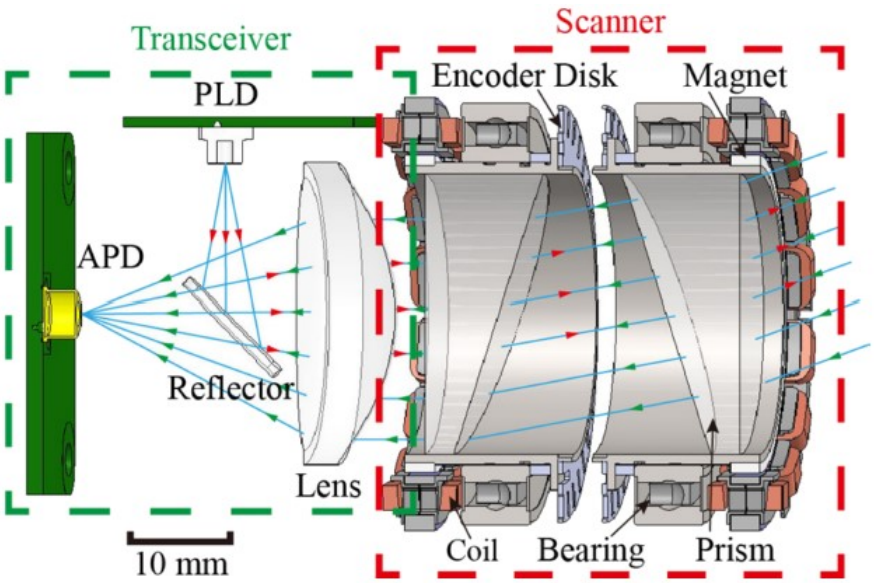


Abbildung 2.5.: Schematische Darstellung eines LiDAR-Sensors mit Risley-Prismen [5]

So hat die Firma Livox Technology Company Limited einen eigenen Ansatz entwickelt, um mithilfe von weit verfügbaren optischen Komponenten und vergleichsweise geringer Anzahl an Laser- und Empfangsquellen, eine große und zugleich dichte Punktverteilung zu erzielen [8]. Bei diesem Verfahren werden gepaarte rotierende Prismen eingesetzt, um die Laserstrahlen abzulenken [5].

Der Aufbau eines solchen LiDAR-Sensors kann dabei in zwei separate Module unterteilt werden, der sog. Transceiver und der Scanner. Im Transceiver werden die PLDs und APDs verbaut. Die Lichtstrahlen der PLDs werden im Transceiver von einem geneigten Spiegel ausgerichtet und durch eine asphärische Linse in Richtung des Scanners kollimiert. Nach der Rückstrahlung durch im Raum befindliche Objekte, treffen die Strahlen wieder auf die gleiche Linse und werden so auf die Detektoren fokussiert. Das Scanner-Modul besteht aus zwei rotierenden Prismen, welche den Lichtstrahl ablenken. Solche Prismen Paare werden als Risley-Prismen bezeichnet. Abhängig davon, wie groß der Unterschied zwischen den Rotationsgeschwindigkeiten der beiden Prismen ist, werden die Strahlen entweder in Spiralen- oder Rosettenform abgelenkt. Dadurch entsteht ein kreisrundes FoV. Die Unterschiede des Scanning-Musters, im Vergleich zu einem rotierenden Sensor mit festen Ebenen, ist in Abbildung 2.6 dargestellt. Es entstehen keine „Leerzeilen“, da durch die Rotation der Prismen, über die Integrationszeit hinweg, das gesamte FoV abgedeckt wird. [5]

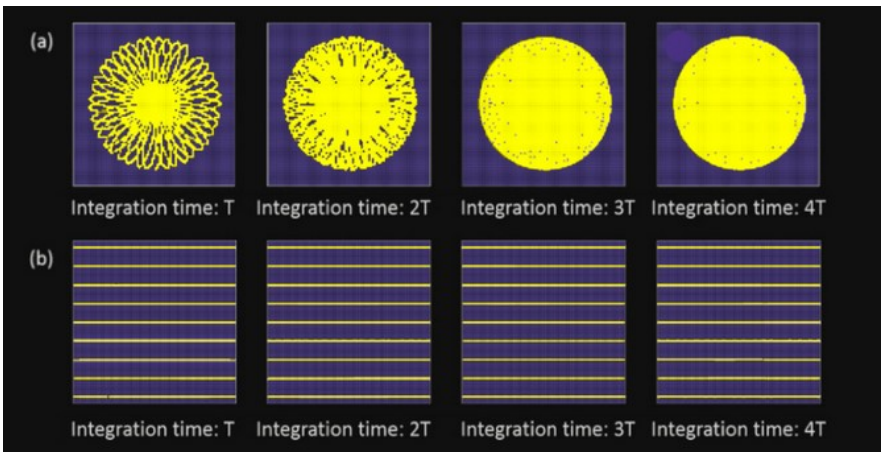


Abbildung 2.6.: Vergleich der Scanning-Muster: (a) Livox-LiDAR und (b) rotierender LiDAR mit festen Ebenen [9]

Für Anwendungen im Bereich der Robotik und Fahrzeugtechnik wird meistens jedoch ein rechteckiges FoV angestrebt. Die horizontale Sichtweite sollte so groß wie möglich sein, um Aktivitäten und Objekte im Umfeld so gut wie möglich wahrnehmen zu können. Das vertikale Sichtfeld hingegen muss nicht so groß ausfallen. Im Luftraum, sowie nah am Boden befinden sich weniger bis kaum relevante Informationen. Dementsprechend ist ein kreisförmiges Sichtfeld ungeeignet. Es gilt die horizontale Sicht zu erweitern und die vertikale Sicht zu limitieren [5].

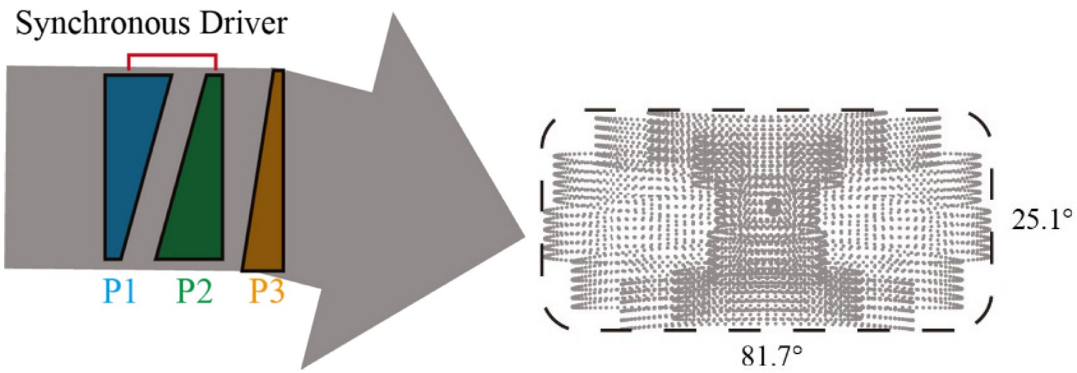


Abbildung 2.7.: Anordnung der Prismen des LIVOX Horizon (links) und erzeugtes Scanning-Muster (rechts) [5]

Der Scanning-Mechanismus auf Basis von Risley-Prismen bietet hier einen Ansatz zur Lösung dieses Problems [5]. Durch das Verwenden eines zusätzlichen dritten Prismas kann das FoV signifikant vergrößert werden. Bei der Verwendung von drei Prismen sind die ersten zwei, welche im Vorherigen bereits erläutert wurden, wie folgt ausgelegt. Sie sind beide identisch. Der Brechungsindex, sowie der Keilwinkel, welcher den Winkel zwischen zwei brechenden Kanten eines Prismas beschreibt, sind dementsprechend gleich. Sie werden dann mit gleicher Geschwindigkeit rotiert, wobei ihre Rotationswinkel immer exakt gegenteilig zueinander sind. Als Ergebnis oszilliert der Lichtstrahl durch die Ablenkung entlang der horizontalen Achse des Sichtfeldes. Das

dritte Prisma wird unabhängig von den ersten zweien mit geringerer Geschwindigkeit angetrieben. Dadurch wird der Lichtstrahl mit einer kleineren Frequenz zusätzlich rotiert und über die Vertikale verteilt. Das Ergebnis ist ein annähernd rechteckiges Muster, erzeugt aus kreisförmigen Bahnen [5]. Die Anordnung der Prismen, sowie das entsprechend erzeugte Scanning-Muster, sind in Abbildung 2.7 dargestellt. Genau dieses Verfahren nutzt der LiDAR-Sensor: Livox Horizon. Er erzeugt so einen sichtbaren Bereich von ungefähr 25° in der Vertikalen und 81° in der Horizontalen [9].

2.2. Sensorik

Der folgende Abschnitt soll einen kurzen Einblick in die technischen Besonderheiten der Sensoren geben, welche im Rahmen dieser Arbeit Anwendung fanden. Zur Auswahl standen mehrere Sensoren. Die Entscheidung fiel dabei auf zwei grundsätzlich unterschiedlich arbeitende LiDAR-Scanner, welche jedoch ähnlich dichte Punktwolken erzeugen.

2.2.1. Livox Horizon

Der im vorherigen Abschnitt erwähnte Livox Horizon (im weiteren Verlauf auch als „Horizon“ oder „Livox“ bezeichnet) findet Verwendung im praktischen Teil dieser Arbeit. Wie bereits im Vorhergehenden erwähnt wurde, ist der Horizon ein mechanischer, mobiler LiDAR-Sensor und nutzt Risley-Prismen zur Verteilung des Laserstrahls. Er erzielt dabei große Reichweiten mit hoher Präzision, in einem großen, rechteckigen FoV. Somit kann der Sensor für Anwendungen im Bereich des autonomatisierten Fahrens, der Robotik oder für Drohnen verwendet werden. Durch die Nutzung von mehreren PLDs und APDs und einem nicht-repetitiven Scanning-Muster wird eine hohe Punktdichte erzielt. Der Horizon verfügt außerdem über einen Dual-Return-Modus, welcher genutzt werden kann, um Punkte von zwei Sensoren gleichzeitig erfassen zu können. So erhöht sich die Punktanzahl auf 480.000 Punkte pro Sekunde [10].

In Abbildung 2.7 wurde bereits das Scanning-Muster des Horizon dargestellt. Die Abbildung 2.8 vergleicht die Punktabdeckung dieses Musters mit traditionellen, rotierenden Sensoren. In der Mitte des FoV ist die Punktdichte vom Horizon am höchsten, mit einem durchschnittlichen Abstand von 0,2° von Linie zu Linie. Traditionelle LiDAR-Sensoren mit 64 Ebenen erreichen im Durchschnitt einen Abstand von 0,3° bis 0,6°. Die kreisrunden Abschnitte an den Seiten (siehe Abbildung 2.7) weisen eine deutlich geringere Auflösung auf und erreichen so einen durchschnittlichen Abstand von nur 0,4°. Bei einer Aufnahmezeit von ungefähr 0,1 s erreicht der Horizon eine Abdeckung des FoV von ungefähren 60 %. Dadurch, dass die Punktdichte mit der Integrationszeit zunimmt, nähert sich diese nach 0,5 s einer Abdeckung von 100 %. Die FoV-Abdeckung nach 0,1 s ist vergleichbar mit der eines LiDAR-Sensor mit 64

Ebenen. Der Sensor mit 16 Ebenen hat hier ein vertikales FoV von 30°, der mit 32 Ebenen 41° und der mit 64 Ebenen erzeugt ein vertikales FoV von 27°. [10]

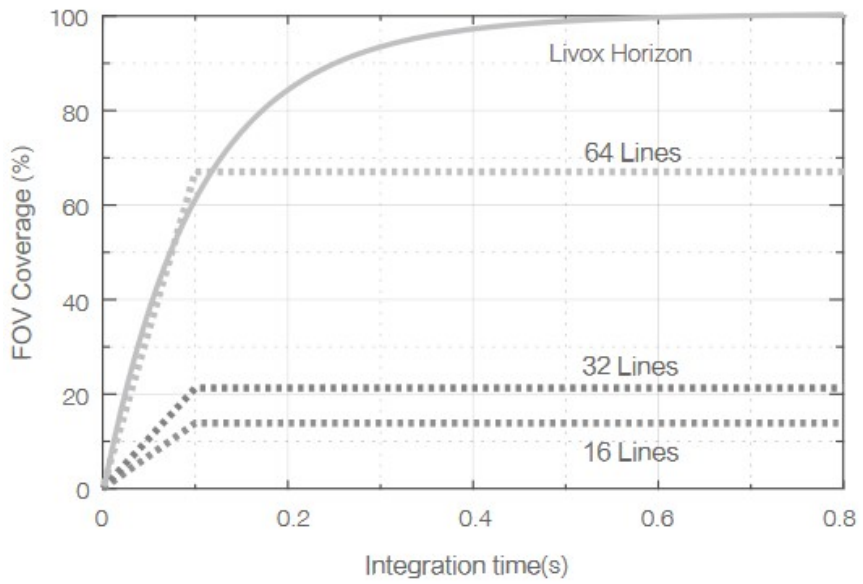


Abbildung 2.8.: Vergleich der FoV-Abdeckung des LIVOX Horizon mit traditionellen LiDAR-Sensoren [10]

Des Weiteren erzeugen die PLDs des Horizon Licht mit einer Wellenlänge von 905 nm. Die erreichbare Messdistanz liegt dabei bei 260 m, insofern das bestrahlte Objekt 80 % des Lichts reflektiert. Bei einer Distanz von 90 m müssen lediglich 10 % reflektiert werden, um ein messbares Signal erzeugen zu können. Zufällige Fehler in der Messung der Distanz liegen bei einer Objektentfernung von 20 m, bei unter 2 cm und der zufällige Winkelfehler bei unter 0,05°. Objekte, die weniger als 0,5 m vom Sensor entfernt sind, können nicht präzise erkannt werden. Bis zu einer Distanz von 3 m können zudem Verzerrungen in der Punktwolke auftreten. [10]

Zusätzlich bietet der Horizon eine bereits verbaute Inertial Measurement Unit (IMU) zur Messung der Orientierung. Es handelt sich dabei um den BMI088 der Bosch GmbH. Der Sensor kombiniert ein Gyroskop sowie einen Beschleunigungssensor. Die IMU besitzt laut Herstellerangaben eine Auflösung von 0,09 mg (Beschleunigung) sowie 0,004° (Gyroskop) und kann über die Lebensdauer hinweg ein Offset von $\pm$ 20 mg bzw. $\pm$ 1 °/s entwickeln [11].

Der Horizon kann Punktwolken als kartesische oder sphärische Koordinaten ausgeben. Befindet sich kein sichtbares Objekt innerhalb der erfassbaren Reichweite, werden die Koordinaten für diesen Punkt mit drei Nulleinträgen für x, y und z ausgedrückt. [10]

2.2.2. Ouster OS1-64

Der OS1 des Herstellers Ouster ist der zweite Lidar-Sensor, der in dieser Arbeit Einsatz findet. Im Gegensatz zum Horizon nutzt der Ouster eine, um die Höhenachse rotierende Messvorrichtung. Dabei kommen 64 Ebenen zum Einsatz. Der verwendete Sensor entstammt der ersten Generation der OS1-Modellreihe. Durch das traditionelle rotierende Messverfahren erzeugt der OS1 eine andere Punktverteilung als der Horizon. Die Verteilung ist repetitiv und somit mit jedem Frame gleich. Dadurch bleiben Leerzeilen bestehen, welche nicht bestrahlt werden (siehe Abbildung 2.6). Es ergibt sich jedoch der Vorteil, dass mit jeder Aufnahme ein Sichtbereich von 360° horizontal abgedeckt wird. Des Weiteren ist die Verteilung der Punkte in jedem Frame gleich. [12]

Mit einer Wellenlänge von 855 nm erreicht der Sensor eine messbare Reichweite von bis zu 110 m, insofern das bestrahlte Objekt 80 % des Lichts reflektieren kann. Die Wahrscheinlichkeit, dass ein Punkt unter diesen Umständen detektiert wird, liegt laut Herstellerangaben bei 90 %. Die maximale Reichweite ist somit ungefähr 150 m geringer als die des Horizon. Beachtet man, dass der Ouster ein horizontales Sichtfeld von 360° abdeckt, so liegt die maximale Distanz zwischen zwei gegenüberliegenden Punkten bei 220 m. Das vertikale FoV erstreckt sich über 33,2° und ist somit ebenfalls größer als das des Livox Horizon. Die horizontale Auflösung, also wie viele Punkte pro Ebene mit einer Umrundung der Messvorrichtung aufgezeichnet werden, kann beim OS1 konfiguriert werden. Die möglichen Einstellungen sind 512, 1024 oder 2048 Punkte pro Ebene. Des Weiteren kann die Rotationsfrequenz auf entweder 10 Hz oder 20 Hz eingestellt werden. So können maximal 1.310.720 Punkte pro Sekunde aufgezeichnet werden. Die minimale messbare Entfernung liegt bei 0,8 m. Die Distanz wird bei lambertschen Reflexionen mit einer Genauigkeit von $\pm$ 5 cm und bei Retroreflexionen mit einer Genauigkeit von $\pm$ 10 cm gemessen. [12]

Der Sensor besitzt ebenfalls eine verbaute IMU, bestehend aus einem Gyro- und Beschleunigungssensor. Diese kann Informationen mit einer Frequenz von 100 Hz ausgeben, mit einer Zeitstempel-Auflösung von $<$ 10 ms. [13]

3. Kartierung und Lokalisierung

In einem Artikel der Internetseite elektroniknet.de beschreibt der Autor Amol Borkar, warum seiner Meinung nach, Fahrzeuge künftig auf fortgeschrittenere Methoden zur Lokalisierung und Kartierung angewiesen sein werden [14]. Borkar selbst arbeitet für das Unternehmen Cadence Design Systems, welches unter Anderem digitale Signalprozessoren (DSPs) für hochentwickelte Kameraanwendungen im Automobilsektor produziert [15].

Borkar erklärt in seinem Artikel, dass die Anzahl implementierter Asssistenzsysteme stetig zunimmt und wie dadurch die Notwendigkeit geschaffen werden würde, genaue Informationen über die Umgebung bereitstellen zu können. Er geht dabei auf den Werdegang der technologischen Entwicklung des Automobils ein und erzählt von der Einführung des GPS. An einem Beispiel erläutert er, dass GPS in Zukunft nicht mehr ausreichen würde, um einem PKW genügend Informationen über Standort und Eigenbewegung zu beschaffen. So könne GPS genutzt werden um zu erkennen, dass man sich an einer Kreuzung befindet, aber nicht in welche Richtung man dabei schaut. Ob man in die richtige Richtung gefahren ist, könne erst erkannt werden, sobald man abgebogen ist. Des Weiteren erklärt er, dass GPS für einige Anwendungen zu ungenau sei und dass eine Verfolgung von sog. „Mikrobewegungen“, Bewegungen die im geringen Meterbereich liegen, nicht möglich sei. Zudem könne nicht an jedem Ort frei auf GPS zugegriffen werden. Laut Borkar könne SLAM diese Lücke schließen. [14]

Das folgende Kapitel befasst sich mit den theoretischen Hintergründen von modernen Kartierungssystemen für mobile Anwendungen sowie Algorithmen für den Abgleich von konsekutiv aufgezeichneten LiDAR-Scans, um diese für den Kartierungsprozess verwenden zu können.

3.1. SLAM - Simultanes Lokalisieren und Kartieren

SLAM beschreibt das Berechnungsproblem, eine Karte der Umgebung zu erstellen und zugleich die Position und Orientierung innerhalb dieser Karte zu bestimmen [16]. Moderne SLAM-Algorithmen erzielen laut Borkar Ergebnisse mit einer Genauigkeit im Zentimeterbereich [14]. Die daraus resultierenden Anwendungsmöglichkeiten seien vielfältig. Da SLAM die Beschreibung eines allgemeinen Problems ist, gibt es viele unterschiedliche Ansätze und Herangehensweisen, oft abhängig davon, welche Sensorik verwendet wird und unter welchen Umweltbedingungen ein Algorithmus erfolgreich funktionieren soll.

Borkar erwähnt, dass SLAM im Automobil häufig als sog. Visual Simultaneous Localization and Mapping (VSLAM) realisiert wird. „Visual“ gibt an, dass der Algorithmus

auf Dateneingaben von Kameras basiert. Es wird über Triangulation die 3D-Position der Kamera ermittelt. Über mehrere Frames hinweg wird dabei eine Menge von Punkten verfolgt, um so die eigene Position und Orientierung abschätzen zu können. Eine typische SLAM-Anwendung sei laut Borkar der Spurhalteassistent. Die genaue Ermittlung von Orientierung und Position ermögliche das Erkennen sowie das Halten einer Spur und zusätzlich Spurwechselvorgänge. Mittlerweile sind neben Kameras auch weitere Sensorik-Systeme verfügbar, die für SLAM zum Einsatz kommen können. [14]

Zum Zeitpunkt des 19. April 2023 existieren in der digitalen Datenbank vom Institute of Electrical and Electronics Engineers (IEEE) 2900 Suchergebnisse, mit der Abkürzung „SLAM“ im Titel des Dokuments und den Begriffen „Simultaneous“, „Localization“ und „Mapping“ im Fließtext. Fügt man der Suche das Wort „Camera“ hinzu, reduziert sich die Anzahl der Ergebnisse auf 1143. Sucht man hingegen nach „LiDAR-SLAM“, so reduziert sich die Anzahl der Ergebnisse auf 310. Von diesen 310 Ergebnissen wurden 306 allein im Zeitraum von 2010 bis 2023 veröffentlicht. [17]

Es ist zu erkennen, dass LiDAR-SLAM eine deutlich geringere Relevanz als beispielsweise VSLAM vorweisen kann. Was zum einen daran liegen könne, dass Kameras seit längerem zu deutlich günstigeren Preisen erworben werden können und die Entwicklung von LiDAR-Sensoren verhältnismäßig jung ist. Im Vergleich zu VSLAM bietet LiDAR-SLAM jedoch einige markante Vorteile. Durch die Nutzung von dreidimensionalen Punktwolken kann LiDAR-SLAM auf räumlich erweiterte Eigenschaften zugreifen. So können anstatt einzelner Punkte, ganze Liniensegmente und Flächenstücke extrahiert und in den Algorithmus eingebunden werden [18]. Des Weiteren bietet das Nutzen von Punktwolken den Vorteil, dass die befahrene Gegend als digitales 3D-Modell nachgebildet werden kann. Zudem werden LiDAR-Sensoren nicht von Belichtungsverhältnissen beeinflusst, was das Arbeiten bei schlechten Lichtverhältnissen ermöglicht.

Im Jahr 2006 wurde im „IEEE Robotics & Automation Magazine“ eine Zusammenfassung der bis dato relevantesten Forschungsergebnisse im Bereich SLAM veröffentlicht. Die Autoren Hugh Durrant-Whyte und Tim Bailey fassen im Artikel die grundlegenden Punkte der Problematik zusammen und gehen auf akutelle Forschungserkenntnisse ein [16], [19]. Bis heute sind einige neue Ansätze entwickelt worden, die zur Zeit des Artikels noch nicht publik waren, vor allem im Bereich des LiDAR-SLAM. Dazu gehört LiDAR Odometry and Mapping (LOAM), ein Algorithmus welcher von Ji Zhang und Sanjiv Singh im Jahr 2014 veröffentlicht wurde [20]. Lightweight and Ground-Optimized (LeGO)-LOAM ist eine von vielen Fortführungen von LOAM und wurde 2018 veröffentlicht [21].

Die aktuellen Ergebnisse der „Karlsruher Institut für Technologie & Toyota Technological Institute (KITTI) Vision Benchmark“ geben einen Einblick in die Leistungsfähigkeit von LiDAR-SLAM [22]. Diese Benchmark (Maßstab) wurde 2012 in Zusammenarbeit beider Institute entwickelt. Es handelt sich dabei um mehrere Datensätze an Aufzeichnungen verschiedenster Sensorik aus diversen Verkehrssituationen. Der Datensatz ist mittlerweile ein beliebtes Vergleichsmittel für SLAM-Algorithmen und wird häufig im Zuge neuer Veröffentlichungen als Vergleichsdatensatz erwähnt. Auf der offiziellen Webseite werden alle eingereichten Algorithmen nach ihrer Genauigkeit auf einer Rangliste sortiert. Von den ersten zehn Plätzen arbeiten neun davon mit den Aufzeichnungen des LiDAR-Sensors. Dazu gehört unter anderem der schon zitierte LOAM-Algorithmus [20], welcher auf Platz drei landet. Es schaffen zudem zwei weitere Fortführungen von LOAM unter die ersten zehn Plätze. KISS-ICP, ein weiterer LiDAR-SLAM Algorithmus, schafft es auf Platz zehn. KISS-ICP wurde an der Uni Bonn entwickelt und fokussiert sich darauf den Algorithmus so simpel wie möglich zu halten. Er erreicht unter alleiniger Verwendung der LiDAR-Daten sehr hohe Genauigkeiten [23]. Anhand der Rangliste ist zu erkennen, dass LiDAR-Sensoren im Bereich der Kartierung und Lokalisierung vorteilhaft sein können. Den ersten Platz belegt jedoch ein Algorithmus, welcher auf der Verwendung von Stereo-Kameras basiert (Stand: 24.April 2023).

3.1.1. Funktionsprinzip LiDAR-SLAM

SLAM wird oft über wahrscheinlichkeitsbehaftete Ansätze gelöst, da bei der Messwertaufnahme Unsicherheiten bestehen. Folgend wird die geschätzte Position eines Roboters, welcher durch eine unbekannte Umgebung navigiert nicht als exakt sondern als Wahrscheinlichkeitsverteilung angesehen [24]. Nach Brian Douglas [25] kann man diese SLAM-Algorithmen nocheinmal in zwei Obergruppen, Filtering (filternd) und Smoothing (glättend), einteilen. Filternde Algorithmen schätzen den aktuellen Zustand durch die jeweils neueste Messung [25]. Dazu gehören beispielsweise Ansätze, die auf der Nutzung eines Kalman-Filter (KF) oder Partikelfilter (PF) basieren [16]. Smoothing-Algorithmen schätzen die vollständige Trajektorie eines Roboters unter Nutzung aller gesammelten Messaufzeichnungen [25]. Zu Smoothing-Algorithmen gehört unter anderem Graph-Based (kurvenbasiertes) SLAM. Im Weiteren wird ausschließlich auf kurvenbasiertes SLAM und dessen Anwendung im Bereich der LiDAR-Sensorik eingegangen.

Zu Beginn sind einige wenige Begriffe, die im Zusammenhang mit kurvenbasierten SLAM häufig verwendet werden, zu klären. Der Begriff „Pose“ ist allgegenwärtig im Bereich der Kartierung und Lokalisierung. Er wird als Bezeichnung für die Position und dazugehörige Orientierung eines Roboters bzw. Sensors zu einem bestimmten Zeitpunkt, verwendet. Um Verwechslungen zu vermeiden, wird im Folgenden anstelle

des englischen Begriffs, das deutsche Äquivalent „Position“ verwendet. Eine Ansammlung von aufeinanderfolgenden Positionen erzeugt die Trajektorie eines Roboters oder Fahrzeugs. Innerhalb des Graphen werden diese Positionen auch als „Nodes“ bzw. Knoten bezeichnet, um eine Unterscheidung zwischen den realen und den ermittelten Positionen gewährleisten zu können. Durch das Zuordnen der jeweiligen LiDAR-Scans zu ihren korrespondierenden Knoten wird es ermöglicht, den befahrenen Raum zu rekonstruieren [26]. Die Ansammlung mehrerer Knoten wird als „Posegraph“ bzw. Positionsgraph bezeichnet. Dieser stellt die geschätzte bzw. berechnete Trajektorie dar. Knoten werden durch sog. „Edges“ (Kanten) miteinander verbunden. Diese setzen sich aus der relativen Beziehung zwischen zwei Knoten zusammen und beinhalten somit Informationen über die räumliche Transformation zwischen ihnen. Zudem können sie genutzt werden um die Sicherheit anzugeben, mit der die relative Beziehung zwischen zwei Knoten ermittelt wurde, auch als „Constraints“ (Einschränkungen) bezeichnet [25], [24]. Kurvenbasiertes SLAM beruht somit auf der Idee, die Trajektorie des Roboters bzw. des Sensorsystems als Graph nachzustellen. Dabei werden Stützpunkte an unterschiedlichen Zeitpunkten bzw. Orten genutzt, um den Graphen zu modellieren. Diese sind wiederum durch die räumlichen Transformationen miteinander verbunden. Je nach Algorithmus werden zusätzliche Optimierungsverfahren verwendet um den Fehler der Knoten-Konfiguration zu minimieren [24].

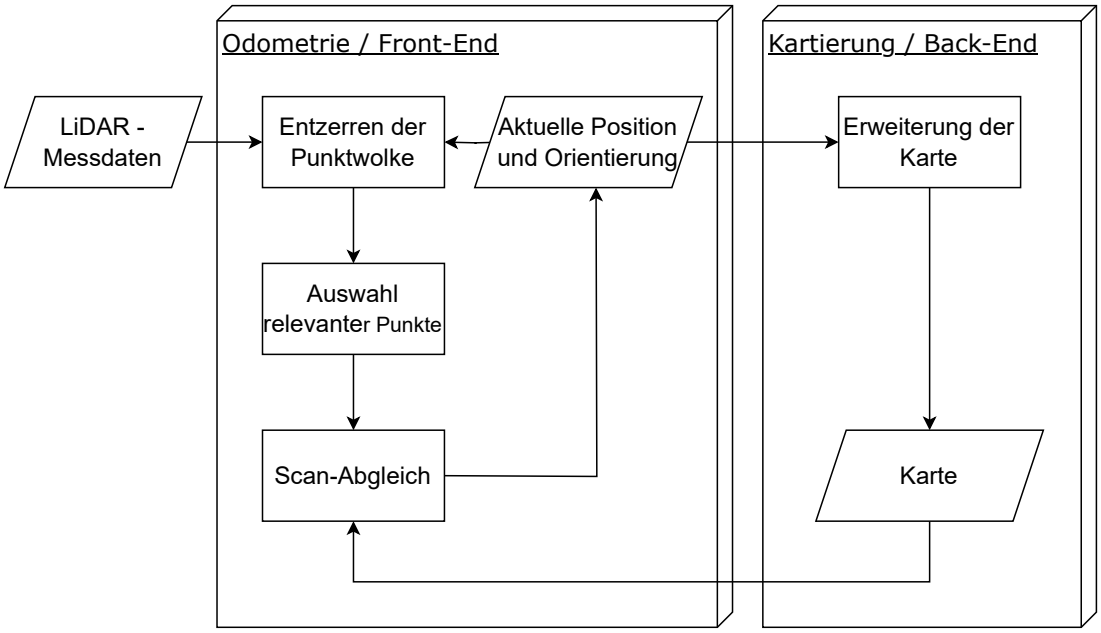


Abbildung 3.1.: Allgemeingültiger PAP eines LiDAR-SLAM-Algorithmus [27]

Abbildung 3.1 zeigt einen allgemeinen Programmablaufplan (PAP) von LiDAR-SLAM. Dabei werden Algorithmen oft in Front- und Back-End unterteilt. Der vordere Teil des Algorithmus (Front-End) wird verwendet um die Odometrie, bzw. Bewegung des Sensors zu schätzen. Der hintere Teil des Algorithmus (Back-End) ist demnach für den Prozess der Kartierung verantwortlich. Die bereits zitierten Algorithmen LOAM

[20], LeGO-LOAM [21] und KISS-ICP [23] arbeiten in ihren Grundzügen nach diesem allgemeinen Ablauf.

Im ersten Schritt wird die eingelesene Punktwolke entzerrt. Aus dem Grund, dass der Sensor während des Scannens in Bewegung ist, kommt es dazu, dass der Abstand zwischen dem Sensor und einem aufgezeichnetem Objekt variieren kann. Das Resultat ist eine verschwommene Darstellung des Objekts. Bei sehr geringen Geschwindigkeiten wirkt dieser Effekt nur minimal, da die Aufnahmezeit pro Frame der meisten Sensoren sehr kurz ist. Bei höheren Geschwindigkeiten macht sich dieser Effekt jedoch bemerkbar. Um die tatsächlichen Punktpositionen berechnen zu können, wird die letzte Information, über die Position des Sensors, ausgewertet [27]. Eine der simpelsten Methoden zur Kompensierung von Verzerrungen ist das Modell der konstanten Geschwindigkeit (MdkG). Es wird angenommen, dass sich der Sensor mit gleichbleibender Translations- und Rotationsgeschwindigkeit bewegt. So kann eine Punktwolke mittels den zuvor berechneten Transformationen entzerrt werden. Da die Aufnahmedauer pro Punktwolke üblicherweise zwischen 0,05 s bis 0,1 s liegen, sind Änderungen in den Geschwindigkeiten von Frame zu Frame sehr gering, dies trifft jedenfalls für viele Applikationen in der Robotik zu. KISS-ICP arbeitet nach dem selben Prinzip und erreicht auch in Automobilanwendungen hohe Genauigkeiten [23].

In der folgenden Abbildung 3.2 wird dargestellt, wie die Eigenbewegung des Sensors die Punktwolke verzerrt. Die grünen Linien sind Reflektionen des Bodens. Es ist zu erkennen, dass diese im unberichtigten Zustand, kontinuierlich verlaufen. Das kommt daher, dass der Boden zu jedem Zeitpunkt den selben Abstand zum Sensor hat, auch während einer Bewegung. Nachdem die Wolke entzerrt wurde, liegt nun ein Versatz vor, da die tatsächlich gescannten Punkte vom Anfang und Ende nicht die gleichen sein können. Die rote Linie visualisiert ein Schild, welches in Fahrtrichtung steht. Da der Abstand zum Schild mit Fortlaufen der Messung geringer wird, ist in der unberichtigten Punktwolke ein Versatz zwischen Beginn und Ende der Messung zu erkennen. Nach der Korrektur liegen beide Linien auf gleicher Höhe. [28]

Bevor die aktuelle Punktwolke mit der vorherigen verglichen wird, um eine Transformation zu berechnen, wird ein „Sampling“ (Auswahl) vorgenommen. Diese Auswahl von Punkten äußert sich in einem Reduzieren der aufgezeichneten Punktwolke, in tatsächlich relevante Punkte. Da Punktwolken aufgrund der hohen Aufnahmegeschwindigkeiten sehr groß werden können, ist es nötig diese auf wertvolle Punkte zu reduzieren, um Rechengeschwindigkeiten niedrig zu halten. Es gilt invalide Punkte und solche die wenig Informationsdichte zum Scan-Abgleich beitragen, zu entfernen. Der Livox Horizon notiert invalide Punkte als Nullzeile, so können diese einfach erkannt und entfernt werden. Welche Punkte relevant sind, hängt von der Methode des

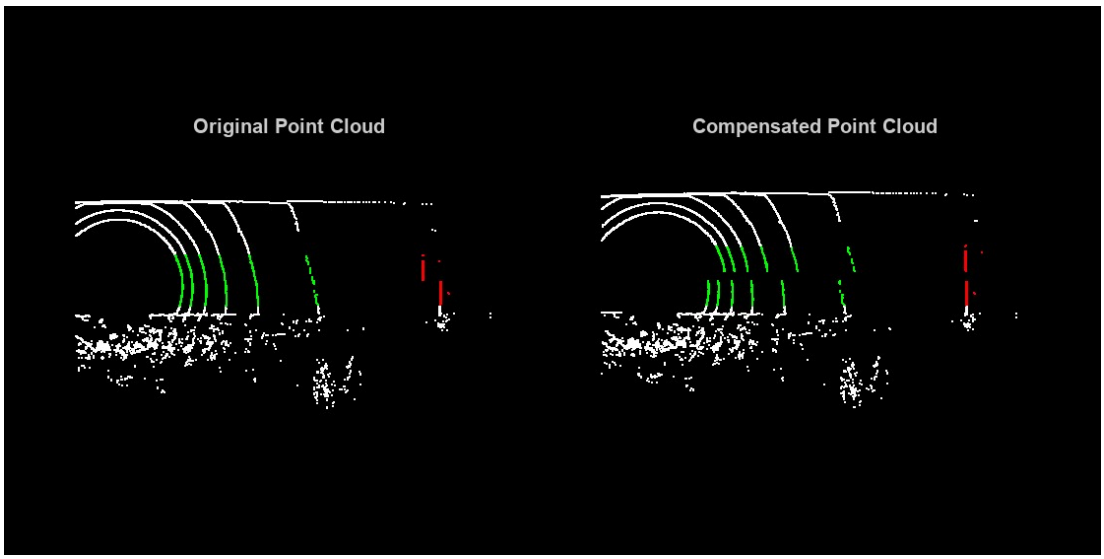


Abbildung 3.2.: Vergleich zwischen nicht-berichtigter Punktwolke (links) und berichtigter Punktwolke (rechts) eines rotierenden LiDAR-Sensors [28]

Scan-Abgleichs ab. LOAM nutzt dafür Punkte die auf scharfen Kanten und planaren Flächen liegen [20]. LeGO-LOAM nutzt zusätzlich Punkte, die auf dem Boden liegen [21]. Je nach Anwendungsgebiet kann es dazu kommen, dass dynamische Objekte, wie andere Fahrzeuge, vom Scanner aufgezeichnet werden. In solchen Szenarien wird versucht nach Punkten zu suchen, die statisch sind und für einen längeren Zeitraum sichtbar bleiben. So können bewegte Objekte aus den Punktwolken entfernt werden [27].

Nachdem die aktuellste Punktwolke entzerrt und gefiltert wurde, wird ein sog. „Scan-Matching“ (Abgleich zweier oder mehrerer Punktwolken) durchgeführt, um die letzte Position und Orientierung des Sensors zu berechnen. In den folgenden Unterkapiteln 3.2 und 3.3 werden zwei grundlegende Scan-Abgleich-Algorithmen ausführlich erklärt. KISS-ICP reduziert die Punktwolke für den Scan-Abgleich ein zweites Mal, um schnellere Rechenzeiten zu erzielen. Für die Abspeicherung in der Karte wird jedoch die einmal gefilterte Punktwolke verwendet, um eine hohe Auflösung beizubehalten [23]. Der Abgleich mit älteren Punktwolken kann dabei mit dem zuvor aufgezeichneten Frame, mehreren Frames oder der gesamten Karte geschehen. KISS-ICP vollzieht einen Frame-zu-Karte Abgleich, mit der Erklärung, dass so eine robustere Ausrichtung der neuen Punktwolke geschehen kann [23]. Mit der berechneten Position und Orientierung kann ein neuer Knoten dem Graphen und so der LiDAR-Frame, der Karte hinzugefügt werden. Des Weiteren kann die darauffolgende Punktwolke mittels der berechneten Transformation entzerrt werden.

3.1.2. Graph-Optimierung durch Schleifenschluss-Erkennung

Aufgrund dessen, dass mit jeder Iteration eines solchen Algorithmus Fehler bestehen bleiben, wächst der Versatz zwischen dem modellierten Graphen und der tatsächlichen Trajektorie. Da die Karte in direktem Zusammenhang zum Graphen steht, weicht auch sie immer mehr von der realen Umgebung ab. Dieser Effekt kann durch Optimieren der oben beschriebenen Schritte minimiert werden. Es existieren jedoch auch andere Wege, um diesem Versatz entgegenzuwirken.

Grafik 3.3 zeigt einen Roboter, welcher sich in kreisförmiger Bahn durch einen Raum (in grau dargestellt) bewegt. Dabei nimmt er mit einem LiDAR-Sensor mehrere Punktwolken, zu unterschiedlichen Zeitpunkten auf. Die Punktwolken werden in schwarzen gestrichelten Linien dargestellt. Die Frames werden von Knoten zu Knoten miteinander verglichen, um eine Transformation rückführen zu können. Der Versatz wird im Beispiel bewusst überproportional dargestellt. Es ist zu erkennen, dass der letzte berechnete Knoten, trotz Wiederkehr zum realen Startpunkt, einen Versatz aufweist.

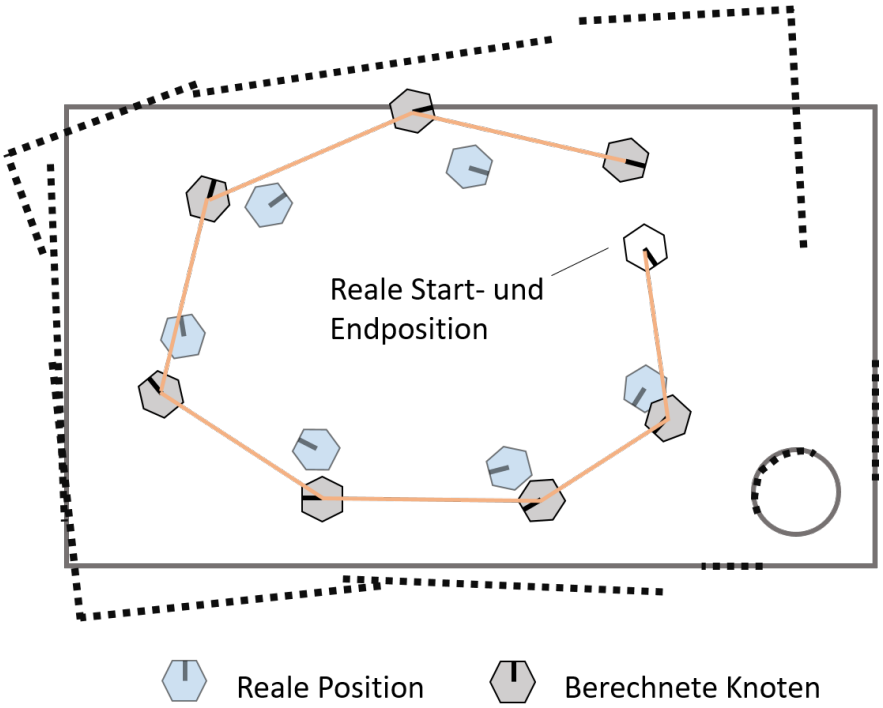


Abbildung 3.3.: Vergleich zwischen realer Positionen (blau) und geschätzten Positionen (grau) eines sich bewegenden LiDAR-Roboters

Um diesen Versatz zu minimieren, können Mechanismen implementiert werden, welche ein erneutes Befahren von bereits besuchten Orten erkennen können. Solche Punkte werden „Loop Closures“ (Schleifenschluss) genannt. Wird ein solcher Punkt korrekt erkannt, kann eine Verbindung zwischen den zwei Knoten hergestellt werden. Mittels Scan-Abgleich-Algorithmen wird wiederum die zugehörige Transformation ermittelt. Werden diesen Knotenverbindungen Wichtungen zugeordnet, kann der Graph im

Nachhinein angepasst werden, was zu einer insgesamt genaueren Darstellung führen kann. Falsche Detektionen solcher Punkte können entsprechenderweise einen ebenso negativen Effekt auf die Genauigkeit des Graphen und der Karte haben. [25]

Um das Beispiel aus Abbildung 3.3 fortzuführen, zeigen die Grafiken aus Abbildung 3.4 inwiefern der Graph durch das Erkennen einer Schleife optimiert werden kann. Der Start- und Endpunkt der Fahrt stellt hier den Schleifen-Schluss dar, die erste LiDAR-Aufzeichnung ist in grün und die zweite in rot dargestellt. Durch graue Linien werden die korrespondierenden Punkte der beiden Scans miteinander verbunden, um den Prozess des Abgleichs zu visualisieren.

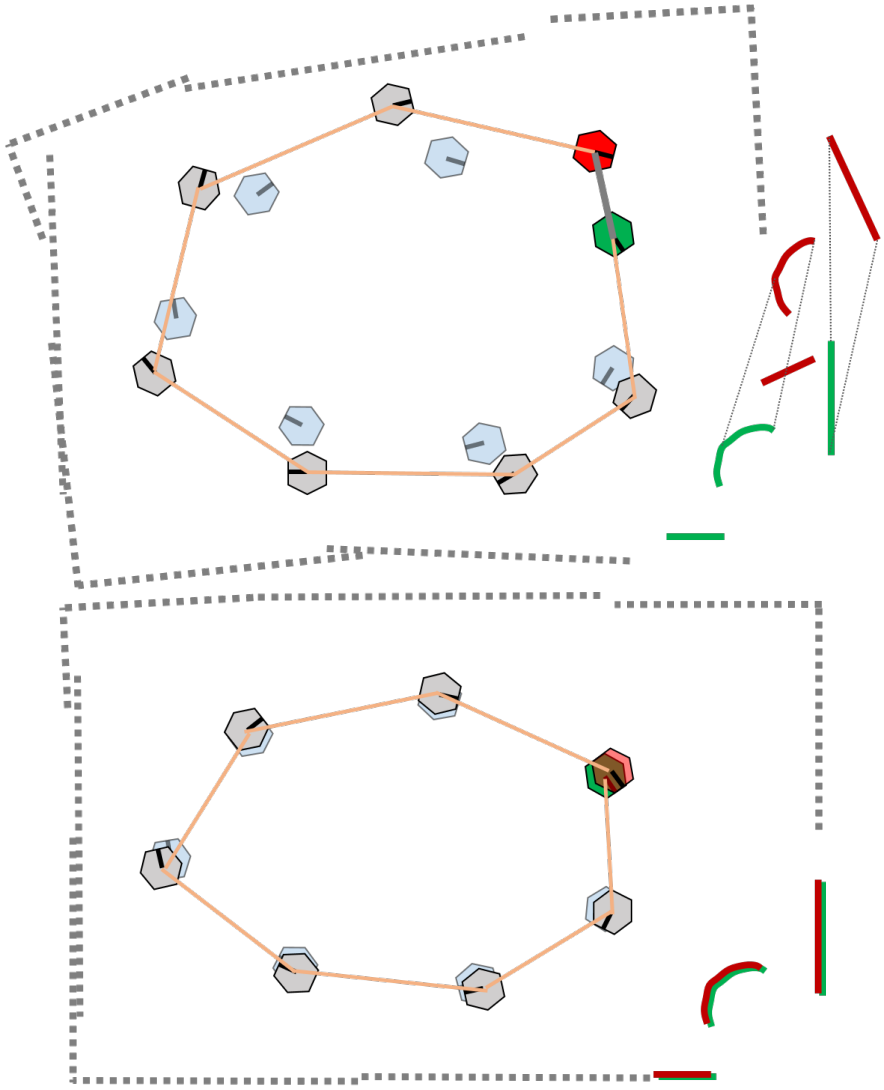


Abbildung 3.4.: Graph vor Schleifen-Korrektur (oben) und Graph nach Korrektur (unten)

Wie in der unteren Grafik der Abbildung 3.4 zu erkennen ist, werden auch vorherige Knoten durch die Erkennung einer Schleife korrigiert. Dies verdeutlicht den allgemeinen Vorteil von Algorithmen, welche auf der Nutzung von Graphen basieren. Die erwähnten Methoden KISS-ICP und LOAM verzichten jedoch auf Schleifen-Erkennungen [23], [20]. Die Autoren von LOAM erwähnen in ihrem Artikel, dass eine

solche Funktion zukünftigen Fortführungen hinzugefügt werden soll. KISS-ICP legt einen großen Wert auf Simplizität, weshalb eine solche Erkennung nicht implementiert wurde [23].

Zur Erkennung von bereits besuchten Orten werden Methoden benötigt, die schnelle Rechenzeiten garantieren. Dies kann nicht mit Scan-Abgleich-Algorithmen sichergestellt werden, da ein Abgleich eines neu aufgezeichneten Scans mit jedem vorherigen viel zu rechenintensiv wäre. Eine mögliche Lösung bietet die Nutzung von sog. Scan-Kontext-Deskriptoren (SKDs). Diese Deskriptoren vereinen bestimmte Merkmale von Punktwolken in deutlich kleineren Informationsmatrizen. Ein Abgleich dieser Deskriptoren ist vergleichsweise schnell und kann neben dem eigentlichen Scan-Abgleich geschehen. Die Autoren Giseop Kim und Ayoung Ki entwickelten 2018 eine Methode die nach diesem Prinzip arbeitet [29] und veröffentlichten 2021 eine Weiterentwicklung ihrer vorherigen Methode [30]. Im Gegensatz zu den anderen erwähnten SLAM-Algorithmen existiert bereits eine öffentlich verfügbare Variante von LeGO-LOAM, welche von Giseop Kim bereitgestellt wurde und diese SKDs nutzt [31]. Des Weiteren wurde die erste Variante der SKDs bereits in Matlab implementiert [32].

3.2. ICP - Iterativ-nächster-Punkt-Algorithmus

Der im vorherigen Abschnitt erwähnte KISS-ICP Algorithmus verwendet eine der ältesten Scan-Abgleich-Methoden. In den folgenden Unterabschnitten wird diese Methode erläutert sowie verschiedene Varianten dieser erklärt. Es handelt sich dabei um den Algorithmus des sog. Iterative Closest Point (ICP) oder auch iterativ-nächster-Punkt-Algorithmus. Dieser ist ein Registrierungsalgorithmus für zwei- und dreidimensionale Formen und Strukturen. In den frühen 1990er Jahren wurden drei Abhandlungen veröffentlicht, welche den Grundstein für moderne Scan-Abgleich-Algorithmen legten und bis heute Anwendung finden. Die am meisten zitierte Abhandlung des ICP-Algorithmus stammt von Besl und McKay aus dem Jahr 1992 [33]. Sie beschreiben in ihrem Artikel die direkte Registrierung von 3D-Formen als Punktwolken oder geometrische Darstellungen. Chen und Medioni befassten sich mit dem spezielleren Problem, Laserscans zueinander auszurichten und führten damit die Punkt-zu-Fläche (PzF) Metrik von ICP ein [34]. Zhang entwickelte fast zeitgleich seine eigene Variante des ICP-Algorithmus, welche jedoch zusätzlich eine Methode beinhaltet um Ausreißer-Punkte, sog. „Outliers“, einer Punktwolke zu ignorieren [35].

3.2.1. Funktionsprinzip

Im Feld der LiDAR-Kartierung wird ICP genutzt, um zwei aufeinander folgende Punktwolken, welche einen überlappenden Bereich aufweisen, in einem gemeinsamen Koordinatensystem auszurichten. Das zugrundeliegende Ziel von ICP liegt darin, die

bewegte Punktwolke zu transformieren, sodass die Summe der Quadrate der euklidischen Abstände, zwischen Punkten aus der bewegten Punktwolke, zu den nächsten Punkten in der fixierten Punktwolke, minimiert werden [4]. Die Methodik ist simpel und dient demnach für viele SLAM-Anwendungen als Grundalgorithmus [36]. Der grundlegende Ablauf eines Algorithmus, welcher auf der ICP-Methodik basiert, weist üblicherweise die in Abbildung 3.5 dargestellte Struktur auf .

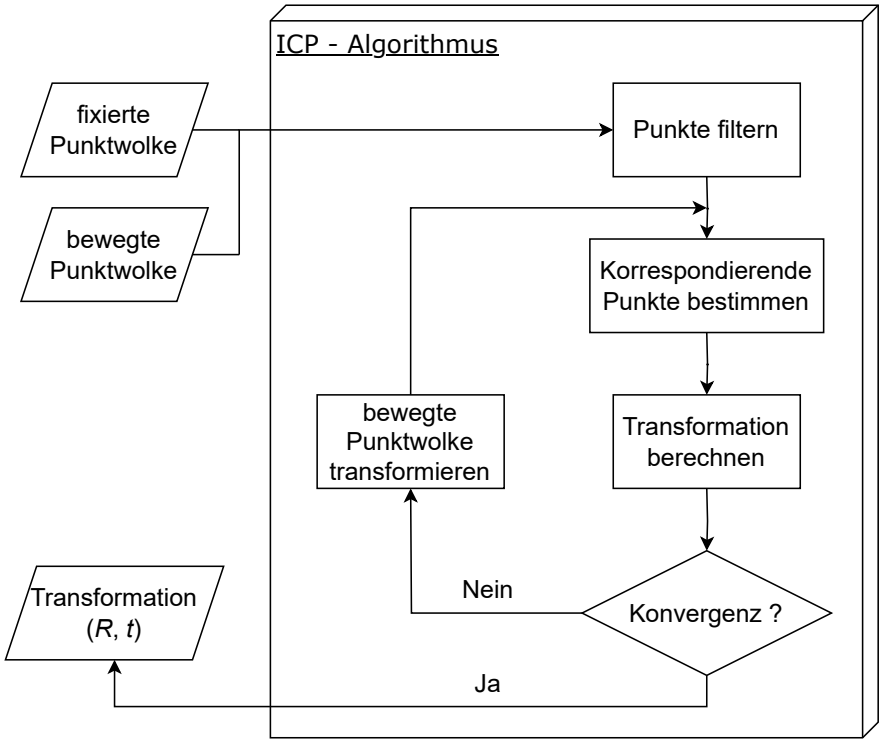


Abbildung 3.5.: PAP eines allgemeinen ICP-Algorithmus

Die sog. fixierte Punktwolke ist die, auf welche eine rückführende Transformation der zweiten durchgeführt wird. Die darauffolgende Punktwolke wird als bewegte Punktwolke bezeichnet. Aus beiden Punktwolken wird eine bestimmte Menge an Punkten ausgewählt, die für den Registrierungsalgorithmus von Wert sind. Im nächsten Schritt werden die miteinander korrespondierenden Punkte beider Punktwolken bestimmt. Hierfür gibt es verschiedene Ansätze. Die Punkt-zu-Punkt (PzP)-Metrik [33] bestimmt zwei Punkte als korrespondierendes Punktepaar, insofern sie die jeweils am nächsten zueinander gelegenen Punkte sind. Im Anschluss können je nach verwendeter Methodik Wichtungen verteilt werden und Ausreißer herausgefiltert werden. Wurden die Relationen zwischen den Punkten bestimmt, folgt als nächstes die Berechnung der Rotation R und der Translation t. Im Anschluss werden R und t auf die bewegte Punktwolke angewendet. Darauffolgend wird der weiterhin bestehende Fehler von R und t ausgewertet. Liegt dieser Fehler über dem festgelegten Minimum und ist kleiner als vorher, so wird der Algorithmus wiederholt. Nach Erfüllen einer der sog. Konvergenzkriterien wird das finale Ergebnis ausgegeben [37].

Cyrill Stachniss, Professor an der Uni Bonn im Bereich der Robotik und Photogrammetrie [38] und Mitbegründer des KISS-ICP Algorithmus [23], beschreibt das elementare Ausrichtungsproblem von ICP wie folgt [39]. Existieren zwei Punktwolken Q und P, welche einen gewissen Anteil an Überlappung aufweisen, so existieren auch die Punktkorrespondenzen C, welche die am nächsten zueinander liegenden Punkte aus Q und P indizieren. Gesucht wird die Transformation T, bestehend aus Rotation R und Translation t, welche den in Formel (3.4) beschriebenen Zusammenhang herstellen kann.

$$Q = \{q_1, \ldots, q_I\} \tag{3.1}$$

$$P = \{p_1, \ldots, p_J\} \tag{3.2}$$

$$C = \{(i, j)\} \tag{3.3}$$

$$Q = R \cdot P + t \tag{3.4}$$

Die Rotation R, sowie die Translation t, welche als Gesamtheit die Transformation T (3.5) darstellen, werden für diese Arbeit nach den Formeln (3.6) und (3.7) definiert. Um eine simplere Darstellung ausgewählter Algorithmen gewährleisten zu können, wird unter Umständen auf eine zweidimensionale Darstellung zurückgegriffen. Die Gleichung (3.7) gibt zudem die für diese Arbeit verwendete Konvention, der Verkettung von Rotationen an, welche nach der von Matlab verwendeten Konvention gewählt wurde [40].

$$T := (R, t) \tag{3.5}$$

$$t = \begin{pmatrix} t_x & t_y & t_z \end{pmatrix}^T \tag{3.6}$$

$$R = R_z \cdot R_y \cdot R_x \tag{3.7}$$

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix} \tag{3.8}$$

$$R_y(\beta) = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix} \tag{3.9}$$

$$R_z(\gamma) = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.10}$$

Die Formeln (3.8) bis (3.10) definieren die Rotationsmatrizen um die jeweilige Achse und repräsentieren in der gegebenen Reihenfolge Nicken, Rollen und Gieren.

3.2.2. Punkt-zu-Punkt-Metrik

Die PzP-Metrik [33] ist die elementarste Variante von ICP. Sie nimmt an, dass Punktepaare aus P und Q mit dem geringsten direkten Abstand zueinander korrelieren. Resultierend wird angenommen, dass jeder Punkt aus P einen zugehörigen Punkt aus Q besitzt. Es wird nicht beachtet, dass in reellen Messungen nur Teile der Punktwolken einen überlappenden Bereich aufweisen. Des Weiteren wird durch die PzP-Metrik angenommen, dass die einzelnen korrespondierenden Punkte überlappen [41]. In Messungen wird jedoch nur durch Zufall zweimal der exakt gleiche Punkt einer Oberfläche gescannt. Es werden zwar die gleichen Objekte erfasst, meist jedoch unterschiedliche Punkte dieser Objekte. Es ist zu erkennen, dass die PzP-Metrik an Grenzen stößt, die eine Berechnung der optimalen Transformation verhindern oder zumindest einschränken.

Da in reellen Aufzeichnungen die Punkte aus Q und P keine korrekte Korrespondenz aufweisen, wird die Rotation R und Translation t gesucht, unter welcher die Summe der quadrierten Abstände minimal wird. So gilt es den folgenden Term (3.11) zu minimieren, um die optimale Transformation zwischen beiden Punktwolken zu finden.

$$e^2 = \sum_{(i,j)\in C} || \, q_i - Rp_j - t \, ||^2 \tag{3.11}$$

Die Gleichung (3.11) bestimmt zudem den bestehenden Fehler e zwischen beiden Punktwolken. Dieser Wert wird Root Mean Square Error (RMSE) genannt, also Wurzel der quadrierten Abstände zwischen den festgelegten Punktepaaren. Er wird dementsprechend als Schwellwert genutzt, um den Algorithmus bei Erreichen einer festgelegten Genauigkeit zu terminieren. Zur Bestimmung der Transformation T können verschiedene mathematische Vorgehensweisen eingesetzt werden. Das Ziel ist dabei immer die Zielfunktion, in dem Fall e^2, zu minimieren, um so die Werte für R und t zu bestimmen [4].

Im Falle dessen, dass die Punktwolken P und Q die gleichen Punkte besitzen, also eine eindeutige Assoziation zwischen allen Punkten hergestellt werden kann, so existiert eine direkte optimale Lösung. Das bedeutet, dass keine initiale Schätzung der Punktkorrespondenzen vorgenommen werden muss, da diese eindeutig und bekannt sind. Die Transformation könnte so in einem Schritt berechnet werden, wie in Abbildung 3.6 anhand einer zweidimensionalen Kurve dargestellt wird [39].

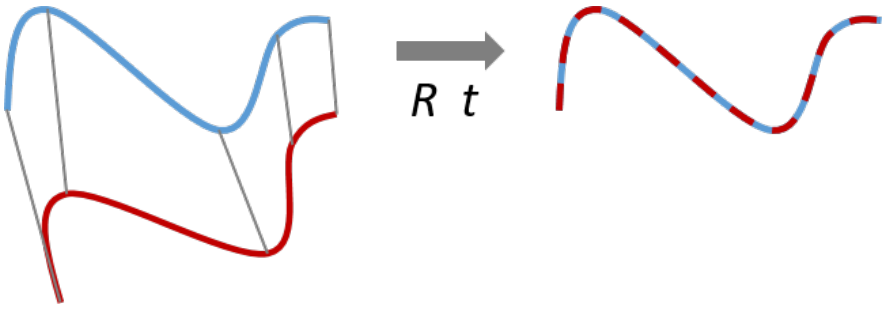


Abbildung 3.6.: Direkte Lösung des Transformationsproblems durch eindeutige und korrekte Assoziation der Punkte aus beiden Kurven [39]

Da die korrespondierenden Punktepaare in reellen Messungen unbekannt sind oder gar nicht existieren, weil nicht zweimal die gleichen Punkte in der Umgebung vom Laser bestrahlt wurden, ist eine direkte Lösung des Problems in der Praxis nicht möglich. Demnach muss der Algorithmus iteriert werden, wie der Name ICP beschreibt. Die Assoziationen zwischen Punkten werden durch verschiedene Metriken geschätzt, wie zum Beispiel durch die PzP-Metrik. Das Beispiel in Abbildung 3.6 zeigt eine Transformation mit korrekten Punktassoziationen. Wie eine solche Transformation nach einer Iteration mit unbekannten, geschätzten Assoziationen aussieht wird beispielhaft in der darauffolgenden Abbildung 3.7 dargestellt. Im Vergleich zu Abbildung 3.6 ist zu erkennen, dass die geschätzten Assoziationen zwischen den Punkten, nicht den tatsächlich korrespondierenden Punkten entsprechen. Als Folge kann nicht die tatsächliche Transformation bestimmt werden. Mit jeder Iteration wird das Ergebnis genauer, da die tatsächlich korrespondierenden Punkte näher aneinander rücken.

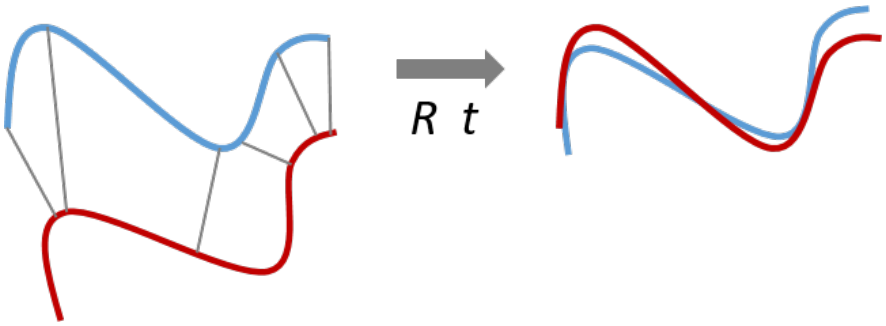


Abbildung 3.7.: Mögliches Ergebnis nach einer Iteration des ICP-Algorithmus mit geschätzten Punktepaaren [39]

Mithilfe der berechneten Werte für R und t kann anschließend die Punktwolke P transformiert und der RMSE-Wert erneut bestimmt werden. Anhand von vorhegehend festgelegten Konvergenzkriterien wird entschieden, ob der Algorithmus beendent oder erneut iteriert wird. Eine detailierte Erläuterung der mathematischen Vorgehensweise ist dem Anhang zu entnehmen (siehe Anhang A.1).

Um die Genauigkeit zu verbessern und die Iterationsanzahl auf ein Minimum zu reduzieren, wurden Methoden entwickelt, die den ICP-Algorithmus optimieren. Es können dabei die vier folgenden Punkte als Stellschrauben betrachtet werden, welche eine Möglichkeit bieten, Algorithmen anwendungsorientiert zu verbessern [37].

- Samplingmethode - Auswahl der relevanten Punkte
- Wichtung von Punktepaaren
- Strategien zum Herausfiltern von Ausreißer-Punkten
- Zielfunktion, Strategie der Punkt-Assoziationen

Die Auswahl der Punkte, welche für den Abgleich genutzt werden, auch Sampling genannt, hat einen Einfluss auf das Ergebnis. Das Verwenden aller verfügbaren Punkte bietet den Vorteil, dass keine Informationen verloren gehen. Dabei entstehen jedoch auch Nachteile. Die nötige Rechenzeit wird maximiert und Fehlerquellen werden aktiv in den Abgleich einbezogen. Daher wäre ein sinnvoller Schritt, abhängig von der Dichte, ein „Downsampling“, also ein Reduzieren der Punktanzahl, vorzunehmen. Dies kann so geschehen, dass eine gleiche Verteilung der Punktdichte über die gesamte Punktwolke hinweg erzeugt wird, auch bekannt als „Uniform Downsampling“ (uniformes Reduzieren) oder Voxelisierungsfilter. Abhängig von der Zielfunktion, die minimiert werden soll, können auch speziell die Punkte erhalten bleiben, welche eine hohe Informationsdichte bieten. So kann nach Punkten gefiltert werden, die größere Normal-Schwankungen aufweisen, also Punkte, die eine vermeintliche Kante oder Ecke abbilden. Alternativ kann nach Punkten gesucht werden die geringe Normal-Schwankungen aufweisen und demnach wahrscheinlich in einer Fläche liegen. Diesen Ansatz kann man insofern erweitern, dass speziell nach geometrischen Formen gesucht wird und nur deren Punkte mit in den Abgleich einbezogen werden. Dieser Ansatz wird als „Feature-Based-Sampling“ (merkmalbasiertes Abtasten) bezeichnet [37]. Solche Samplingmethoden werden in den bereits erwähnten Algorithmen LOAM [20] und LeGO-LOAM [21] verwendet. Grafik 3.8 veranschaulicht die Unterschiede zwischen einem voxelisierenden Filter, einer Methode die nach Normalschwankungen, sowie einer Methode die nach Flächen filtert.

Bei der Verwendung eines allgemeinen ICP-Algorithmus hat es keinen Mehrwert nach jenen Normal-Eigenschaften zu filtern, da diese in der Zielfunktion (3.11) nicht berücksichtigt werden. Einen Vorteil bieten hier Samplingmethoden, welche die Punktwolken vereinheitlichen, um etwa ungünstige Scanningmuster zu kompensieren oder welche die Rauschen sowie Fehldetektionen herausfiltern.

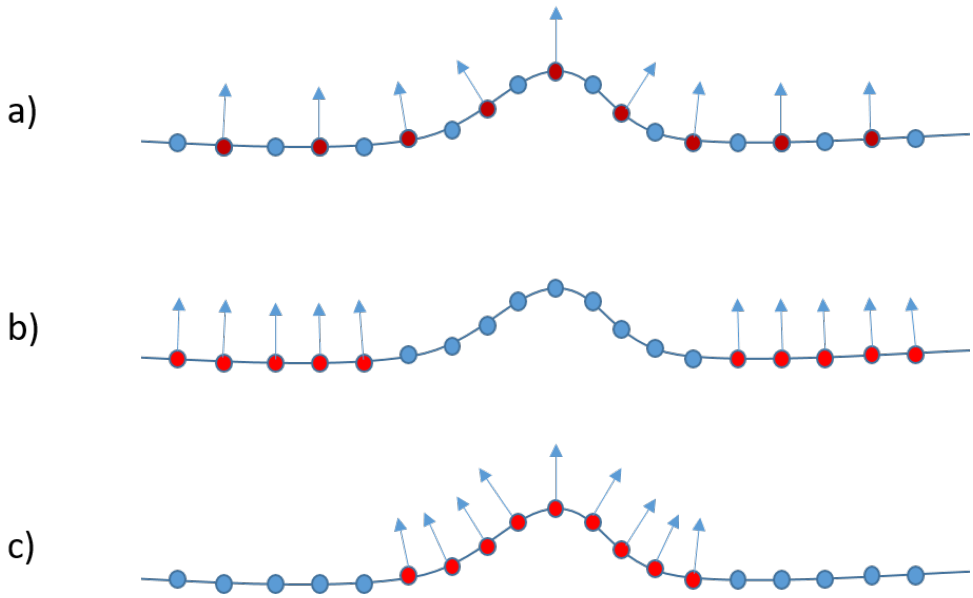


Abbildung 3.8.: Visueller Vergleich von: (a) uniformes bzw. voxelisierendes Sampling, (b) Filtern nach geringen Normalschwankungen und (c) Filtern nach hohen Normalschwankungen [37]

Die Wichtung von Punktepaaren sowie das Herausfiltern von Ausreißern werden zum Teil in einem Schritt getätigt. Bei ICP-Algorithmen werden im Zuge einer Iteration die Distanzen von jedem Punktepaar bestimmt. Diese Informationen können, neben dem Prozess zur Bestimmung der Transformation, genutzt werden, um Punkte die eine gewisse Distanz überschreiten zu ignorieren. So kann die Distanz zwischen beiden Flächenschwerpunkten, siehe Formel (A.3), als Toleranz oder auch Wichtung verwendet werden. Punktepaare die eine größere Strecke zueinander aufweisen, wurden höchstwahrscheinlich falsch zugeordnet und würden das Ergebnis negativ beeinflussen. Der Toleranzwert kann ebenso, anhand vorheriger Transformationen festgelegt werden, ähnlich dem MdkG.

Die Methode, nach der Punkte für den Abgleich ausgesucht werden, wird maßgeblich durch die gewählte Metrik bestimmt. Hier gibt es ebenfalls verschiedene Ansätze, um genauere Ergebnisse zu erzielen. In den folgenden zwei Abschnitten wird auf zwei weitere Variationen des ICP-Algorithmus eingegangen und erklärt, worin diese sich von der PzP-Metrik unterscheiden.

3.2.3. Punkt-zu-Fläche-Metrik

Die bereits erwähnte PzP-Metrik ist generell robust und liefert genaue Ergebnisse, insofern eine initiale Transformation, also eine Schätzung der Rotation und Translation, gegeben ist. Eine Erweiterung dieses Ansatzes stellt die PzF-Metrik dar. Sie berücksichtigt, dass die gescannten Objekte keine einzelnen Punkte sind, sondern Flächen, auf denen die gescannten Punkte liegen. Einhergehend wird nicht mehr angenom-

men, dass die gleichen Punkte, wie im Scan zuvor, aufgezeichnet wurden, sondern nur, dass diese auf denselben Flächen liegen. Der Fehlervektor ist nun nicht mehr der Abstand zwischen zwei Punkten, wie in Formel (A.10), sondern die Distanz entlang des Normalvektors zur Fläche, auf welcher ein Punkt liegt. Es wird also das Skalarprodukt aus dem Abstand zweier Punkte mit dem Normalvektor gebildet. Um den Fehler zwischen zwei Punkten zu eliminieren, reicht es nun, dass beide Punkte auf derselben Fläche liegen [41], [34]. Demzufolge benötigen normalbasierte Metriken deutlich weniger Iterationen, um eine Konvergenz zu erreichen [37].

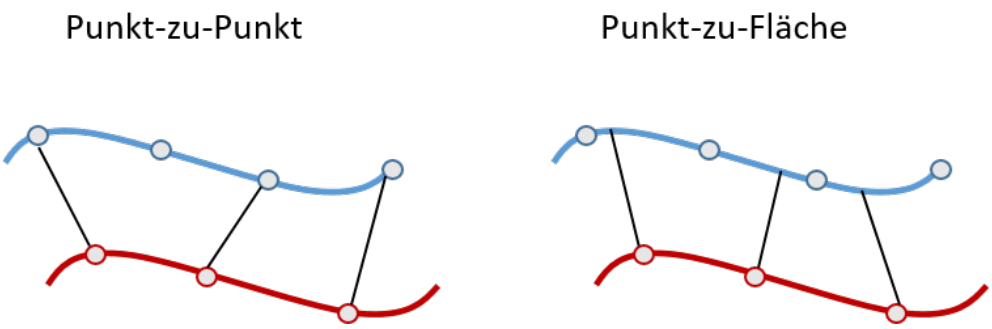


Abbildung 3.9.: Vergleich der Fehlervektoren zwischen PzP-Metrik (links) und PzF-Metrik (rechts) [37]

Abbildung 3.9 zeigt, in Form einer Skizze, wie sich die Fehlervektoren zwischen den beiden Metriken, bei gleichen Punktepaaren unterscheiden. In der darauffolgenden Abbildung 3.10 werden die beiden Varianten am Beispiel eines Punktepaars vergleichend dargestellt. Der Fehlervektor der PzF-Metrik, wird durch die Projizierung entlang der Richtung des Normalvektors n kürzer.

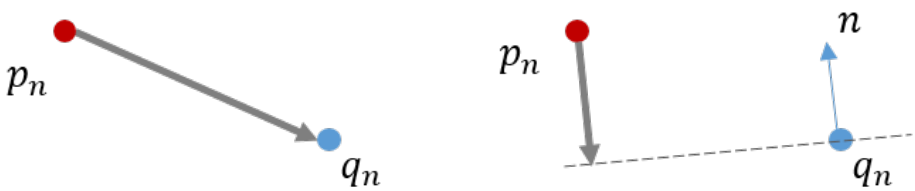


Abbildung 3.10.: Vergleich eines Punktepaars zwischen PzP-Metrik (links) und PzF-Metrik (rechts) [37]

Die Fehler- bzw. Zielfunktion der PzP-Metrik (3.11) wird um den entsprechenden Normalvektor n des aktuellen Punktepaars erweitert und wird so durch die Formel (3.12) definiert.

$$e^2 = \sum_{n=1}^{N} ((q_n - p_n) \cdot n_n)^2 \tag{3.12}$$

Für normalbasierte Metriken werden Optimierungsansätze wie die Methode der kleinsten Quadrate (MKQ) verwendet, da diese die Möglichkeit bieten, komplexere Ziel-

funktionen minimieren zu können [42]. Im Abschnitt A.2 des Anhnangs wird am Beispiel der Zielfunktion (3.12) gezeigt, wie eine Optimierung eines solchen nichtlinearen Problems durchgeführt werden kann. Es wird dafür das Gauß-Newton-Verfahren verwendet. Der Rechenweg orientiert sich an Igor Bogoslavskyis Dokumentation zur PzF-Metrik [43] sowie an den Ausführungen von Janusz Będkowski [26].

3.2.4. Generalisierter ICP-Algorithmus

Generalized Iterative Closest Point (GICP) bzw. generalisierter-iterativ-nächster-Punkt wurde im Jahr 2009 von Segal, Haehnel und Thrun eingeführt [41]. Die Metrik kombiniert die bis dahin bereits bekannten Methoden der PzP- und PzF-Metriken, zu einer generalisierten Variante von ICP. Zusätzlich führt GICP die sog. Fläche-zu-Fläche (FzF)-Metrik ein. Es werden die grundlegenden Methoden des ICP-Algorithmus kombiniert und dadurch die Vorteile der einzelnen Varianten vereint. Die Methode lasse sich laut Cyrill Stachniss als erster Ansatz mit beliebiger LiDAR-Sensorik verwenden und führe dabei meist zu ausreichend guten Ergebnissen. Durch die vielseitige Einsetzbarkeit habe sich der Algorithmus weitgehend durchgesetzt und sei mittlerweile Stand der Technik. GICP berechnet lokale Kovarianzmatrizen, welche die gescannten Flächen der Umgebung statistisch modellieren. Dies wird gleichermaßen für beide Punktwolken getan, wodurch der Scan-Abgleich symmetrisch wird. Die PzF-Metrik hingegen verwendet die Normalvektoren von nur einer Punktwolke und ist dementsprechend nicht symmetrisch. Jene Kovarianzmatrizen werden in die Zielfunktion einbezogen und beeinflussen damit die Form der Fehlerfunktion. So kann in bestimmten Grenzfällen die Zielfunktion der PzP- oder die der PzF-Metrik modelliert werden. [42], [41]

GICP fügt den bisherigen Metriken des ICP-Algorithmus die Nutzung von Wahrscheinlichkeitsverteilungen hinzu. Während GICP die Simplizität und Geschwindigkeit des einfachen ICP-Algorithmus erhalten bleiben würden, erlauben Wahrscheinlichkeitsverteilungen, dass der Zielfunktion weitere Bedingungen hinzugefügt werden können. Im Gegensatz zu den vorherigen Metriken verbessere sich dadurch die Genauigkeit sowie die Robustheit gegenüber Ausreißer-Punkten und inkorrekt assoziierten Punktepaaren. Der Ansatz liegt folglich zwischen konventionellen ICP-Methoden und vollständig wahrscheinlichkeitsbasierten Modellen. Es werden also weiterhin Korrespondenzen zwischen Punkten, in Abhängigkeit von ihrem Abstand zueinander, hergestellt, weshalb GICP trotzdem zur Kategorie der ICP-Algorithmen gehört. [41]

Die Punktwolken werden als lokal planar betrachtet, da die meisten Umgebungen aus stückweisen glatten Flächen bestehen. Durch das Hinzufügen der Wahrscheinlichkeitsverteilungen, wird es ermöglicht auf strukturelle Informationen beider Punktwolken zuzugreifen. Als Folge wird die besagte Symmetrie eingeführt, welche die Genauigkeit

verbessert und Abhängigkeiten von Parametern minimiert. Der restliche Ablauf des Algorithmus bleibt im Vergleich zur PzF-Metrik unverändert. Resultierend wird laut den Autoren eine gleich schnelle Rechengeschwindigkeit beibehalten. [41]

Abbildung 3.11 bietet eine Darstellung des Algorithmus am Beispiel einer Extremsituation. Alle Punkte entlang des vertikalen Abschnittes des grünen Scans sind inkorrekt mit einem Punkt aus dem roten Scan assoziiert worden. Die Paarungen werden aufgrund dessen, dass die Kovarianzmatrizen konträr zueinander sind weniger gewichtet. In diesem Beispiel werden die summierten Matrizen von jeder dieser Paarungen kugelförmig. Was bedeutet, dass sie eine gleichgroße Verteilung in jede Richtung (x, y und z) vorgeben. Sie tragen somit deutlich weniger zur Zielfunktion bei, als die, welche dünne, scharfe Kovarianz-Matrizen formen. Man kann alternativ sagen, dass diese Paarungen eine sog. „soft constraint“ oder „weiche Bedingung“ bilden. Die inkorrekten Paarungen erlauben den Punkten des roten Scans entlang der x-Achse und denen des grünen Scans entlang der y-Achse bewegt werden können. Resultierend formen diese Paarungen nur sehr schwache und wenig informative Bedingungen für die Gesamtausrichtung der Punktwolken, was zu einem genaueren Ergebnis führt. [41]

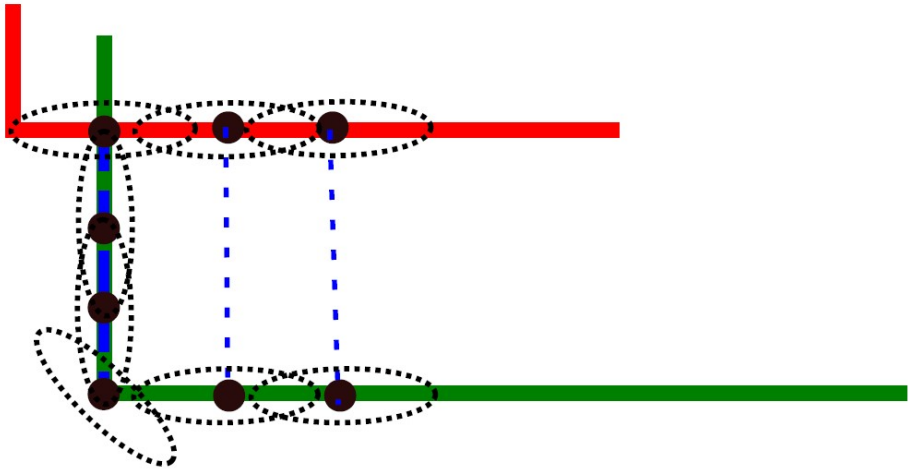


Abbildung 3.11.: Grafische Darstellung von Punktepaarungen (blaue Linien) und Kovarianzstrukturen der einzelnen Punkte (schwarze Linien) [41]

Im Abschnitt A.3 des Anhangs wird die mathematische Herleitung des Algorithmus anhand der Veröffentlichung von Segal, Haehnel und Thrun [41] genauer erläutert.

3.3. NDT - Normalverteilungs-Transformation

Neben den vorher betrachteten Varianten der ICP-Registrierung, existiert die Methode der Normal-Distributions-Transform (NDT) von Biber und Straßer, beschrieben für zweidimensionale Punktwolken [44]. „Normal-Distributions-Transform“ bedeutet ins Deutsche übersetzt so viel wie „Normalverteilungs-Transformation“. Dem Namen entsprechend, arbeitet dieser Algorithmus mit Wahrscheinlichkeitsverteilungen. Folglich verzichten NDT-Algorithmen vollständig darauf, Punktepaare zu finden und diese für die Abstandsminimierung zu verwenden. Das Ergebnis ist ein Algorithmus, welcher ohne einzelne Punktassoziationen die günstigste Transformation zwischen zwei Punktwolken findet, indem die Wolken so ausgerichtet werden, dass ihre Wahrscheinlichkeitsdichten sich am besten ergänzen. Der große Vorteil gegenüber ICP besteht darin, dass die Suche nach den jeweils nächstgelegenen Punkten nicht erforderlich ist. [44]

NDT transformiert diskrete Punktwolken in eine stückweise stetige und differenzierbare Wahrscheinlichkeitsdichtefunktion. Diese Wahrscheinlichkeitsdichte besteht dabei aus mehreren Normalverteilungen. Dafür wird der Raum um den Sensor herum in Zellen gleicher Größe aufgeteilt, dieser Prozess wird auch als „Voxelization“ bzw. Voxelisierung bezeichnet und ähnelt dem im Vorraus erwähnten Sampling-Prinzip [27]. Der beschriebene Algorithmus von Biber und Strasser verwendet dabei eine Zellengröße von 1 m $\times$ 1 m. Anschließend werden für jede Zelle, die mindestens drei gemessene Punkte beinhaltet, die folgenden Schritte absolviert:

1) alle Punkte $q_{i=1\ldots n}$ innerhalb der Zelle werden in einer Matrix gesammelt

2) für die gesammelten Punkte wird der Durchschnitt $\bar{q}_i$ bestimmt:

$$\bar{q}_i = \sum_i q_i \tag{3.13}$$

3) anschließend wird eine Kovarianz-Matrix K berechnet:

$$K_i = \frac{1}{n} \sum_i (q_i - \bar{q}_i)(q_i - \bar{q}_i)^T \tag{3.14}$$

Die Wahrscheinlichkeit, dass an einer beliebigen Stelle x innerhalb dieser Zelle ein Punkt gemessen wird, ist dann abhängig von der Normalverteilung $\mathcal{N}(\bar{q}, K)$:

$$p(x) \sim \exp\left(-\frac{(x-\bar{q})^T K^{-1} (x-\bar{q})}{2}\right) \tag{3.15}$$

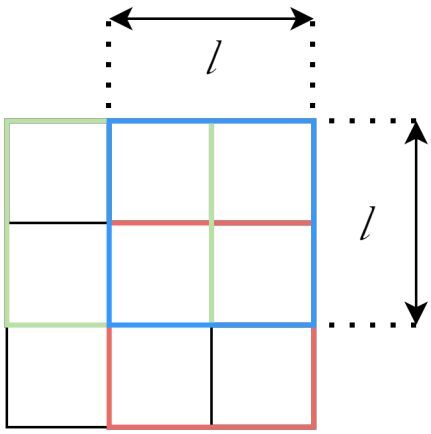


Abbildung 3.12.: Schematische Darstellung der Netzverteilung: Die weiße Zelle stellt das ursprüngliche Netz dar, die farblich markierten Zellen visualisieren die mit Versatz eingefügten Netze

Um den Diskretisierungseffekt minimal halten zu können, wird das erste Netz durch drei zusätzliche Netze erweitert. Sei die Seitenlänge einer quadratförmigen Zelle eines solchen Netzes l, so werden die weiteren Netze mit gleicher Zellengröße um eine halbe Zellenlänge $\frac{l}{2}$ versetzt. Aufgrund dessen liegt jeder gemessene Punkt in vier separaten Zellen und ist so Teil von vier unterschiedlichen Normalverteilungen. Eine Visualisierung dieser Verteilung wird in Abbildung 3.12 gezeigt. In der Grafik sind je Netz nur eine Zelle abgebildet worden, um den Effekt klar visualisieren zu können. Infolgedessen bildet nur das mittlere Quadrat eine Fläche, in der alle gemessenen Punkte Teil jeder Zelle wären.

Die Aufteilung der Zellen wird in den mathematischen Beschreibungen der Autoren nicht weiter berücksichtigt. Es ist wichtig zu wissen, dass die Wahrscheinlichkeiten aus allen Zellen ausgewertet und aufsummiert werden. Alle weiteren Schritte werden mit diesen Wahrscheinlichkeitsverteilungen getätigt.

Für die Berechnung einer Transformation werden wie im Vorhergehenden die Parameter (A.1) und (A.2) gesucht, die die günstigste Transformation aus Formel (A.9) beschreiben. Im Unterschied zu den abstandsbasierten Methoden aus Abschnitt 3.2, wird nun jedoch die Transformation gesucht, welche die aufsummierten Wahrscheinlichkeiten der einzelnen Punkte der bewegten Punktwolke maximieren. Die Unterschritte eines solchen Algorithmus können dabei wie folgt beschrieben werden:

1) Erstellen der NDT der fixierten Punktwolke

2) Transformieren der bewegten Punktwolke mittels einer Initialschätzung aus vorhergehenden Berechnungen

3) Ermitteln der zugehörigen Normalverteilungen der NDT, zu den Punkten der bewegten Wolke

4) Aufsummieren der zugeordneten Wahrscheinlichkeiten von jedem Punkt

5) Berechnung neuer Parameter, indem die Summe der Wahrscheinlichkeiten maximiert wird

6) Ab Schritt 3 wiederholen, bis ein Konvergenzkriterium erfüllt wird

Die zu bestimmenden Parameter lassen sich erneut, wie in der Zielfunktion (A.11), zu einem Vektor zusammenfassen. Da hier ein beliebiger Punkt mit x_i betitelt wird, wird der Vektor der Parameter mit $p = (t_x \quad t_y \quad \Phi)^T$ betitelt. Folglich bezeichnet x_i' den Punkt, welcher der Transformation von Punkt x_i und den Parametern p entspringt: $x_i' = T(x_i, \quad p)$. K_i und $\bar{q}_i$ sind die Kovarianz-Matrix und der Druchschnitt der Normalverteilung der NDT der fixierten Punktwolke, welche dem Punkt x_i' zugeordnet werden. Die Ausrichtung nach p kann als optimal angesehen werden, sobald die Summe aller ausgewerteten Normalverteilungen, aller Punkte x_i' maximal ist. Diese Summe wird $\texttt{score}(p)$ genannt. [44]

$$\texttt{score}(p) = \sum_i \exp\left(\frac{-(x_i' - \bar{q}_i)^T K_i^{-1} (x_i' - \bar{q}_i)}{2}\right) \tag{3.16}$$

$\texttt{score}(p)$ kann schließlich über einen MKQ-Ansatz, wie dem Gauß-Newton-Verfahren, optimiert werden. Da dieses bereits für die PzF-Metrik erläutert wurde (siehe Anhang A.2), wird auf ein weiteres Beispiel verzichtet.

4. Rahmenbedingungen und Vorbetrachtungen

Nach Betrachtung der technischen Grundlagen sowie der vorgestellten Algorithmen und mathematischen Ansätze werden im folgenden Kapitel die Rahmenbedingungen, unter denen das eigene Konzept entwickelt wurde, vorgestellt. Diese entspringen den durch die Aufgabenstellung gesetzten Zielen sowie den Erkenntnissen, welche dem Stand der Technik entnommen wurden.

4.1. Matlab als Softwareumgebung

Als Softwareumgebung für Aufnahme, Vorverarbeitung und Auswertung der Messwerte wurde Matlab von Mathworks gewählt. Matlab ist ein Softwareprodukt des Herstellers Mathworks, welches eine eigene Programmiersprache nutzt und diese in einer Desktopanwendung vereint. Matlab wurde darauf spezialisert eine Umgebung zu schaffen, in der vor allem matrizenbasierte Mathematik leicht verständlich formuliert werden kann [45].

Matlab bietet im Vergleich zu anderen Softwareanwendungen und Programmiersprachen den Vorteil, dass es für viele Anwendungen bereits vorgefertigte „Toolboxen“ anbietet. Diese Toolboxen können als zusätzliche Softwarepakete verstanden werden, welche professionell entwickelte Funktionen für verschiedene technische Anwendungen beinhalten [45]. So bietet Mathworks eine Toolbox für das Arbeiten mit LiDAR-Sensoren an, welche bereits grundlegende Funktionen beinhaltet. Dazu gehören Funktionen zum Einlesen, Darstellen oder auch zum Manipulieren von Punktwolken [46]. Für ein erleichtertes Arbeiten mit LiDAR-Daten existiert in Matlab eine separate Objektklasse für Punktwolken. Mit dem Befehl `pointCloud(xyz)` kann ein solches Punktwolkenobjekt aus den in der Matrix `xyz` gegebenen Koordinaten erstellt werden [47]. Variablen werden innerhalb von Matlab in einem separaten Arbeitsspeicher abgelegt, welcher „Workspace“ genannt wird. Alle im Workspace befindlichen Daten werden beim Schließen von Matlab gelöscht, insofern diese nicht als .mat-Datei auf der Festplatte des Systems gespeichert werden. Zusätzlich können im Punktwolken-Objekt Werte für die Intensität, die Normalvektoren oder die Farben der einzelnen Punkte hinterlegt werden. Der Objektklasse werden eine Reihe von Objektfunktionen zugeordnet, welche das Arbeiten mit Punktwolken erleichtern. Dazu gehören beispielsweise Funktionen, um Punkte aus einem geometrisch definierten Bereich zu selektieren, wie der Befehl `findPointsInROI`, welcher Punkte innerhalb eines Quaders auswählt und deren Indizes in einer Variable speichert [48].

Zusätzlich existieren innerhalb der LiDAR-Toolbox, vorgefertigte Registrierungsalgorithmen für dreidimensionale Punktwolken. Neben weiteren in dieser Arbeit nicht behandelten Algorithmen, gibt es Funktionen für die in Kapitel 3.2 und 3.3 vorgestell-

ten Algorithmen [49], [50]. Diese erzeugen ein `rigidtform3d`-Objekt, welches die errechnete Transformation beinhaltet [40]. Mithilfe einer weiteren Funktion können Punktwolken, unter Eingabe eines solchen `rigidtform3d`-Objekts, entsprechend dieser transformiert werden [51].

Neben den genannten Funktionen bietet Matlab diverse Möglichkeiten, um Punktwolken visuell darstellen zu können. Auch hier existieren bereits vorgefertigte Funktionen, wie zum Beispiel die Vergleichsvisualisierung zweier Punktwolken via der Funktion `pcshowpair` [52]. Der „Lidar Viewer“ schafft eine Umgebung zur Darstellung von Punktwolken, in der Manipulationen direkt visualisiert werden können, ohne dass nach jedem Schritt ein neuer Plot angefertigt werden muss [53].

4.2. Erfassen der Messwerte

Aufgrund dessen, dass verschiedene Sensoren zum Einsatz kamen, welche unterschiedliche Dateiformate erzeugen, mussten separate Wege gefunden werden, um diese in ähnlicher Form für Matlab zugänglich zu machen. Da für die verfügbaren Sensoren bereits Skripts zur Aufzeichnung bzw. Konvertierung in ein Format, welches mit Matlab kompatibel ist, aus vorherigen Diplomarbeiten vorlagen, wurde darauf verzichtet neue Skripts zu programmieren.

Das Ziel dieser Arbeit ist die Untersuchung der Genauigkeit eines solchen Konzeptes, in Hinsicht auf Sensorik und Software. Deshalb wurde auf ein in Echtzeit arbeitendes Konzept verzichtet. Echtzeit bedeutet hier, dass der gesamte Prozess von Datenerfassung bis Kartierung, während der Messwertaufnahme selbst erfolgt und innerhalb dieser abgeschlossen wird. Gründe dafür sind unter anderem, dass der Prozess der Kartierung nicht parallel stattfinden muss, um dessen Genauigkeit auswerten zu können, sowie die teils aufwendige Konvertierung der Messwerte und die Schwierigkeit in Matlab parallele Rechenprozesse auszuführen.

4.2.1. Einbindung LIVOX Horizon

Der Horizon von Livox arbeitet mit dem vom Hersteller selbst entwickelten .lvx-Dateiformat [54]. Mit dem „Livox Viewer“ bietet der Hersteller zudem eine kostenlose Software, mit welcher die Sensoren kalibriert, Messungen aufgezeichnet, wiedergegeben und in andere gängige Punktwolkenformate formatiert werden können [55]. Abbildung 4.1 zeigt einen Bildschirmausschnitt der Software, in welcher eine bereits getätigte Messung visualisiert wird. In der oberen Menüzeile befinden sich Werkzeuge zur Wiedergabe, Aufzeichnung und Auswertung von Punktwolken.

Zum Zeitpunkt des Verfassens dieser Arbeit exisitert für Matlab keine explizite Einbindung des Livox Horizon oder des .lvx-Dateiformats. Es wurde darauf verzichtet ein separates Skript für die Aufzeichung innerhalb von Matlab zu entwickeln, anstelle dessen wurde die Entscheidung getroffen, Messwerte über den Livox Viewer aufzuzeichnen und diese mittels Matlab in ein auslesbares Punktwolkenformat zu konvertieren. Dies konnte nicht mit dem Livox Viewer durchgeführt werden, da dieser nur statische Messungen konvertieren kann, was darin resultiert, dass alle aufgezeichneten Punkte in einer großen Punktwolke fusioniert werden. Die Weiterverarbeitung für mobile Anwendungen wäre damit nicht möglich.

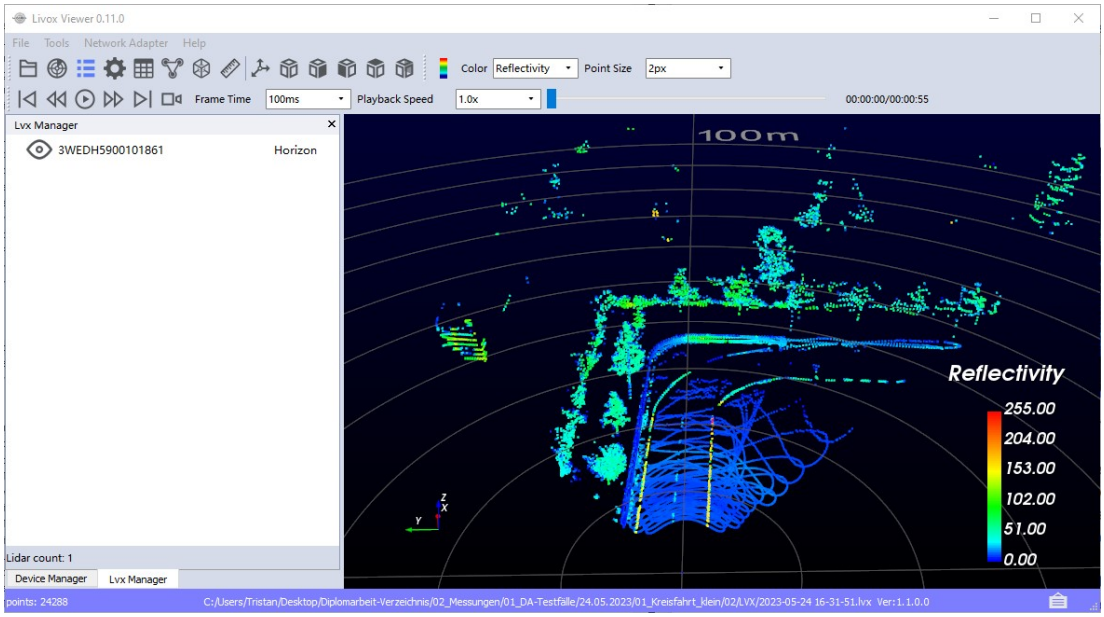


Abbildung 4.1.: Bildschirmausschnitt der Benutzeroberfläche des „Livox Viewer“

Dipl.-Ing. Blechschmidt entwickelte im Rahmen seiner Diplomarbeit ein Skript, welches unter Nutzung der offengelegten Struktur des .lvx-Formats die Punktwolkeninformationen innerhalb von Matlab auslesen und als Punktwolkenobjekt abspeichern kann. Dabei werden die voneinander getrennten Frames beibehalten und als separate Punktwolke deklariert [56]. Für den praktischen Teil dieser Arbeit wurde auf dieses Skript zurückgegriffen. In Zusammenarbeit mit Dipl.-Ing. Blechschmidt wurde das Skript erweitert, sodass neben den Punktwolken auch die Daten der verbauten IMU mit ausgelesen werden können. Mithilfe des Skripts werden zwei sog. `Timetable`-Objekte erstellt. Dieser Dateityp ist ähnlich einer üblichen Tabelle, mit dem Zusatz, dass jeder Zeile ein Zeitstempel zugeordnet wird [57]. Dabei werden in der einen Tabelle die Punktwolken abgelegt und in der anderen die Messwerte der IMU, mit den zugehörigen Zeitstempeln. Aufgrund dessen, dass die entwickelten Algorithmen keine zeitsynchronisierten Sensoren benötigen, wird die bis dato absolute Messzeit ab Einschalten des Horizon notiert. So kann nachvollzogen werden, in welcher Frequenz die Messwerte erzeugt wurden, bzw. in welchem zeitlichen Abstand diese zueinander

liegen. Die Skripte zur Konvertierung der .lvx-Dateien können dem Datenträger im Anhang entnommen werden (siehe Anhang C).

Während der Inbetriebnahme fiel auf, dass der Horizon Punktwolkenpakete im zeiltichen Abstand von 0,05 s aussendet, also mit einer Frequenz von 20 Hz. Das Konvertierungsskript definiert diese Pakete als vollständigen Frame. Ein tatsächlicher Frame besteht jedoch aus mindestens zwei dieser aufeinanderfolgenden Pakete. Der Livox Viewer hingegen definiert Punktwolken über die sog. „Frame Time“, anstelle fester Punktwolken, kann über die Menüzeile festgelegt werden, wie lang die Messdauer eines Frames sein soll. Anhand dieser Frame Time fügt der Livox Viewer die einzelnen Pakete zu einer Punktwolke zusammen. Definiert man diese Pakete als alleinstehende Frames, wird das in Abbildung 2.7 dargestellte Scanningmuster in der Mitte geteilt. Abbildung 4.2 zeigt zwei aufeinanderfolgende Pakete. Das erste Paket wird in blau und das zweite in rot dargestellt.

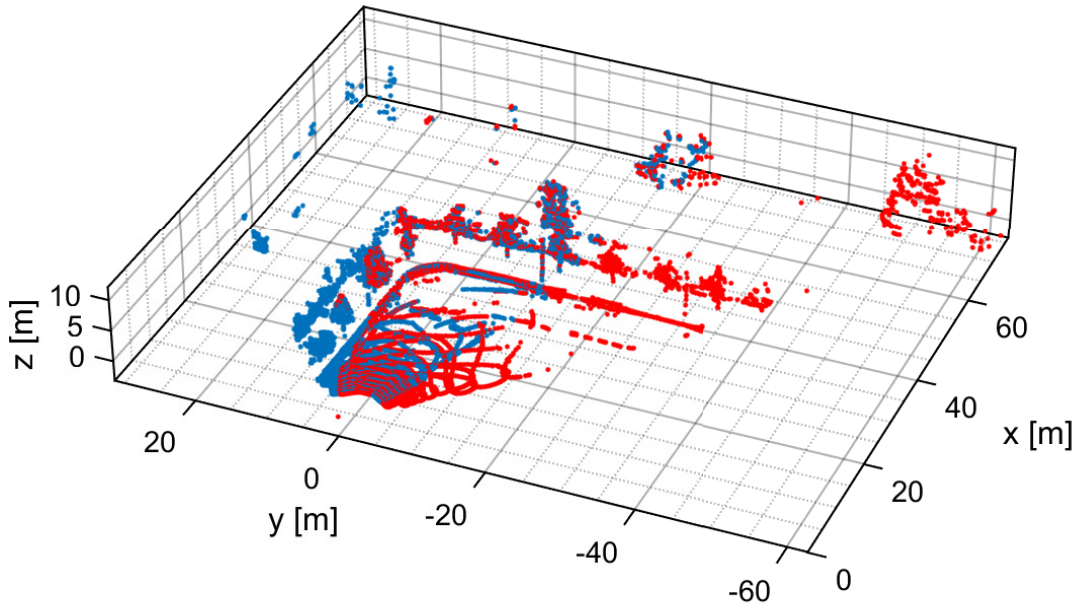


Abbildung 4.2.: Visualisierung zwei aufeinanderfolgender Pakete des Livox Horizon

Es ist zu erkennen, dass beide Punktwolken einen überlappenden Bereich in der Nähe der x-Achse aufweisen. Die darauffolgende dritte Punktwolke deckt wieder den Bereich des ersten, blau markierten, Pakets ab. Es wird angenommen, dass dieses wiederkehrende Muster dem Scanningverfahren entspringt. Für den Registrierungsprozess wurde festgelegt, dass eine Punktwolke aus zwei aufeinanderfolgenden Paketen besteht. Anderenfalls würde man die Punktdichte des Sensors halbieren. Zudem würde der Registrierungsprozess stark eingeschränkt werden, wenn die Punktwolken nicht den gleichen Bereich abdecken.

In Tabelle 4.1 werden die wichtigsten Eigenschaften einer Punktwolke des Livox Horizon aufgelistet. Eine Punktwolke oder auch Frame, besteht aus zwei aufeinanderfolgenden Paketen. Die Punktanzahl pro Frame wurde ermittelt, indem über eine Testmessung hinweg der Durchschnitt berechnet wurde. Dabei wurden vorher jeweils zwei Pakete zusammengefügt und alle Nulleinträge entfernt.

Livox Horizon	
Punktwolkenstruktur	unorganisiert M x 3
Anzahl Punkte pro Frame (gefiltert)	$\approx$ 19.317
Aufnahmedauer pro Frame	0,1 s (10 Hz)
FoV	81,7° x 25,1°

Tabelle 4.1.: Übersicht der Eigenschaften einer Punktwolke des Livox Horizon

4.2.2. Einbindung Ouster OS1

Im Gegensatz zum Horizon, werden für ausgewählte Sensoren von Ouster Kompatibilitätspakete von Matlab angeboten [58]. Mit den Softwarepaketen können direkt innerhalb von Matlab Messungen aufgenommen und ausgelesen werden, solange der Sensor mit dem Netzwerk des Computers verbunden ist. Durch die Zusatzsoftware kann mit dem Befehl `ousterlidar`, ein Sensorobjekt erstellt werden. Im Objekt werden die separierten LiDAR-Frames einer Aufzeichnung gespeichert. Messungen können über die Befehle `start` und `stop` gestartet bzw. gestoppt werden. Mit dem zusätzlichen Befehl `read` können einzelne oder alle `pointCloud`-Objekte ausgelesen und als isolierte Variable im Workspace hinterlegt werden.

Um mit dem Ouster OS1 Aufzeichnungen tätigen zu können, muss der Sensor ggf. zuvor über die Hersteller-Software „Ouster Studio“ eingerichtet werden [59]. Ouster Studio ist, ähnlich dem Livox Viewer, ein Softwaretool zur Einrichtung von Ouster Sensorik, Aufnahme, Visualisierung und Verarbeitung von Punktwolken. Das Aufnehmen von Messwerten über Matlab hat sich insofern als nützlich erwiesen, da das Konvertieren der Messungen in `pointCloud`-Objekte entfällt und so Zeit bzw. Aufwand gespart werden kann. Dementsprechend wurde die Entscheidung getroffen, nicht die Hersteller-Software zu verwenden, sondern Skripts in Matlab zu implementieren. Für den Sensor von Ouster lag bereits ein Skript von vorherigen Projekten vor, welches situationsbedingt angepasst wurde. Das Skript ist ebenfalls dem Datenträger zu entnehmen.

Ouster OS1	
Punktwolkenstruktur	organisiert M x N x 3
Vertikale Auflösung	64 Ebenen
Horizontale Auflösung	1024 Punkte pro Ebene
Anzahl Punkte pro Frame (ungefiltert)	65.536
Aufnahmedauer pro Frame	0,1 s (10 Hz)
FoV	360° x 45°

Tabelle 4.2.: Übersicht der Eigenschaften einer Punktwolke des Ouster OS1

In Tabelle 4.2 sind die wichtigsten Eigenschaften einer Puntkwolke des Ouster OS1 aufgelistet. Im Vergleich zum Horizon besitzen Punktwolken des Ouster eine andere Struktur. Es handelt sich dabei um sog. organisierte Punktwolken, welche nach Anzahl der Ebenen M, Punkte pro Ebene N und den drei Dimensionsrichtungen x, y und z sortiert werden. Innerhalb von Matlab bestehen solche Punktwolken aus drei Matrizen (eine pro Dimension), mit jeweils N Spalten, welche wiederum jeweils M Zeilen beinhalten [60]. Die Struktur der Punktwolken hängt mit der Art des LiDAR-Sensors zusammen. Der Livox Horizon kann aufgrund dessen, dass er keine traditionelle Ebenenstruktur besitzt, keine Punktwolken dieser Art produzieren. Einige Funktionen, wie das Entfernen des Bodens aus einer Punktwolke, setzen jedoch eine solche Struktur voraus.

Mit 65.536 Punkten pro Frame zeichnet der Sensor von Ouster mehr als dreimal so viele Punkte auf wie der Horizon. Infolgedessen beanspruchen die Messdateien deutlich mehr Speicherplatz. Während der Messfahrten fiel auf, dass Matlab ab einer Dateigröße von $>$1 GB Schwierigkeiten bekommt, die Punktwolken abspeichern zu können. Was daher kommt, dass Matlab die Messwerte im Arbeitsspeicher sammelt und erst nach Stoppen der Messung im Speicher ablegt. Dementsprechend mussten kürzere maximale Messzeiten in Kauf genommen werden.

4.3. Versuchsaufbau

Im folgenden Abschnitt sollen die allgemeinen Rahmenbedingungen, unter denen die Versuche durchgeführt wurden, ausführlicher erläutert werden. Dazu zählen der Aufbau der Messeinrichtungen, die Versuchsumgebung sowie die definierten Testfälle.

4.3.1. Versuchsträger

Für die Durchführung von Messfahrten wurde ein Fahrzeug benötigt, das mit der Messtechnik ausgestattet werden konnte. Diesen Zweck erfüllte ein BMW I3, welcher im Labor als allgemeines Testfahrzeug genutzt wird. Die Art und Ausstattung des genutzten Fahrzeugs spielen hierbei jedoch keine Rolle, da alle Messwerte über externe Sensorik aufgenommen werden. Unterschiede in den Messwerten würden nur aufgrund von anderen Sensorpostionen auftreten. Abseits davon wären die aufgezeichneten LiDAR- Scans identisch, insofern eine gleiche Sichtfreiheit bereitgestellt werden kann. Damit das vollständige FoV der Sensoren genutzt werden kann, insbesondere derer die 360° abdecken, wurden die Sensoren auf dem Fahrzeugdach befestigt, wie in Abbildung 4.3 zu erkennen ist.

Abbildung 4.3.: Anbringung der Sensorik am Testfahrzeug

Auf dem Dach des I3 wurde bereits zuvor eine Plattform befestigt, auf welcher Sensorik montiert werden kann. Die Aluminuimprofile der Befestigungsvorrichtung stellten sich als geeignete Montagevorrichtung heraus. Die Sensoren wurden dann entsprechend ihrer Befestigungsmöglichkeiten, auf der vorderen, zur Längsachse querstehenden Schiene verschraubt. Dabei wurde darauf geachtet, dass sie so nah wie möglich an der Mittelebene des Fahrzeugs liegen und dabei nicht interferieren. Der Ouster wurde über ein zusätzliches Schienenstück und einer gedruckten Halterung erhöht, sodass der Horizon nicht im sichtbaren Bereich liegt. Die Postionen beider Scanner wurde über Markierungen auf der Schiene festgehalten, um bei Abnahme und erneuter Befestigung eine möglichst ähnliche Konfiguration reproduzieren zu können. Der Abstand zwischen den Mittelpunkten beider Sensoren liegt bei 17 cm.

Abbildung 4.4 zeigt anhand eines Struktogramms wie die Messtechnik innerhalb des Fahrzeugs eingerichtet wurde. Beide Sensoren verfügen über ein eigenes Hub, welches die Scanner mit den nötigen Schnittstellen für Stromversorgung und Datenübertragung ausstattet. Für den Betrieb der Sensoren wurde eine externe Spannungsquelle im Kofferraum montiert. Über einen Netzwerkverteiler werden die beiden Sensoren mit einem Messrechner verbunden, welcher genutzt wird um die Aufzeichnungen durchzuführen. Durch die Nutzung eines Netzwerkverteilers können zeitgleich mehrere Sensoren angesteuert werden. Infolge dessen ist es möglich, eine Messfahrt aus der Sicht beider Sensoren vergleichen zu können.

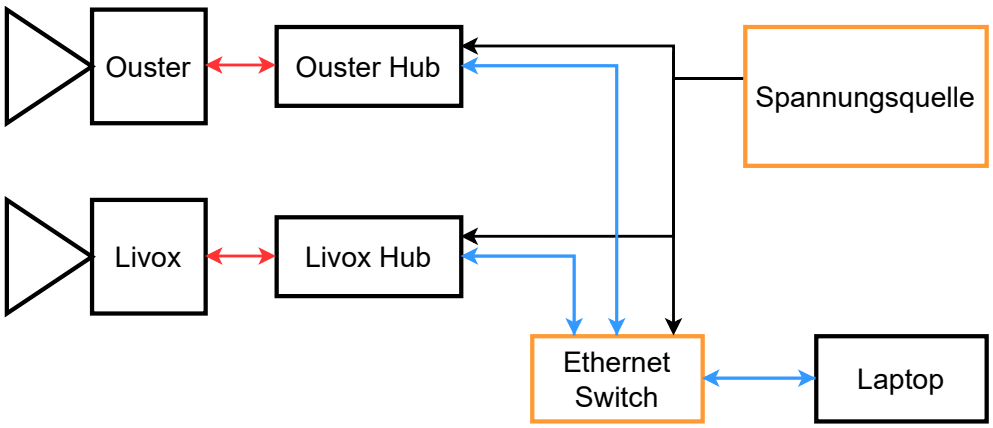


Abbildung 4.4.: Aufbau und Vernetzung der Messtechnik im Fahrzeug

4.3.2. Versuchsumgebung

Für die Durchführung der Versuche wurde eine geeignete Umgebung benötigt. Diese Umgebung sollte eine Reihe von Anforderungen erfüllen, die erforderlich sind, um aussagekräftige Messwerte sammeln zu können. Einerseits wird ein störfreies Umfeld benötigt, in dem Testszenarien klar definierbar und ausführbar sind, zudem darf jedoch die Vergleichbarkeit zu echten Fahrsituation nicht verloren gehen. Daraus entspringen einige Grenzen, denen die entwickelten Konzepte unterliegen, welche im nächsten Kapitel genauer erläutert werden.

Das K-Gebäude der HTW Dresden bietet dafür eine eigene Testfläche, abseits des öffentlichen Straßenverkehrs, siehe Abbildung 4.5. Ein daraus entspringender Vorteil ist die Abwesenheit von anderen bewegten Objekten, was den Kartierungsprozess erleichtert. Die Testfläche ist außerdem mit Fahrstreifen- und Fahrbahnbegrenzungen versehen. Diese wurden genutzt, um reproduzierbare Routen definieren und abfahren zu können. Des Weiteren sind die Markierungen durch ihr hohes Maß an Retroreflexion, für die LiDAR-Sensoren leicht zu erfassen, wie schon in Abbildung 4.1 zu erkennen ist.

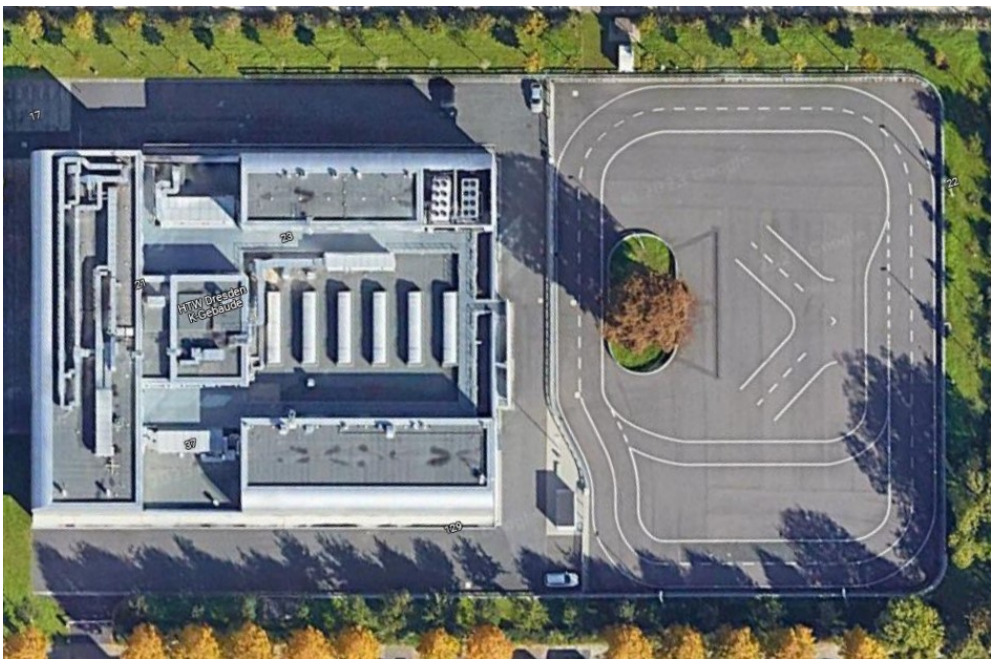


Abbildung 4.5.: Das K-Gebäude der HTW Dresden und die anliegende Testfläche [61]

Außerdem wurden in der Vergangenheit einzelne Punkte der Testfläche mittels GPS ausgemessen und in OpenStreetMap (OSM) hinterlegt. So konnte eine digitale Nachbildung der Testfläche samt Markierungen mit den genauen Abständen erstellt werden. Dafür wurden die GPS-Koordinaten der einzelnen Punkte, von OSM heruntergeladen [62]. Zur weiteren Verarbeitung wurden die OSM-Daten mit dem Java OpenStreetMap Editor (JOSM) [63] geöffnet und in .gpx-Dateien konvertiert. Dabei wurden alle Punkte entfernt, die nicht zum Testgelände gehören, sodass nur noch die Begrenzungen der Fläche sowie die Spurmarkierungen und -begrenzungen bestehen blieben.

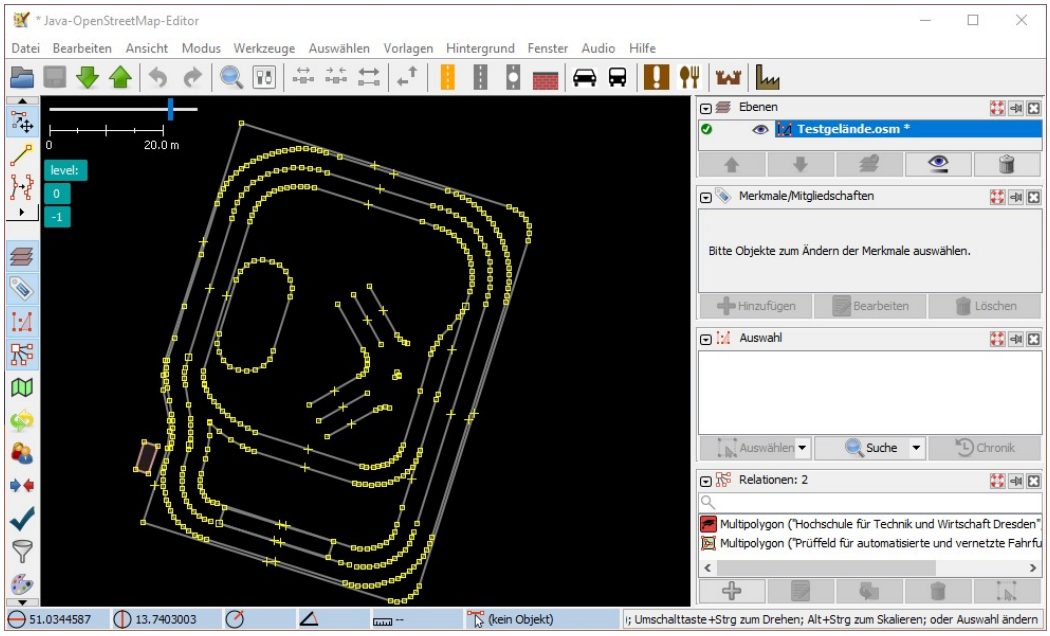


Abbildung 4.6.: GPS-Punkte der Testfläche visualisiert in JOSM

Mit dem Befehl `gpxread` [64], welcher Teil der Mapping Toolbox ist [65], konnten die einzelnen .gpx-Dateien in Matlab eingelesen werden. Unter Nutzung der Haversine-Formel [66], wurden die Längen- und Breitengrade in das kartesische Koordinatensystem umgerechnet. Dabei entstehen kleine Abweichungen, da die Formel davon ausgeht, dass die Erde eine perfekte Sphäre darstellt. Der Nullpunkt wurde auf die nordwestliche Ecke der Flächenbegrenzung gelegt und die gesamten Punkte so rotiert, dass die obere Begrenzung annähernd parallel zur x-Achse verläuft, wie in Abbildung 4.7 zu erkennen ist. Die dadurch entstandene Karte wurde genutzt, um Abstände innerhalb der Testfläche ausmessen zu können. Des Weiteren dienten die Koordinaten der einzelnen Punkte als Referenz im Auswertungsprozess.

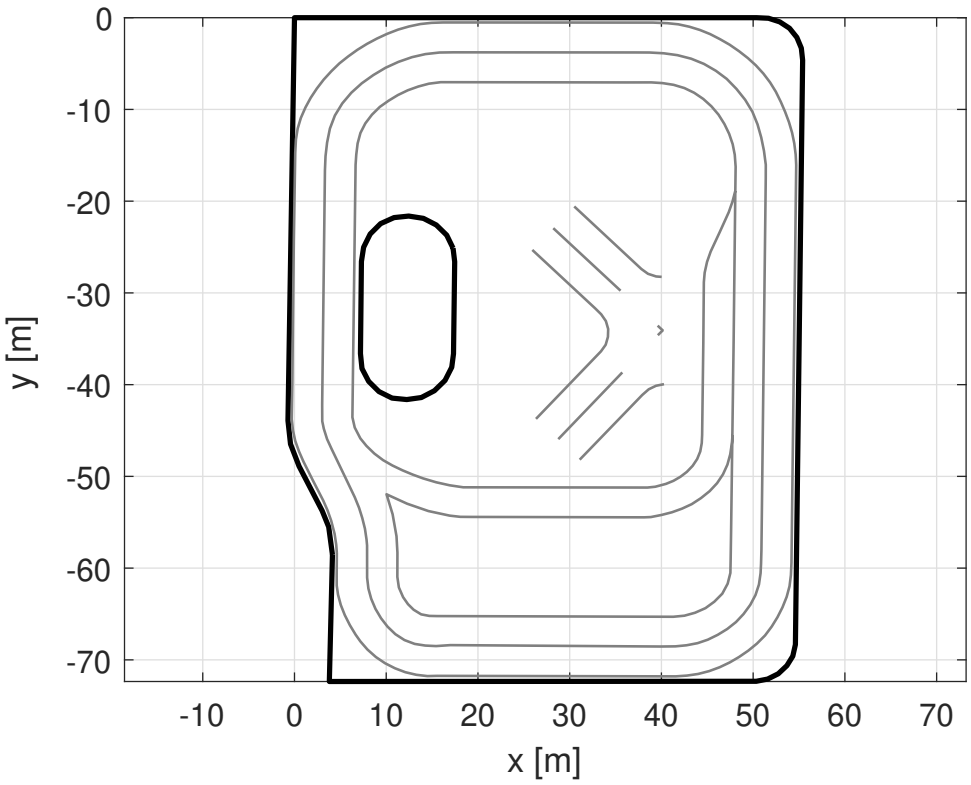


Abbildung 4.7.: Karte der Testfläche als Matlab Figure

4.3.3. Testfälle

Zur Evaluierung der implementierten Algorithmen sowie der Sensorik, mussten spezifische Testfälle definiert werden. Diese wurden so ausgelegt, dass die zur Verfügung stehende Testfläche vollumfänglich genutzt werden kann. In Hinsicht auf Anwendungen im realen Straßenverkehr, wurden die Spurführungen insofern eingebunden, dass sie die Auswahl an möglichen Routen vorgeben. Da in realen Situationen, vor allem in urbanen Gegenden, die Bewegungen des Fahrzeugs größtenteils auf lange Geraden und einheitliche Kurvenradien begrenzt sind, wurden die möglichen Routen nicht als Einschränkung, sondern viel mehr als eine Annäherung an tatsächliche Fahrsituatio-

nen gesehen. Die Größe der Testfläche ist vergleichbar mit der eines mittelgroßen Parkplatzes und könnte dementsprechend das Szenario einer Parkplatzsuche nachbilden. Im Falle des anfänglich erwähnten VP, könne so überprüft werden, ob die implementierten Algorithmen eine hinreichende Genauigkeit bieten, um die Fläche eines vermeintlichen Parkplatzes zu kartieren.

Des Weiteren bietet das Fahren innerhalb der vorgegebenen Routen den Vorteil, dass vor Auswertung der Messwerte bekannt ist, wie die Form der berechneten Trajektorie des Fahrzeugs aussehen muss. Folglich ist es leichter zu erkennen, an welchen Stellen der Berechnung bzw. Messwerterfassung größere Abweichungen auftreten, da die Trajektorie immer innerhalb der bekannten Spurführungen verlaufen muss. Der Rundkurs bietet den weiteren Vorteil, dass innerhalb kurzer Strecken der Startpunkt wieder erreicht werden kann. So können Fehler in der Transformation leichter erkannt werden, als bei Fahrten, in denen der Start- und Endpunkt nicht gleich sind. Das liegt daran, dass ein Versatz in der Kartierung deutlicher zu Erfassen ist, wenn die gleiche Umgebung zweimal kartiert wird, da sich die erfassten Objekte aufgrund des Fehlers doppeln, ähnlich wie in den Abbildungen 3.3 und 3.4.

Aufgrund von Sicherheitsvorkehrungen und der begrenzten Größe des Areals, ist die maximal erlaubte Geschwindigkeit auf 30 km/h begrenzt. Infolgedessen beschränken sich die möglichen Testfälle auf Szenarios mit niedrigeren Geschwindigkeiten. Eine Vergleichbarkeit besteht so nur zu den vorraus erwähnten Parkplatzsituationen sowie zu Teilen des innerstädtischen Verkehrs. here Technologies in Zusammenarbeit mit BVL.digital fanden in einer Studie heraus, dass zu Stoßzeiten an Werktagen die durchschnittliche Geschwindigkeit auf Hauptverkehrsadern in Dresden bei ungefähr 27,3 km/h liegt, in Frankfut am Main sogar nur bei 17,4 km/h [67]. Ausgehend davon könnte in Teilen eine allgemeine Vergleichbarkeit zu Fahrsituationen im Stadtverkehr angenommen werden.

Ein großer Unterschied zu realen Szenarien bleibt bei den gewählten Testfällen jedoch bestehen. Es werden keine anderen bewegten Objekte auf der Testfläche, neben dem Messfahrzeug, implementiert. Das Inkludieren von bewegten Objekten verlangt das Hinzufügen weiterer Ausschlussmechanismen in den Prozess der Punktwolkenregistrierung. Bewegte Objekte müssen getrennt von statischen Objekten betrachtet werden, um eine korrekte Transformation zwischen zwei Punktwolken abschätzen zu können. Da das Detektieren von bewegten Objekten vergleichbar umfangreich zum Kartierungsprozess selbst ist, wurde darauf verzichtet zusätzliche Algorithmen zu entwickeln. Als Konsequenz wurde allein die Genauigkeit der Registrierungsalgorithmen in statischen Umgebungen überprüft. Unter der Annahme, dass eine zukünftig implementierte Objektdetekion den größten Teil der bewegten Objekte vom Prozess der

Kartierung ausschließen könne, würden die gleichen Genauigkeiten vorliegen, wie sie unter den momentanen Bedingungen ermittelt wurden.

Zur Untersuchung der Genauigkeit der verschiedenen Algorithmen und Sensoren wurden die in Abbildung 4.8 skizzierten Fahrmanöver definiert. Diese setzen sich zusammen aus einer kleinen Runde (Testfall 1), welche in blau eingezeichnet wurde, einer großen Runde (Testfall 2), welche in grün eingezeichnet wurde und einer kombinierten Fahrt (Testfall 3), bestehend aus der kleinen Runde und einer weiteren größeren, welche mit der gestrichelten Linie gekennzeichnet wurde.

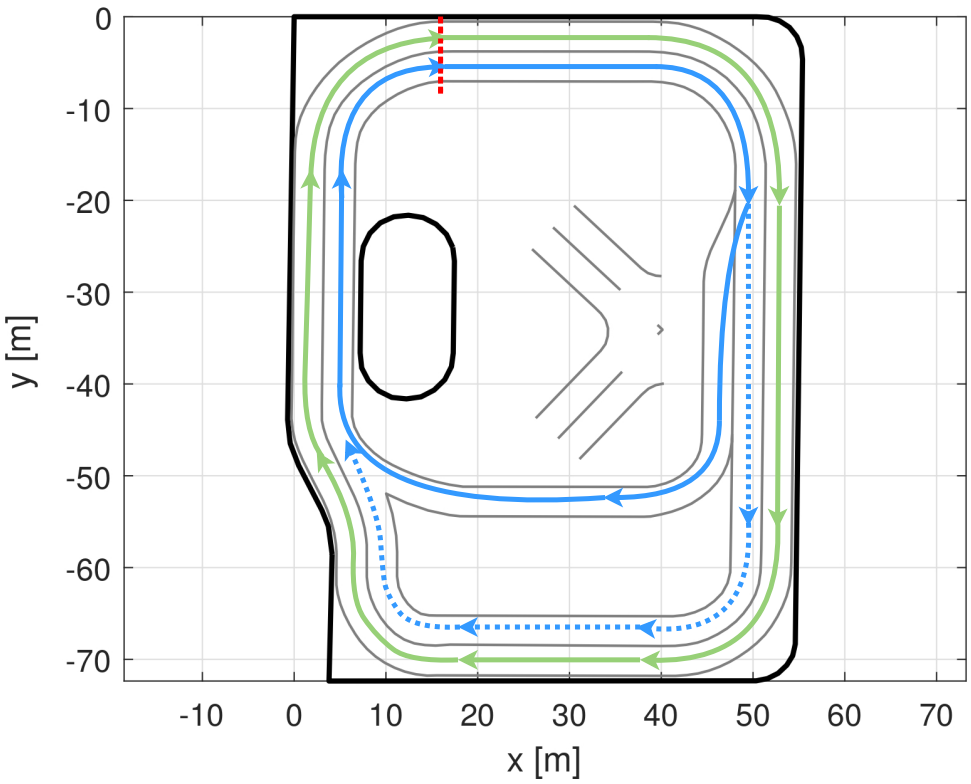


Abbildung 4.8.: Skizzierung der Testfälle auf der Testfläche

Durch die Unterteilung soll überprüft werden, inwiefern die zurückgelegte Strecke einen Einfluss auf das Gesamtergebnis hat. Des Weiteren soll mittels der kombinierten Fahrt untersucht werden, in welchem Ausmaß eine Graphoptimierung durch die Erkennung von bereits besuchten Orten, Einfluss auf das Gesamtergebnis hat. Der Start- und Endpunkt ist für jeden Testfall derselbe und wurde auf die Höhe des letzten GPS-Punkt der inneren Spurbegrenzung, der oberen linken Kurve gelegt. Da die aufgezeichneten GPS-Koordinaten in keiner Weise auf der Testfläche markiert wurden, wurde die Annahme getroffen, dass dieser letzte Punkt der Kurve, an der Stelle gemessen wurde, wo die Spurmarkierung der Kurve endet und die der Geraden beginnt.

Die Fahrmanöver wurden dabei mit Geschwindigkeiten von 10 bis 30 km/h getätigt, abgesehen vom Anfahren und Halten. Es wurde darauf verzichtet mit konstanter

Geschwindigkeit zu fahren, da dies nur funktionieren würde, wenn die Messungen bei aktiver Fahrt gestartet werden würden. Dabei wäre es jedoch nicht möglich, die Messungen vom selben Punkt aus zu starten. Des Weiteren wird durch das Schwanken der Geschwindigket ein realistischeres Szenario geschaffen, da in realen Situationen nicht mit konstanter Geschwindigkeit gefahren wird.

5. Implementierung der Konzepte und Algorithmen

Das folgende Kapitel widmet sich der praktischen Umsetzung der vorab beschriebenen Rahmenbedingungen und Vorbetrachtungen. Es folgt die Vorstellung eines eigenen Konzepts und dessen Vergleich mit zuvor erwähnten Algorithmen. Final wird die praktische Umsetzung in Matlab erläutert. Alle im Folgenden erwähnten Skripts können dem Datenträger des Anhangs entnommen werden (siehe Ahang C).

5.1. Konzeptvergleich

In Abbildung 3.1 des vorherigen Kapitels wird ein allgemeines Konzept von LiDAR-SLAM in Form eines Struktogramms dargestellt. Dieses wurde als allgemeiner Ansatz verwendet. Ausgehend davon folgte die Entwicklung eines eigeneb Konzepts innerhalb von Matlab.

Als zusätzliche Quelle dienten von Mathworks veröffentlichte Beispiele von LiDAR-SLAM. In einem Beitrag von Mathworks [68] wird ein Algorithmus beschrieben, welcher unter der Nutzung der NDT-Methode einen Frame-zu-Frame Abgleich vollzieht und so iterativ die Transformationen zwischen den Punktwolken bestimmt. Zusätzlich wird eine Schleifenschluss-Erkennung, analog der in Kapitel 3.1.2 beschriebenen Methode implementiert. Hierfür verwenden die Autor*innen die bereits erwähnte Funktion von Matlab [32], welche auf den Ausarbeitungen von Giseop Kim u.A. basiert [29]. Neben den LiDAR-Daten, wird zusätzlich auf Messwerte einer IMU zugegriffen, um eine initiale Transformation bestimmen zu können. Diese fließt in den Registrierungsprozess ein und soll dabei helfen eine genauere Ausrichtung zu erzielen. Dieses Beispiel wird zum besseren Verständnis im Weiteren mit „Mathworks B“ betitelt. In einem weiteren Beispiel von Mathworks wird der GICP-Algorithmus im Frame-zu-Frame Abgleich verwendet, um so eine Karte aus den vorhandenen LiDAR-Daten zu generieren. Es werden auch hier auf die Messungen einer IMU sowie zusätzlich auf die eines GPS zurückgegriffen, um den Kartierungsprozess zu optimieren [69]. Im Folgenden wird dieses Beispiel mit „Mathworks A“ betitelt.

Das eigens erstellte Softwarekonzept orientiert sich an den Beispielen von Mathworks sowie an den Algorithmen, die im Kapitel 3.1.1 vorgestellt wurden, insbesondere an KISS-ICP [23]. Die folgende Tabelle 5.1 liefert einen Überblick über die Gemeinsamkeiten und Unterschiede zwischen den praktischen Umsetzungen dieser Arbeit und den vorgestellten Algorithmen. Das eigene Konzept wurde so gestaltet, dass die Art des LiDAR-Sensors irrelevant ist, solange dieser dreidimensionale Punktwolken ausgibt und eine ausreichend große Menge an Punkten, vergleichbar mit den in Kapitel 2.2 vorgestellten Sensoren, erfassen kann. Dieser Aspekt ist wichtig, da die Skripts einen Vergleich der Sensorik ermöglichen sollen. Daher konnte nicht mit Algorithmen

gearbeitet werden, welche auf Muster bestimmter Scanningverfahren beruhen.

Vergleich der Konzepte					
Algorithmen	Konzept	KISS-ICP [23]	LOAM [20]	Mathworks A [69]	Mathworks B [68]
LiDAR-Sensor	jede Art	jede Art	rotierend	jede Art	jede Art
Registrierungs-algorithmus	ICP, NDT	ICP (PzP)	LOAM (ICP ähnlich)	GICP	NDT
Schleifener-kennung	wahlweise	Nein	Nein	Nein	Ja
Initialtransfor-mation	MdkG	MdkG	IMU oder MdkG	INS	IMU

Tabelle 5.1.: Übersicht der Unterschiede und Gemeinsamkeiten der betrachteten Algorithmen

Aufgrund dessen, dass in Matlab bereits gängige Registrierungsverfahren mit der „Lidar Toolbox“ inkludiert sind, fällt die Implementierung dieser sehr simpel aus. Daher wurde die Entscheidung getroffen, neben dem Vergleich der Sensoren, einen Vergleich von Registrierungsalgorithmen hinzuzufügen. Aus den vorhandenen Funktionen wurden die gängigen ICP-Algorithmen sowie der NDT-Algorithmus ausgewählt. ICP gehört zu den grundlegendsten und ältesten Algorithmen im Bereich der Kartierung, siehe [33], [34] sowie [35] und findet bis heute Anwendung in modernen SLAM-Lösungen, wie zum Beispiel KISS-ICP [23]. Um der Relevanz nachzugehen, wurden der Algorithmus und seine Abwandlungen miteingebunden. Die Wahl des NDT-Algorithmus wird damit begründet, dass er jünger ist, laut der KITTI-Benchmark weniger Anwendung findet [22] und eine grundlegend andere Funktionsweise vorweist.

Bei der Recherche fiel auf, dass vergleichsweise wenige Anwendungen die Erkennung von Schleifen implementieren. Unter der KITTI-Benchmark wurde kein solcher Anlgorithmus gefunden [22]. Da die Einbindung in Matlab, aufgrund von vorgefertigten Funktionen jedoch sehr simpel ausfällt, wurde hier ebenfalls die Entscheidung getroffen eine solche Erkennung miteinzubeziehen. Dabei soll untersucht werden, inwiefern eine solche Erkennung, unter den gegebenen Testbedingungen sinnvoll sein kann. Die Umsetzung orientiert sich dabei am Beispiel Mathworks B [68] und nutzt die bekannten SKDs zur Erkennung solcher Schleifenpunkte.

Für den Abgleich von Punktwolken ist eine vorausgehende Schätzung der Transformation, eine sog. Initialtransformation notwendig. Dies entspringt der Tatsache, dass ein Registrierungsablauf schneller geschehen kann, wenn die Punktwolken bereits nä-

her zueinader ausgerichtet wurden. Des Weiteren können die für den Registrierungsalgorithmus notwendigen Parameter genauer definiert werden, da die Transformationsschritte kleiner ausfallen. Daraus resultieren neben geringeren Rechenzeiten auch höhere Genauigkeiten. Kartierungsalgorithmen wie zum Beispiel KISS-ICP arbeiten allein mit den Daten der LiDAR-Sensorik und berechnen räumliche Transformationen dementsprechend nur mit Hilfe dieser [23]. Andere Methoden nutzen weitere Sensorik, wie zum Beispiel IMUs zur Erfassung der Orientierung. Für das Konzept dieser Arbeit wurde zur Bestimmung der Initialtransformation das MdkG gewählt. Gründe für diese Entscheidung sind zum einen, dass die verwendeten LiDAR-Sensoren nicht über einheitliche IMUs verfügen und zum anderen keine weitere externe Sensorik zur Verfügung stand. Des Weiteren erzielt das MdkG augenscheinlich ebenbürtig gute Ergebnisse, wie der KISS-ICP Algorithmus vorweisen kann. Zudem liegt das Ziel dieser Arbeit darin, die Tauglichkeit der LiDAR-Sensoren zu vergleichen. So könnten Defizite dieser Sensoren ausfindig gemacht werden, welche durch externe Sensorik eventuell ausgeglichen werden würden. Auf eine zusätzliche Entzerrung der Punktwolken wurde aufgrund der geringen Fahrtgeschwindigkeiten verzichtet. Zudem variiert die Punktmenge des Livox pro Frame, wodurch keine gleichmäßigen Zeitstempel verteilt werden können. Durch das Messen mittels Matlab konnte außerdem nicht auf die Punktzeitstempel des Ouster zugegriffen werden.

5.2. Vorverarbeitung der Punktwolken

In den folgenden Programmablaufplänen 5.8 und 5.10 wird die Vorverarbeitung der Punktwolken als alleiniger Funktionsblock dargestellt. Dieser Prozess setzt sich jedoch aus mindestens zwei Unterfunktionen zusammen. Dazu gehört das Sampling bzw. Reduzieren der Punktwolkendichte sowie das Entfernen der Bodenpunkte.

5.2.1. Entfernen der Bodenpunkte

Abbildung 5.1 zeigt zwei LiDAR-Punktwolken der selben Position. Auf der linken Hälfte ist ein Frame des Livox Horizon und auf der rechten Hälfte ein Frame des Ouster OS1 zu erkennen. Um die Punktwolkenstruktur sichtbarer zu machen, wurden die Punkte vergrößert.

Durch ihre jeweiligen Scanningverfahren produzieren die beiden Sensoren unterschiedliche Muster. Diese sind vor allem auf glatten Oberflächen gut zu erkennen. Der selbe Effekt tritt auf, wenn die LiDAR-Sensoren den Boden scannen, wie in der Abbildung 5.1 gezeigt wird. Dadurch entstehen jedoch ungewollte Nachteile für den Registrierungsprozess. Es handelt sich bei diesen Punkten nicht um spezifische Strukturen des Bodens, sondern lediglich um solche die aufgrund der Funktionsweise des Sensors erzeugt werden. Folglich würde der Sensor in einem darauffolgenden Frame dieselben

Strukturen aufzeichnen, jedoch an einer anderen Stelle. Für den Registrierungsprozess sind diese Punkte hinderlich. Sie verfälschen das Ergebnis, da sie die Zielfunktionen des jeweiligen Algorithmus optimieren, indem sie nah beieinander liegen. Im Falle des Ousters würden die kreisrunden Bögen das Ergebnis so beeinträchtigen, dass die Transformation in die rückwärtige Richtung manipuliert werden würde, so dass die Bögen übereinander liegen. Um diesen Effekt zu mitigieren, wurden Funktionen implementiert die Bodenpunkte herausfiltern.

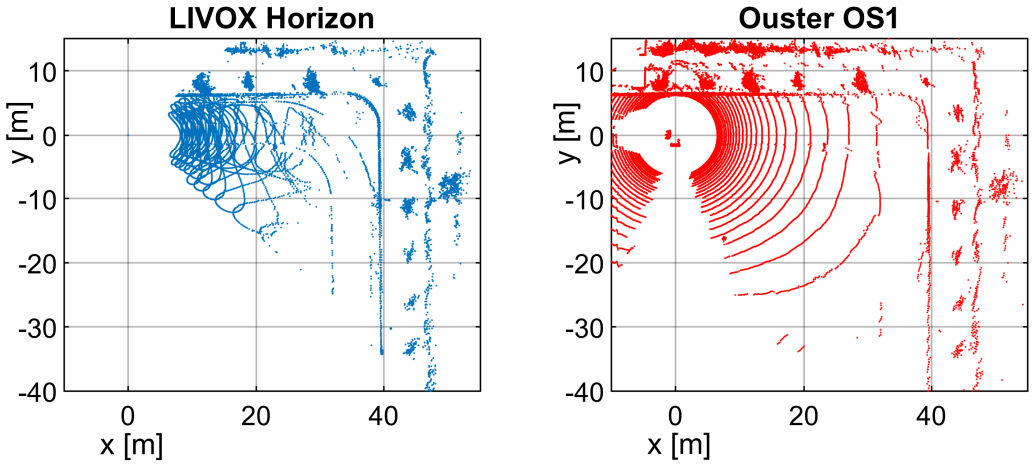


Abbildung 5.1.: Frame des Livox Horizon und Frame des Ouster OS1 mit Bodenpunkten

Matlab bietet auch hier eine vorgefertigte Funktion zur Erkennung dieser Punkte. Mit dem Befehl `segmentGroundFromLidarData` können die Indizes der erkannten Punkte ausgelesen werden [70]. Über `select` kann eine neue Punktwolke mit den selektierten Indizes erstellt werden. `segmentGroundFromLidarData` arbeitet allerdings nur mit organisierten Punktwolken und funktioniert dementsprechend nur mit dem Ouster OS1. Für die Punktwolken des Livox Horizon existiert eine weitere Möglichkeit um die Bodenpunkte zu erfassen. Der Befehl `pcfitplane` kann genutzt werden, um Flächen innerhalb einer Punktwolke zu approximieren. Die Funktion nutzt den M-estimator Sample Consensus (MSAC) Algorithmus, um eine Ebenengleichung zu erstellen, die die gestellten Bedingungen am besten erfüllt. Mit den Parametern `maxDistance`, `referenceVector` und `maxAngularDistance` wird der Funktion vorgegeben, welchen maximalen Abstand ein Punkt zur Fläche haben darf, in welche Richtung der Normalenvektor der Fläche zeigen soll und um wie viel Grad dieser abweichen darf [71].

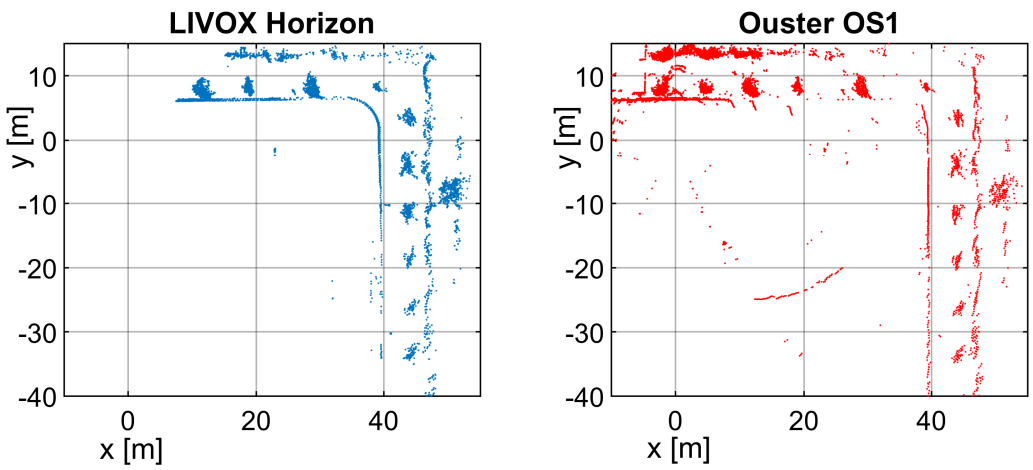


Abbildung 5.2.: Frame des Livox Horizon und Frame des Ouster OS1 nach Entfernung der Bodenpunkte

In Abbildung 5.2 sind die Frames aus Abbildung 5.1 zu erkennen, nachdem über die gennanten Methoden die Bodenpunkte herausgefiltert wurden. Wie zu erkennen ist, wurden nicht alle Punkte erfasst. Im Frame des Ouster OS1, welcher mit der `segmentGroundFromLidarData`-Funktion bearbeitet wurde, gehen zudem Punkte verloren, die nicht zum Boden gehören. Dennoch reicht die Güte der Funktionen aus, um einen genauen Registrierungsprozess zu garantieren. Die weitere Abbildung 5.3 zeigt am Beispiel des Livox Horizon, wie stark die Bodenpunkte das Ergebnis beeinflussen, bzw. wie viel genauer die tatsächliche Transformation bestimmt werden kann, wenn diese herausgefiltert wurden.

Obwohl der Boden der zweiten Punktwolken nicht vollständig herausgefiltert werden konnte, weist die Transformation ein deutlich besseres Ergebnis vor. Im oberen Bild ist ein klarer Spalt in Fahrtrichtung zwischen den Leitplanken der Testfläche zu erkennen. Mit dieser Ungenauigkeit können keine hinreichend guten Ergebnisse in einem tatsächlichen Kartierungsprozess produziert werden.

Für die Umsetzung der Bodenfilterung wurde ein `function`-Skript von Matlab verwendet, welches beide erwähnten Filter so kombiniert, dass je nach Struktur der Punktwolke die passende Methode gewählt wird. Das Skript heißt `helperProcessPointCloud` und wurde einem Kartierungsbeispiel von Mathworks entnommen [68].

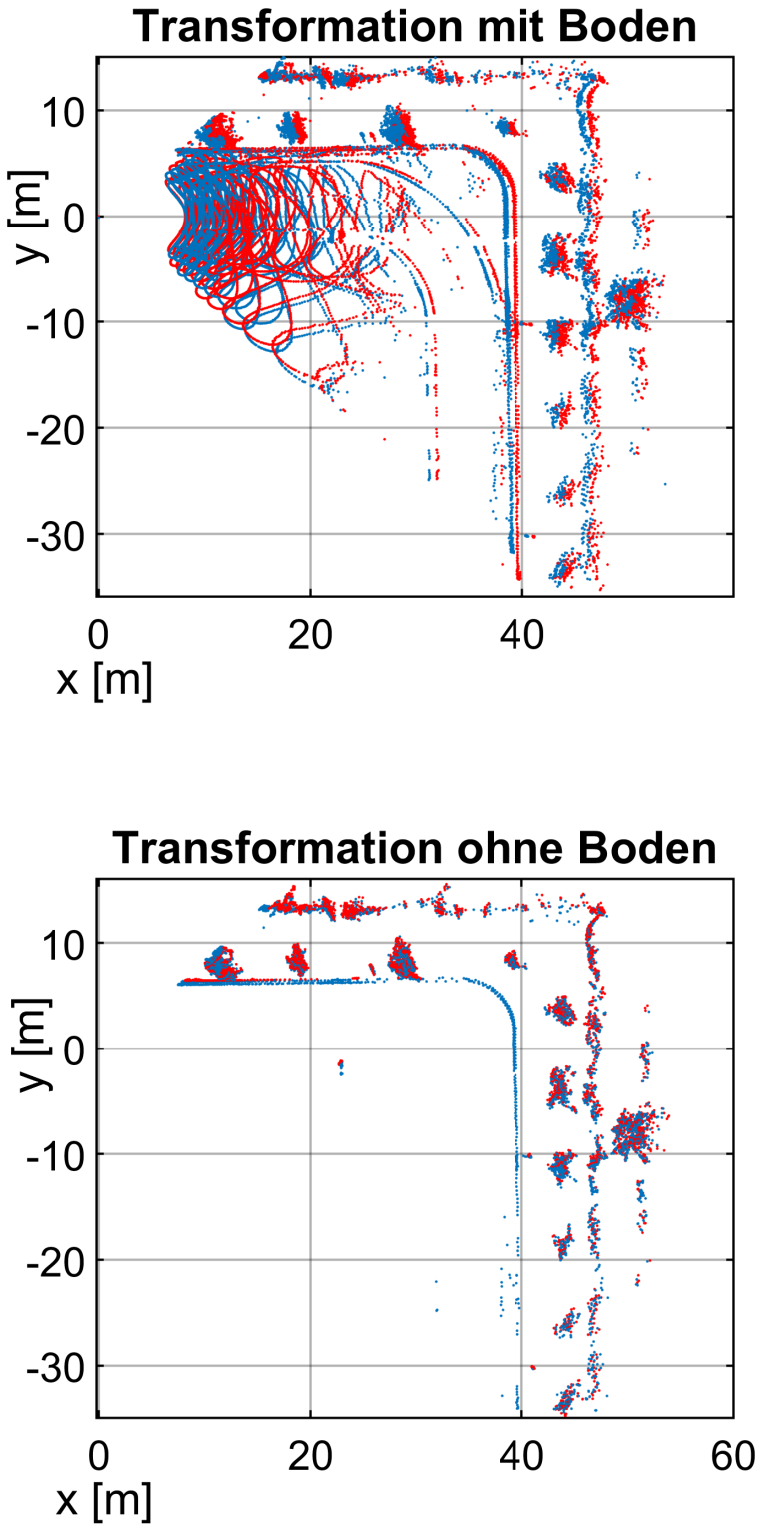


Abbildung 5.3.: Vergleich der Transformationsgenauigkeit zwischen Punktwolken des Livox Horizon mit Bodenpunkten und Punktwolken ohne Bodenpunkte

5.2.2. Punktwolkenfilterung

Es ist üblich die eingelesenen Punktwolken vor dem Registrierungsprozess auf eine minimale Anzahl relevanter Punkte zu reduzieren, bzw. ein Sampling vorzunehmen. Der ausschlaggebende Faktor dafür ist die Korrelation zwischen Punktanzahl und Rechenzeit. Je mehr Punkte miteinander verglichen werden müssen, desto länger dauert der Rechenprozess. Ausgehend davon gilt es ein Maß der Reduzierung zu ermitteln, welches die Rechenzeit auf ein Minimum senkt, ohne dabei die Genauigkeit der Transformation zu beeintächtigen.

Matlab bietet mit der LiDAR-Toolbox drei vorgefertigte Sampling-Methoden an [72]. Mit dem Befehl `pcdownsample` kann eine Punktwolke anhand drei verschiedener Methoden reduziert werden. Die Methoden und ihre jeweiligen Parameter werden in der folgenden Tabelle 5.2 aufgelistet.

Methode	**Parameter & Beschreibung**
random	Reduziert eine Punktwolke, sodass nur noch der durch percentage vorgegebene Anteil erhalten bleibt, die Auswahl der Punkte geschieht zufällig
gridAverage	Reduziert eine Punktwolke per Voxelisierung, gridSize spezifiziert dabei die Größe der dreidimensionalen Boxen
nonuniformGridSample	Reduziert eine Punktwolke per Voxelisierung, die Größe der Boxen wird jedoch durch die maximale innenliegende Anzahl an Punkten maxNumPoints vorgegeben und ist je nach lokaler Punktdichte unterschiedlich groß

Tabelle 5.2.: Übersicht der verfügbaren Sampling-Methoden in Matlab

In Abbildung 3.8 wurde bereits veranschaulicht wie Normal-basierte Methoden arbeiten. Die durch Matlab zur Verfügung stehenden Filter gehen jedoch nicht auf die Normalinformation der Punkte ein. Des Weiteren profitieren auch die hier verwendeten Algorithmen nicht von Punktwolken, die nach Flächen oder Kanten gefiltert wurden, wie es zum Beispiel der LOAM-Algorithmus tut. Aufgrund dessen werden auch nur die genannten Methoden, die eine Punktwolke lediglich reduzieren, benötigt.

Die `random`-Methode wurde nicht implementiert. Da diese Methode zufällig Punkte aussortiert, ist es schwer nachzuvollziehen, an welchen Stellen Punkte entfernt wurden und wo Punkte stehen gelassen wurden. Infolge dessen ist es ebenso schwer nachzuvollziehen, ob eventuelle Ungenauigkeiten in der Transformation daher entspringen, dass die Punktwolken an unterschiedlichen Stellen verschieden stark reduziert wur-

den. Die beiden anderen Methoden vollziehen im Vergleich dazu einen einheitlicheren Reduzierungsprozess.

Der Begriff „Voxelization“ bzw. Voxelisierung beschreibt das Einteilen des dreidimensionalen Raumes in mehrere Zellen. Im vergleich zum NDT-Algorithmus, werden bei Voxelisierungsfiltern die Punkte innerhalb einer dieser Zellen gemittelt und zu einem Punkt zusammengefügt. Fällt lediglich ein einzelner Punkt in so eine Zelle, bleibt dieser Punkt erhalten. Entsprechend wird die Anzahl der Punkte innerhalb einer Wolke kleiner, desto größer der Parameter `gridSize` gewählt wird. Die `nonuniformGridSample`-Methode bedient sich der Möglichkeit die Größe der Zellen variabel zu gestalten. So wird lediglich durch den Parameter `maxNumPoints` festgelegt, wie viele Punkte maximal in einer Zelle liegen dürfen, die Einteilung des Raumes wird dann so gestaltet, dass immer maximal diese Anzahl an Punkten vertreten ist. Anschließend wird einer dieser Punkte zufällig ausgewählt. Aufgrund der weiteren Zufallskomponente wurde auf diese Form des Samplings ebenfalls verzichtet. Dementsprechend wurde der `gridAverage`-Filter ausgewählt. Weitere Vorteile des Filters sind, dass die ursprüngliche Struktur der Punktwolke erhalten bleibt und dass die Punktdichte der Punktwolke ausgeglichen wird. Abbildung 5.4 zeigt den Frame des Ouster OS1, welcher bereits in Abbildung 5.1 dargstellt wurde, vor der Filterung und nach der Filterung mit einer Zellengröße `gridsize` von 1 m.

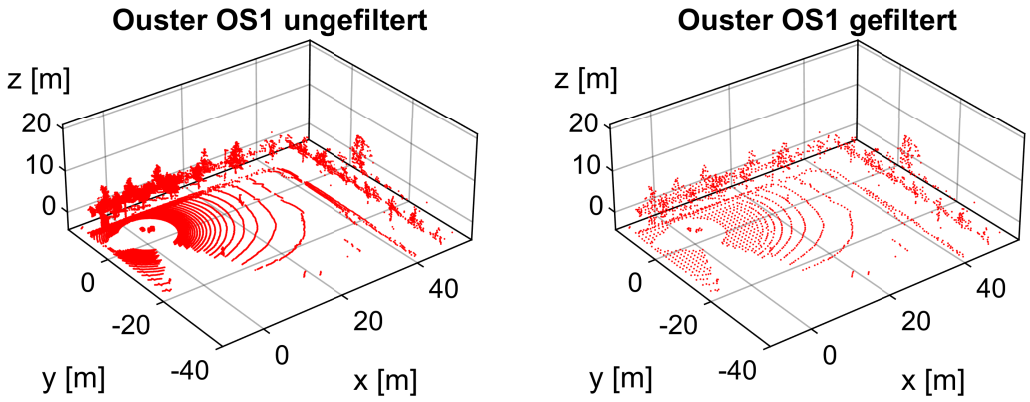


Abbildung 5.4.: Ungefilterter Frame des Ouster OS1 und gefilterter Frame

Die Punktwolke besitzt vor der Filterung 65.536 Punkte. Nach dem Sampling sind es lediglich noch 5.366 Punkte. Um den Einfluss auf den Registrierungsprozess verdeutlichen zu können, werden in Tabelle 5.3 die Werte der Transformationen zwischen zwei ungefilterten und zwei gefilterteten Frames des Ouster verglichen. In Abbildung 5.5 werden beide Ergebnisse visuell dargestellt. Der Boden wurde dabei aus allen Frames entfernt. Für die Registrierung wurden die zwei gleichen aufeinanderfolgenden Frames verwendet. Dabei wurde der GICP-Algorithmus genutzt.

Beide Transformationen sehen augenscheinlich gleich aus. Die Transformation der ungefilterten Punktwolken weist jedoch einen geringeren RMSE-Wert auf, was auf eine genauere Ausrichtung deuten könnte. Da der Voxelisierungsfilter jedoch zu einer gleichmäßigeren Punktverteilung führt, kann dieses Ergebnis auch daher kommen, dass die ungefilterten Punktwolken eine höhere Dichte an Punkten in den Bereichen aufweisen, in denen nach der Transformation viele Punkte nah beieinander liegen. Dazu gehören Punkte, die auf der Leitplanke und den Bäumen gemessen wurden. Die gefilterten Frames haben relativ dazu mehr Punkte, die frei auf der Fläche liegen oder von der `segmentGroundFromLidarData`-Funktion nicht entfernt werden konnten. Da diese Punkte eine deutlich längere Distanz zu ihren nächstgelegenen Nachbarn aufweisen, heben sie den RMSE-Wert an. Die Rechenzeit des Algorithmus wurde durch das Sampling jedoch deutlich verringert und liegt bei einem Sechstel der Zeit, die für die vollständigen Punktwolken benötigt wurde.

Parameter	**Ungefiltert**	**Gefiltert**
RMSE [m]	0,2566	0,4460
Rechenzeit [s]	0,9634	0,1562

Tabelle 5.3.: Vergleich der Transformationswerte zwischen ungefilterten und gefilterten Frames

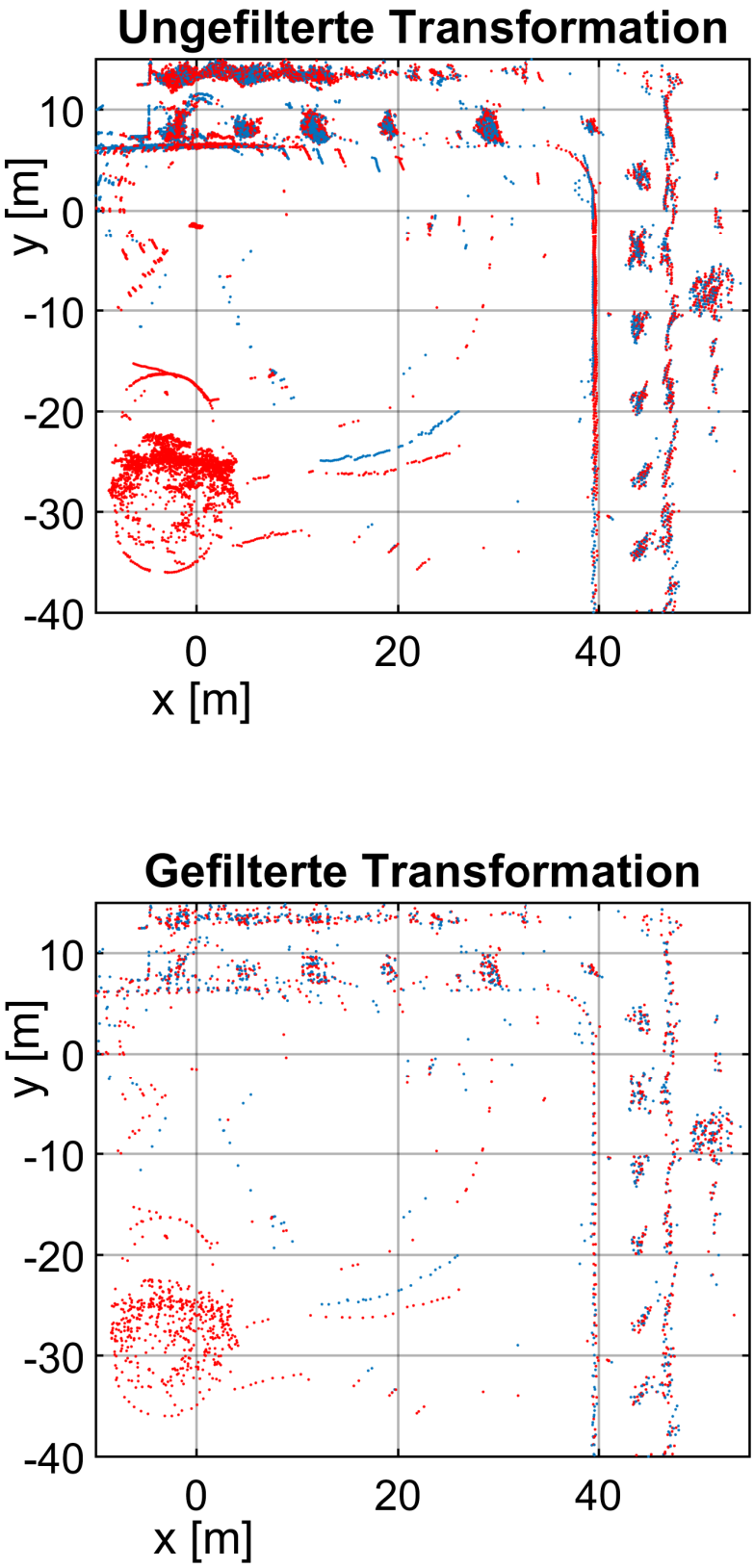


Abbildung 5.5.: Vergleich der Transformationen zwischen zwei aufeinanderfolgenden ungefilterten und gefilterteten Punktwolken des Ouster OS1

5.3. Implementierung der Registrierungsprozesse

Da das entwickelte Konzept für verschiedene Sensoren entworfen wurde und mit unterschiedlichen Registrierungsalgorithmen arbeitet, wurden mehrere Versionen erstellt. Bei der Aufteilung wurde beachtet, dass für jeden Sensor die funktionsgleichen Skripts existieren. Eine weitere Unterteilung wurde hinsichtlich der verwendeten Registrierungmethode umgesetzt. Zusätzlich dazu existiert für jedes Skript eine Version mit implementierter Schleifenschluss-Erkennung. Die Bezeichnungen der einzelnen Programme können der Tabelle 5.4 entnommen werden.

Sensor	**Skriptbezeichnung**
Livox Horizon	MapBuilder_ICP_LIVOX.m
	MapBuilder_NDT_LIVOX.m
	MapBuilder_ICP_Loop_LIVOX.m
	MapBuilder_NDT_Loop_LIVOX.m
Ouster OS1	MapBuilder_ICP_Ouster.m
	MapBuilder_NDT_Ouster.m
	MapBuilder_ICP_Loop_Ouster.m
	MapBuilder_NDT_Loop_Ouster.m

Tabelle 5.4.: Übersicht der entwickelten Matlab Skripts

Die Namensgebung wurde so gestaltet, dass nach der Programmbezeichnung „MapBuilder“ der verwendete Registrierungsalgorithmus notiert wird, gefolgt von „Loop“ für die Skripts, welche eine Graphoptimierung verwenden und dem jeweiligen Sensornamen. Eine Vereinheitlichung in ein Gesamtprogramm wäre möglich, diese Entscheidung wurde jedoch bewusst nicht getroffen. Da beide Sensoren, und die jewiligen Algorithmen unterschiedlich arbeiten, bzw. abweichende Ergebnisse produzieren, sind für jede Version unterschiedliche Parameter zu wählen. Um Verwechslungen zu vermeiden, wurden daher die verschiedenen Varianten erstellt. Die jeweiligen optimalen Parameter werden im Kopfteil des Skripts definiert. Die Anpassung je nach Anwendungsfall entfällt somit.

Für die Erstellung der Skripts in Matlab wurden die folgenden Toolboxen verwendet:

- `Computer Vision Toolbox (Version 10.3)` [73]
- `Image Processing Toolbox (Version 11.6)` [74]
- `Lidar Toolbox (Version 2.2)` [46]

- `Mapping Toolbox (Version 5.4)` [65]
- `Navigation Toolbox (Version 2.3)` [75]
- `Sensor Fusion and Tracking Toolbox (Version 2.4)` [76]

Die folgenden Unterabschnitte 5.3.1 und 5.3.2 sollen einen detailierteren Einblick in die Funktionsweise der entwickelten Programme geben.

5.3.1. Frame-zu-Frame Registrierung

Die Skripts ohne Schleifen-Erkennung und -Optimierung nutzen den zuvor erwähnten Frame-zu-Frame Abgleich. Der Programmablaufplan aus Abbildung 5.8 zeigt einen simplifizierten Ablauf der Funktionsweise dieser Skripts. Zwischen den einzelnen Versionen bestehen dabei wenige Unterschiede, die allerdings in Hinsicht auf die allgemeine Struktur der Algorithmen vernachlässigt werden können. So unterscheiden sich die benötigten Parameter je nach Registrierungsalgorithmus. Die Skripts, welche die ICP-Metriken verwenden, benötigen neben der Angabe, welche Metrik verwendet werden soll (PzP, PzF und FzF), den zusätzlichen Parameter `MaxInlierDistance`, welcher die maximale Distanz für zu akzeptierende Punktepaare festlegt. Die NDT-Skripts benötigen den Parameter `gridStep`. Mit diesem wird die Größe der Zellen definiert, für die eine eigene Normalverteilung bestimmt wird.

Alle weiteren Parameter werden von jeder Version des Konzepts verwendet. In Tabelle 5.5 werden diese aufgelistet und deren Zweck beschrieben.

Parameter	**Funktion**
skipFrames	Anzahl der Frames zwischen fixierter und bewegter Punktwolke, die übersprungen werden sollen
gridAverage	Größe der Zellen des Voxelisierungsfilters
mapGridSize	Auflösung der zusammengefügten LiDAR-Karte
numFrames	Gesamtanzahl der unverarbeiteten Frames
numSelectedFrames	Anzahl der tatsächlich verwendeten Frames

Tabelle 5.5.: Übersicht der restlichen Parameter und deren Funktion

Die Skripts werden durch die Schaltfläche „Run all sections“ gestartet. Danach öffnet sich ein Kontextmenü, über welches die jeweilige Messdatei ausgewählt wird. Matlab öffnet im Anschluss die Datei und hinterlegt diese im Workspace. Durch die Parameter `numFrames`, `numFramesSelected` und `skipFrames` wird der benötigte Speicherplatz für die später generierten Variablen vorab definiert, um Rechenzeit einzusparen (siehe Abbildung 5.6). Die Punktwolkenobjekte werden über die Variable

ptCloudsSelected initalisiert. Dabei wird jeder Eintrag mit einem Punktwolkenobjekt, bestehend aus einem Punkt im Koordinatenursprung spaceHolderPcloud belegt. Gleiches wird für die absoluten Transformationen absPoses getätigt, hier wird jeder Eintrag mit einem leeren rigidtform3d-Objekt belegt.

```
%% Parameter
numFrames           = height(PointClouds);
numSelectedFrames   = ...
    floor((numFrames-skipFrames)/skipFrames);
spaceHolderPcloud   = [0 0 0];

%% Preallocation
ptCloudsSelected(1:numSelectedFrames) = ...
    pointCloud(spaceHolderPcloud);
absPoses(1:numSelectedFrames)         = rigidtform3d;
```

Abbildung 5.6.: Matlab Code zur Vorabdefinierung des benötigten Speicherplatzes

Nachdem die Variablen preallokiert wurden, beginnt die Registrierungsschleife. Diese startet mit dem ersten Frame und überspringt mit jeder Iteration so viele Frames, wie durch skipFrames vorgegeben wird. Das Überspringen von Frames ist notwendig, um die Größe der entstehenden Karte und somit die Rechenzeit minimal zu halten. Des Weiteren wird dadurch beeinflusst, wie viel Bewegung zwischen zwei registrierten Punktwolken stattfindet. Je geringer der Abstand zwischen zwei Punktwolken ist, desto größer wird der überlappende Bereich, was vorteilhaft sein kann. Es muss jedoch beachtet werden, dass das Verhältnis zwischen der Fehlertoleranz der Algorithmen und der tatsächlich stattgefundenen Transformation immer kleiner wird, je weniger Frames übersprungen werden. So kann es zu einer größeren Anhäufung von Fehlern kommen, wenn die Variable skipFrames zu gering definiert wird.

Die erste eingelesene Punktwolke wird lediglich vorverarbeitet und anschließend an erster Stelle (ViewId = 1) im sog. vSet mit einem leeren Transformationsobjekt absTform abgelegt. vSet ist ein pcviewset-Objekt. Diese Objekte werden in Matlab genutzt, um mehrere Punktwolken mit den zwischen ihnen bestehenden Relationen abspeichern zu können [77]. Dabei werden innerhalb des Objekts zwei Tabellen angelegt. Die dreispaltige Tabelle Views beinhaltet, sortiert nach der ViewId, die absoluten Positionen (AbsolutePose) und die dazugehörige Punktwolke. Die zweite Tabelle Connections besitzt vier Spalten und stellt die Verbindungen zwischen zwei absoluten Positionen dar. Dabei wird in der ersten Spalte die vorherige ViewId, in der zweiten Spalte die nächste ViewId und in der dritten Spalte die relative Transformation zwischen beiden abgespeichert. Die vierte Spalte InformationMatrix wird genutzt, um die „Constraints“, also die Wichtungen der Verbindungen abzuspeichern. Dies geschieht in Form einer 6x6 Matrix. Für den Frame-zu-Frame Abgleich ist diese Spalte jedoch vernachlässigbar, da jede Transformation mit der gleichen Wich-

tung bestimmt wird. Nach Ablegen der Punktwolke, wird sie als fixierte Punktwolke (`ptCloudPrev`) abgespeichert.

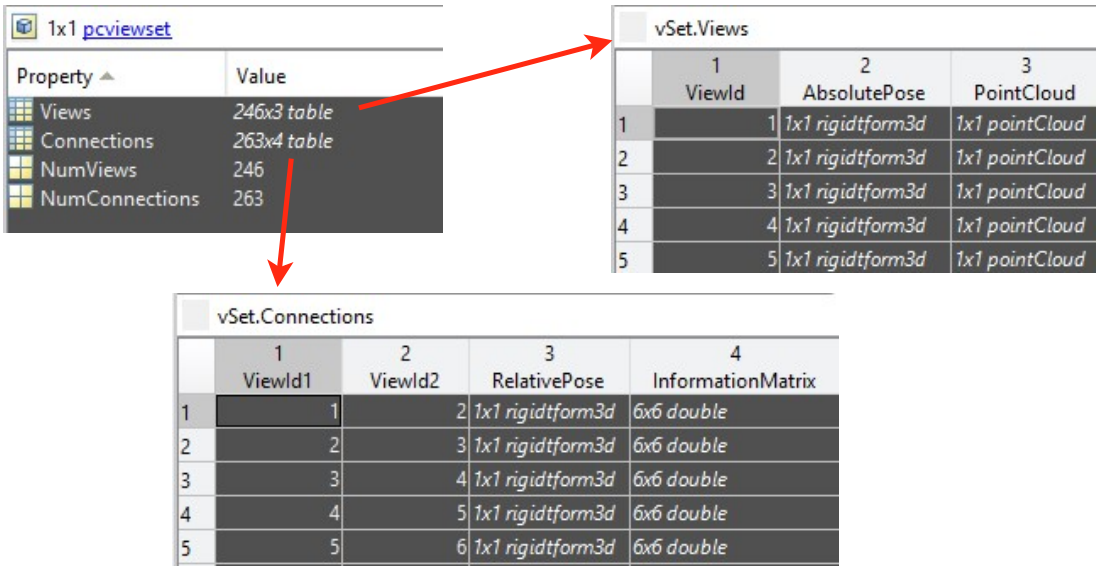


Abbildung 5.7.: Übersicht der Struktur eines `pcviewset`-Objekts

Mit dem Einlesen des zweiten Frames beginnt der tatsächliche Kartierungsprozess. Die Punktwolke wird wie die erste vorverarbeitet. Anschließend wird mittels der Funktionen `pcregistericp` oder `pcregisterndt` die relative Transformation `relTform` zwischen beiden Frames bestimmt. Im nächsten Schritt wird die neue absolute Position `absTform` berechnet, indem das Transformationsobjekt der alten Position mit dem der ermittelten relativen Transformation multipliziert wird. Über die Befehle `addView` und `addConnection` werden die neue Punktwolke, deren absolute Position und deren Relation zur vorherigen im `vSet` abgelegt. `ptCloudPrev` wird mit der aktuellen Punktwolke überschrieben und `relTform` als Initialtransformation `initTform` für die nächste Iteration hinterlegt.

Nachdem alle Frames registriert wurden, öffnet sich ein weiteres Kontextmenü, über welches das `vSet`, die Karte und die restlichen Parameter als .mat-Datei abgespeichert werden können. Im Anschluss werden über die Funktion `pcalign` der Lidar-Toolbox [46] alle registrierten Punktwolken des `vSet`, unter Eingabe der absoluten Positionen, zu einer großen Punktwolke `ptCloudMap` vereint. Durch `plot(vSet)` kann zusätzlich die Trajektorie des LiDAR-Sensors in die Karte geplottet werden.

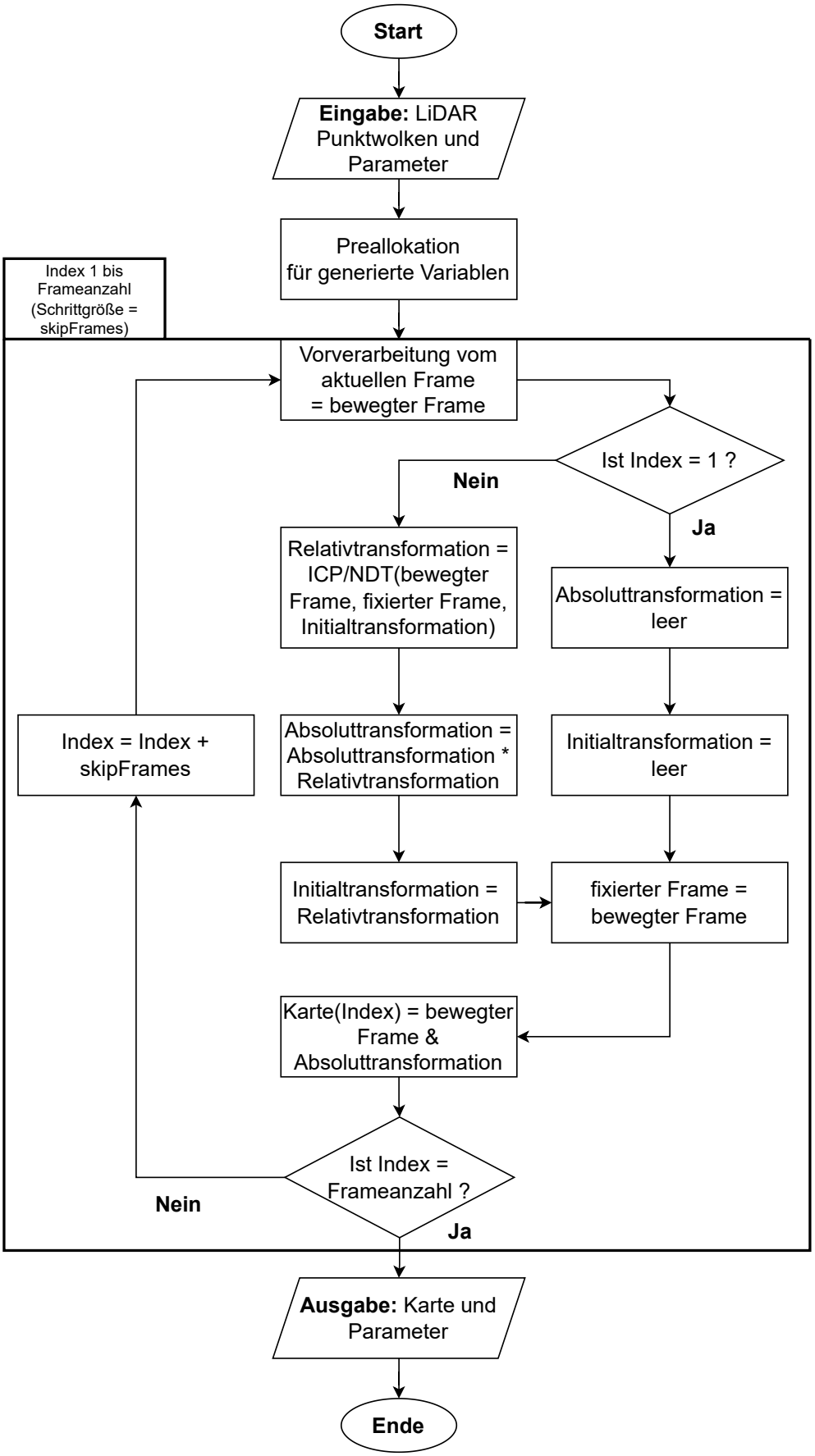


Abbildung 5.8.: Vereinfachter Programmablaufplan des Kartierungsalgorithmus ohne Schleifen-Optimierung

5.3.2. Graphoptimierung

Die Skripts mit Schleifen-Erkennung und -Optimierung nutzen neben den bereits erwähnten, weitere Parameter, Objektklassen und Unterfunktionen. Eine übersicht dieser wird in Tabelle 5.6 dargestellt.

Parameter/Objekt/Unterfunktion	**Beschreibung**
scanContextLoopDetector	Objekt in Matlab, zur Speicherung von SKDs mit zugehöriger ViewId [78]
scanContextDescriptor	Extrahiert einen SKD aus einer Punktwolke [32]
addDescriptor	Fügt dem scanContextLoopDetector einen SKD hinzu [78]
detectLoop	Sucht innerhalb des scanContextLoopDetector nach geschlossenen Schleifen [78]
maxTolerableRMSE	Maximal tolerierbarer RMSE-Wert einer Transformation, zwischen zwei Punktwolken einer geschlossenen Schleife
createPoseGraph	Erzeugt anhand eines vSet einen „Pose Graph“ [79]
optimizePoseGraph	Optimiert den „Pose Graph“ anhand der Wichtungen aus der Tabelle Connections [80]
updateView	Erstellt ein neues aktualisiertes vSet mithilfe des optimierten „Pose Graph“ [81]

Tabelle 5.6.: Übersicht der zusätzlichen Parameter, Objekte und Unterfunktionen für Skripte mit Schleifen-Optimierung

Im Unterschied zu den vorher betrachteten Skripts, wird hier mit jeder Iteration ein SKD der aktuellen Punktwolke extrahiert und im `scanContextLoopDetector` an der Stelle der aktuellen `ViewId` gespeichert. Mit Beginn des zweiten eingelesenen Frame wird die `detectLoop`-Funktion verwendet, um nach geschlossenen Schleifen zu suchen. Das Skript wird anschließend durch eine zusätzsliche If-Bedingung verzweigt. Wird eine mutmaßliche geschlossene Schleife erkannt, so wird der Code aus der Bedingung ausgeführt. Dabei wird die alte Punktwolke zu der eine Schleife geschlossen wurde, aus dem `vSet` entnommen und eine Transformation zu dieser bestimmt. Dies geschieht mit Hilfe des selbigen Registreirungsalgorithmus, welcher im iterativen Prozess verwendet wird. Optionalerweiser können hier die Parameter so angepasst werden, dass die Registrierung besonders genau ausgeführt wird, indem beispielsweise die Anzahl der maximalen Iterationen erhöht wird. Überschreitet das Ergebnis den vorher gesetzten Parameter `maxTolerableRMSE` wird der Schleifenpunkt ignoriert und das Skript regulär fortgesetzt. Anderenfalls wird mit dem Befehl `addConnection` dem

vSet eine Verbindung der zwei Punkte hinzugefügt. Es wird ein weiterer Eintrag in die Tabelle Connections getätigt, dabei wird in der ersten Spalte die ViewId der alten Punktwolke und in der zweiten Spalte die der aktuellen eingtragen. In der dritten Spalte wird die berchnete Transformation relTform abgelegt. Im Gegensatz zu den vorherigen Skripten wird an dieser Stelle Gebrauch von der vierten Spalte InformationMatrix gemacht. Hier wird eine 6x6 Diagonalmatrix, welche die Wichtung der Verbindung darstellt, für den Optimierungsprozess abgelegt. Die Einträge der Diagonalen werden hier um ein vielfaches kleiner gewählt als die der konsekutiven Verbindungen. So sollen die Auswirkungen einer falsch bestimmten Verbindung minimiert werden.

vSet.Connections

	1 ViewId1	2 ViewId2	3 RelativePose	4 InformationMatrix
122	122	123	1x1 rigidtform3d	6x6 double
123	123	124	1x1 rigidtform3d	6x6 double
124	10	124	1x1 rigidtform3d	6x6 double
125	124	125	1x1 rigidtform3d	6x6 double
126	125	126	1x1 rigidtform3d	6x6 double

Abbildung 5.9.: Eintrag einer erkannten Schleifenverbindung in der Tabelle Connections

Nach Durchlaufen aller Frames wird mit dem Befehl createPoseGraph das vSet in einen gewichteten Graphen transformiert. Durch die Funktion optimizePoseGraph kann dieser dann anhand der vorher festgelgten Verbindungen und Wichtungen optimiert werden. Hierfür wird das Optimierungsverfahren nach Levenberg-Marquardt verwendet, ein weiteres Lösungsverfahren von MKQ-Problemen, welches generell robuster als das Gauß-Newton-Verfahren ist [82]. Über den Befehl updateView kann aus dem optimierten Graphen ein optimiertes vSet, vSetOptim, zurückgeführt werden. Dieser Schritt ist notwendig, da der Graph nur die strukturellen Eigenschaften der Trajektorie beinhaltet, also die Positionen, Verbindungen und Relationen. Durch die Rückführung in ein pcviewset-Objekt kann wieder auf die Punktwolken und deren nun optimierten Transformationen zugegriffen werden.

Wie zuvor werden über ein Kontextmenü die Variablen in einer .mat-Datei gespeichert und mit dem Befehl pcallign nun zwei LiDAR-Karten, ptCloudMap und ptCloudMapOptim generiert.

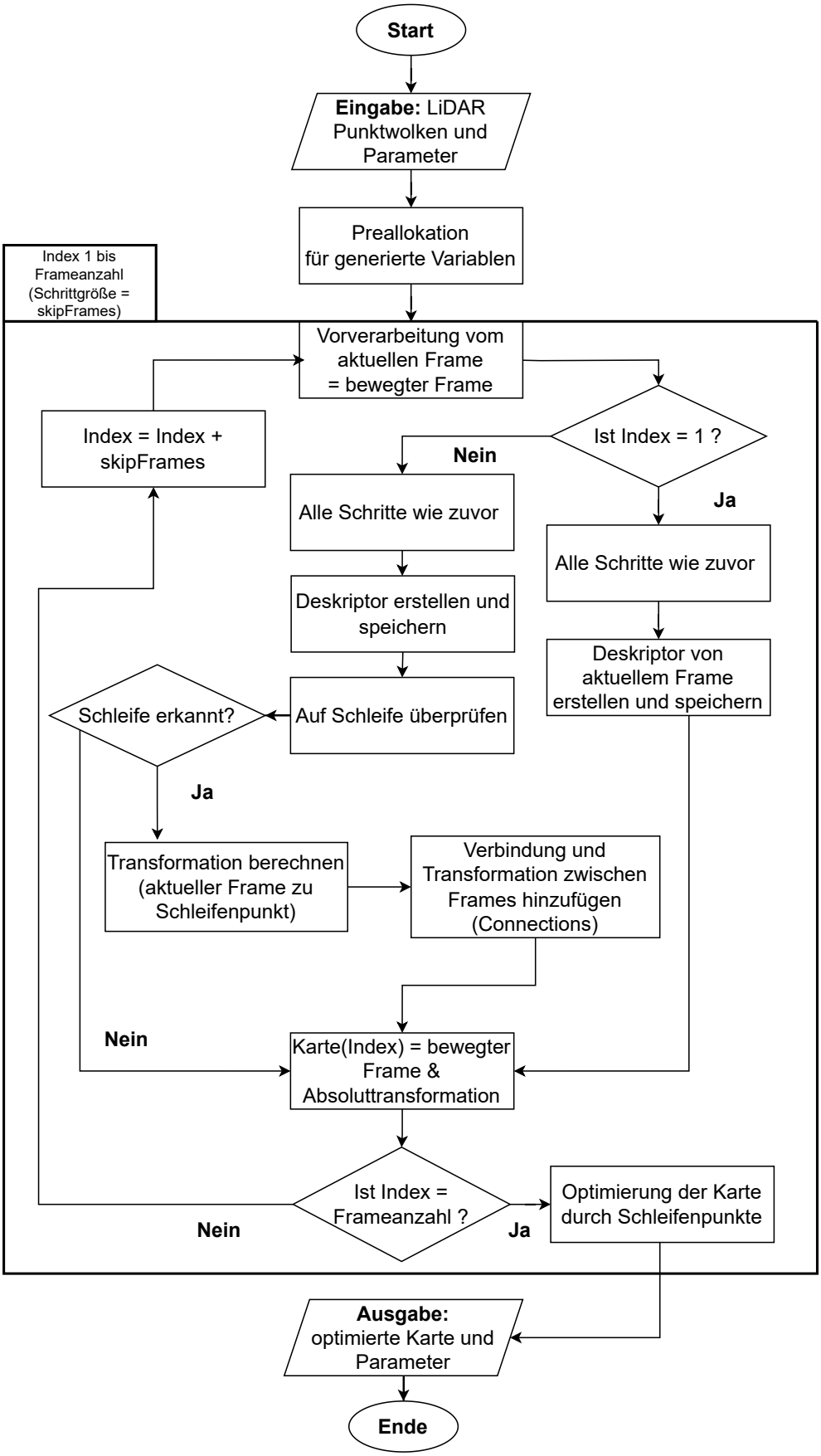


Abbildung 5.10.: Vereinfachter Programmablaufplan des Kartierungsalgorithmus mit Schleifen-Optimierung

6. Auswertung der Ergebnisse

Das sechste Kapitel widmet sich der Auswertung der eigens erstellten Konzepte. Dabei wird im ersten Abschnitt erläutert, welche Parameter für die Versuchsauswertung gewählt wurden. Im zweiten Teil wird erklärt wie die erzielten Ergebnisse ausgewertet und miteinander verglichen wurden. Anschließend werden die Ergebnisse innerhalb einer Zusammenfassung ausgewertet und visuell dargestellt.

6.1. Ermittlung der verwendeten Parameter

Im vorherigen Kapitel wurden die entwickelten Konzepte vorgestellt. Dabei wurden einige Parameter erwähnt auf welche die Skripte angewiesen sind und deren jeweilige Funktion erläutert. Da diese Werte einen erheblichen Einfluss auf das Gesamtergebnis nehmen, müssen sie sorgfältig ausgewählt werden. Dabei haben einige dieser Parameter eine größere Auswirkung als andere, bzw. exisitieren solche, deren Einfluss analytisch untersucht werden kann und solche bei denen dies nicht der Fall ist. Weitere Parameter können durch Überlegungen vorab bestimmt werden, ohne weitere Überprüfungen vornehmen zu müssen.

In der folgenden Tabelle 6.1 werden die Parameter aufgelistet, deren Wert durch Überlegungen bzw. Annahmen bestimmt wurde.

Parameter	**Zahlenwert**
maxDistance [m]	0,4
maxAngDistance [°]	5
skipFrames	5 / 10
maxTolerableRMSE [m]	1
InformationMatrix	0,001
mapGridSize [m]	0,2

Tabelle 6.1.: Übersicht der gewählten Parameterwerte

Die Parameter `maxDistance` und `maxAngDistance` der `pcfitplane`-Funktion wurden nach ihren Standardvorgaben durch Matlab gewählt [71]. Da es schwierig ist, eine automatiserte analytische Auswertung anzufertigen, welche überprüft bei welchen Parametereinstellungen die wenigsten Bodenpunkte erhalten bleiben, wurden verschiedene Parameterzusammensetzungen händisch ausprobiert und rein visuell ausgewertet. Dabei kam es zu dem Entschluss, dass die Standardvorgaben, die augenscheinlich robustesten Ergebnisse produzieren.

Die Anzahl der zu überspringenden Frames pro Registrierungsschritt (`skipFrames`) wurden so gewählt, dass pro Sekunde je zwei Frames der beiden Sensoren ausgewertet werden. Im Falle des Livox handelt es sich um die Anzahl der Packages. Da diese mit einer Frequenz von 20 Hz ausgegeben werden, wurde der `skipFrames`-Parameter verdoppelt. Das jeweils nächste Package wird dem ersten hinzugefügt, um so einen vollständigen Frame zu erstellen. Da die Messfahrten mit beiden Sensoren gleichzeitig durchgeführt wurden, stellt diese MEthode sicher, dass der räumliche Versatz zwischen zwei Punktwolken beider Sensoren immer gleichgroß ist. Somit vergehen zwischen zwei registrierten Frames 0,5 s. Bei einer maximalen Geschwindigkeit von 30 km/h besteht demnach ein maximaler Versatz von ungefähr 4,17 m zwischen zwei Punktwolken. Eine Erhöhung würde hier das Risiko von Fehltransformationen steigern und eine Senkung des Parameters würde dazu führen, dass in Bereichen geringerer Geschwindigkeiten kaum noch Bewegung zwischen zwei registrierten Punktwolken auftritt.

Die Parameter `maxTolerableRMSE` und `InformationMatrix` finden nur in den Skripts mit Schleifen-Optimierung Verwendung. Der maximal akzeptierbare RMSE-Wert wurde so gewählt, dass bei der Erkennung einer Schleife alle korrekten Transformationen sicher akzeptiert und nur diese, die mit hoher Wahrscheinlichkeit falsch sind ausgeschlossen werden. Da in den meisten Fällen der RMSE-Wert der regulären Frame-zu-Frame-Registrierungen nie über 1 m lag, wurde dieser Wert als Schwelle festgelegt. Die Wichtungen der Schleifenverbindungen, welche in der `InformationMatrix` hinterlegt sind, wurden im Vergleich zu den regulären Verbindungen, um ein vielfaches verringert. In den Versuchen fiel auf, dass eine identische Wichtung zu Verzerrungen des Graphen führen. Die Einträge der Diagonalen der konsekutiven Verbindungen betragen 0,01, die Einträge der Schleifen-Verbindungen wurden auf 0,001 reduziert.

`mapGridSize` hat keinerlei Einfluss auf das Ergebnis des Registrierungsprozess. Der Parameter legt lediglich die Auflösung fest, mit der die generierte LiDAR-Karte dargestellt wird. Er kann dementsprechend, je nach Bedürfnis frei gewählt werden. Für die Visualisierung der Ergebnisse hat sich eine Zellengröße von 0,2 m als optimal bewiesen. Genauere Darstellungen benötigen mehr Rechenzeit und können teilweise zu Problemen bei der Visualisierung führen und Matlab überlasten.

Das Sampling der Punktwolken hat einen direkten Einfluss auf die Registrierungsqualität und die benötigte Rechenzeit. Um den optimalen Wert des `gridSize`-Parameters ermitteln zu können, wurden beide Sensoren und die jeweiligen vier Registrierungsalgorithmen separat untersucht. Zur Ermittlung wurden automatisierte Skripts in Matlab implementiert (siehe Anhang **??**), welche mit einer Schrittgröße von 0,1 m über die Kartierungsalgorithmen iterieren. `gridSize` wird dabei in einem Bereich von 0,1 m

bis 2 m ausgewertet. Die Rechenzeiten sowie die RMSE-Werte wurden für jede Größe des Parameters abgespeichert, um diese später in Abhängigkeit dessen untersuchen zu können. Dabei wurden nach jeder Iteration die RMSE-Werte und Rechenzeiten pro Registrierung gemittelt und für jeden Sensor in einer eigenen Tabelle abgelegt (siehe B.1). Um die Ergebnisse zu veranschaulichen wurde für jeden Algorithmus ein Diagramm erstellt, welches die Rechenzeiten und RMSE-Werte beider Sensoren als Graphen darstellt (siehe B.1). Diese wurden genutzt um eine Entscheidung bezüglich der gewählten Parameter treffen zu können.

Sensor	**Algorithmus**	`gridSize` [m]
Livox Horizon	ICP-PzP	0,6
	ICP-PzF	0,6
	GICP	0,4
	NDT	0,2
Ouster OS1	ICP-PzP	0,6
	ICP-PzF	0,8
	GICP	0,8
	NDT	0,1

Tabelle 6.2.: Ermittelte Parameterwerte für `gridSize`

Deutlich zu erkennen ist, dass für die Punktwolken des Ouster OS1 immer ein vielfaches mehr an Rechenzeit benötigt wird. Die RMSE-Werte hingegen unterscheiden sich nicht so stark. Die Punktwolken des OS1 liefern nur geringfügig bessere RMSE-Werte, als die des Livox Horizon. Die Ausnahme macht hier der NDT-Algorithmus. Dieser weist im Gegensatz zu den ICP-Varianten eine deutlich schlechtere Transformation vor. Des Weiteren scheint er mit geringeren Punktmengen schlechter umgehen zu können, als der ICP-Algorithmus. Ab einer Größe des `gridSize`-Parameters von 1,1 m kann er aus den Punktwolken des Horizon keine Transformationen mehr berechnen, bzw. ab 1,5 m im Falle des Ouster OS1. Die ICP-Algorithmen weisen hier ein anderes Verhalten zwischen Rechenzeiten und Genauigkeit auf. Die gemittelte Dauer eines Registrierungsprozesses scheint mit steigendem `gridSize`-Parameter exponentiell abzunehmen und sich einem Grenzwert zu nähern. Dies trifft besonders auf die GICP- und PzF-Metrik zu. Die RMSE-Werte steigen dagegen scheinbar linear, bis ungefähr `gridSize` = 1,4 m. Danach scheinen diese Kurven abzuflachen. Letztlich wurden die Werte gewählt, an denen die Abnahme der Rechendauer abflacht, um ein optimales Kosten-Nutzen-Verhältnis zwischen Zeit und Genauigkeit garantieren zu können. Für den NDT-Algorithmus wurden, aufgrund der schlechteren Präzision,

die Parameter gewählt, an denen der RMSE-Wert am geringsten ist. So ergeben sich die in Tabelle 6.2 aufgelisteten Werte.

Die beiden verbleibenden Parameter `MaxInlierDistance` und `gridStep` wurden wie die vorherigen Parameter durch visuelle Überprüfung sowie durch Überlegungen ermittelt. `MaxInlierDistance` gibt vor, welche maximale Distanz zwischen zwei Punkten einer Paarung bestehen darf. Er ist demzufolge nur relevant für die ICP-Algorithmen. Aufgrunddessen, dass mit jedem Schritt eine Initialtransformation ausgeführt wird, liegen die Punktwolken bereits näher beieinander. Zu Beginn des Algorithmus liegt gar kein Versatz vor, da die Messungen aus dem Stand heraus gestartet werden. Infolgedessen wurde `MaxInlierDistance` auf 1 m gesetzt und verhindert so, dass Punkte die deutlich über der möglichen Entfernung liegen, ignoriert werden. `gridStep` bestimmt die Zellengröße des NDT-Algorithmus. Aufgrund der starken Abhängigkeit von großen Punktdichten, durfte die Zellengröße nicht zu klein definiert werden, da sonst zu wenig Punkte in jeder Zelle liegen, um aussagekräftige Normalverteilungen zu generieren. Wählt man diesen Wert zu groß verlieren die Normalverteilungen ebenfalls ihre Genauigkeit. Die Spanne in der sich der Wert befinden darf ist sehr gering. Bei einer Größe von 1 m waren die gefilterten Punktwolken bereits nicht mehr dicht genug, weshalb der Algorithmus regelmäßig keine Transformation mehr bestimmen konnte. Lag der Wert über 2 m, so wurden die Ergebnisse, trotz erfolgreicher Bestimmung einer Transformation, zu ungenau. Deshalb wurde `gridStep` auf 2 m begrenzt, um zu verhindern, dass der Algorithmus aufgrund von zu wenig Punkten abbricht sowie die Ergebnisse zu ungenau werden.

6.2. Methode der Versuchsauswertung

Um die Sensoren, sowie die Registrierungsalgorithmen analytisch vergleichen zu können musste eine Methode entwickelt werden mit welcher die Ergebnisse zuverlässig ausgewertet werden können. Dabei wurden anfänglich verschiedene Ansätze verfolgt. Es ist möglich präzise Fehlerwerte mithilfe eines fehlerkorrigierten GPS bzw. Differential Global Navigation Satellite System (DGNSS) zu ermitteln. Bei diesem Korrekturverfahren werden zwei Empfänger genutzt, um eine fehlerkorrigierte Position bestimmen zu können. DGNSS nutzt dafür genau eingemessene Bodenstationen. Sind beide Empfänger nah genug beieinander, unterliegt ihr GPS-Signal ähnlichen Ungenauigkeiten. Aufgrund dessen, dass die exakte Position der Bodenstation bekannt ist, kann so der Fehler herausgerechnet werden [83]. Ein solches Messsystem war während der Versuchsdurchführung jedoch nicht verfügbar.

Ein weiterer Ansatz, wäre die bekannten GPS-Koordinaten der Testfläche als Orientierungspunkte zu nutzen und anhand dieser die Abweichung zu berechnen. So können diese als Bezugspunkte genutzt werden, um die Position des Fahrzeugs bzw. der

Sensoren vor Beginn und nach Beendigung der Messfahrt zu bestimmen. Die ausgemessene Distanz der Start- und Endpunkte könnte mit der des Kartierungsalgorithmus berechneten Distanz verglichen werden, um so auf einen Fehlerwert zu kommen. Um jedoch den Rotationsfehler, zumindest den des Gierwinkels, berechnen zu können, müssten mehre Abstände ausgemessen werden. So wurde überlegt, ausgehend von den Radmittelpunkten des I3 den Abstand zu einem der GPS-Punkte auszumessen, jeweils zu Beginn und Ende jeder Messung. Über Winkelbeziehungen könnte so, neben dem absoluten Abstand auch die Orientierung der Sensorik vermessen werden. Die Umsetzung dieser Idee erwies sich jedoch als nicht hinreichend genau, da bereits beim Bestimmen der Sensorpositionen bezüglich der Radmittelpunkte Schwierigkeiten, sowie Messungenauigkeiten auftraten. Zusätzlich sind die GPS-Koordinaten nicht auf der Testfläche markiert worden, wodurch weitere Ungenauigkeiten durch Annahmen entstehen würden. Folglich musste eine andere Methode entwickelt werden.

Die Entscheidung fiel auf einen quantitativen Vergleich der Punktwolken. Da die Testfälle so ausgelegt wurden, dass die Start- und Endpositionen nah beieinander liegen, müssen folglich die Punktwolken dieser Positionen die gleiche Umgebung abbilden. Anhand von strukturellen Informationen aus den LiDAR-Messungen können so die Fehlerwerte bestimmt werden.

Die Spurmarkierungen der Testfläche erwiesen sich dabei als optimale Referenz. Durch ihren hohen Reflektionsgrad ließen sich die Punkte dieser leicht herausfiltern, umso die korrekte Struktur erhalten zu können. Werden hier geradlinige Segmente dieser Markierungen genutzt, können die Schnittwinkel zwischen ihnen berechnet werden. Die Translation zwischen den betrachteten Punktwolken kann über ein weiteres distinktives Objekt ermittelt werden. Während der Versuchsdurchführung wurder daher vor Beginn der Nord-Ost-Kurve auf die innerste Markierung ein Warnkegel platziert. Dieser besitzt ebenso starke Refelxionseigenschaften und kann daher gleichermaßen gut extrahiert werden. Das Auswertungskonzept wurde in Form eines Matlabskripts umgesetzt. Eine Automatisierung dessen war jedoch nicht möglich, da die Spurmarkierungen nur von Hand extrahiert werden konnten, um ein genaues Ergebnis garantieren zu können. Die Vorgehensweise des Skripts kann dabei mit den folgenden Schritten beschrieben werden:

1) Einlesen der Ergebnisse der Kartierungsalgorithmen (`vSet`)

2) Jeweils die ersten und letzten zehn Frames zusammenfügen und nach Intensität filtern

3) Die Punktwolken plotten und die innere Spurmarkierung mithilfe der Brush-Funktion extrahieren

4) Bestimmen von Regressionsgeraden mithilfe einer Hauptkomponentenanalyse (HKA)

5) Schnittwinkel zwischen Geraden und xy-Ebene bestimmen

6) Geraden in die xy-Ebene projizieren und Schnittwinkel zur Geraden der GPS-Punkte bestimmen

7) Rotationsmatrix aus den ermittelten Winkeln berechnen und die Punktwolken transformieren

8) Schritt 3) wiederholen und dabei zusätzlich die äußeren Spurmarkierungen extrahieren

9) mithilfe einer HKA die Ebenengleichungen der extrahierten Punkte bestimmen, sowie deren Schnittwinkel

10) Die ursprünglichen Punktwolken anhand der bestimmten Winkel rotieren und die Translation mittels der Kegelpositionen bestimmen

11) Ausgabe der Fehlerwerte: Translation, Gierwinkel, Rollwinkel und Nickwinkel

Die HKA wird in diesem Fall genutzt, um die senkrechten Abstände der Messpunkte, zum jeweiligen Modell, zu minimieren. So kann die Gerade bzw. Ebene gefunden werden, welche am besten durch die vorliegenden Daten beschrieben wird. Die Umsetzung im Skript orientiert sich dabei an einem Beispiel von Matlab, welches die vorgefertigte Funktion `pca()` benutzt [84]. Mithilfe der Stütz- und Richtungsvektoren der bestimmten Geraden und Ebenen können dann die Schnittwinkel berechnet werden, siehe Abbildung 6.1 und die dazugehörigen Formeln (6.1).

$$\cos(\alpha) = \frac{|u \cdot v|}{|u| \cdot |v|} \qquad \sin(\alpha) = \frac{|n_E \cdot u|}{|n_E| \cdot |u|} \qquad \cos(\alpha) = \frac{|n_E \cdot n_F|}{|n_E| \cdot |n_F|} \tag{6.1}$$

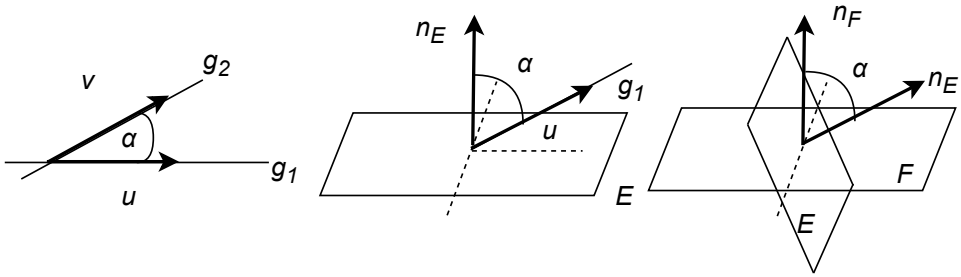


Abbildung 6.1.: Winkelbeziehungen zwischen Gerade-Gerade, Gerade-Ebene und Ebene-Ebene [85]

Aus den errechneten Fehlerwerten werden anhand der Rotationskonvention (3.7) ein Transformationsobjekt (`rigidtform3d`) erstellt und die ursprünglichen letzten Punktwolken transformiert. Zum Vergleich und zur Überprüfung der Genauigkeit folgt am

Ende des Skripts die Anfertigung eines Plots beider Punktwolken. Die folgende Abbildung 6.2 zeigt zwei `pcshowpair`-Plots von Punktwolken des Livox Horizon. Der obere Plot stellt die ersten und letzten Punktwolken ohne Fehlerkorrektur dar, im unteren sind die letzten Punktwolken (rot) korrigiert worden.

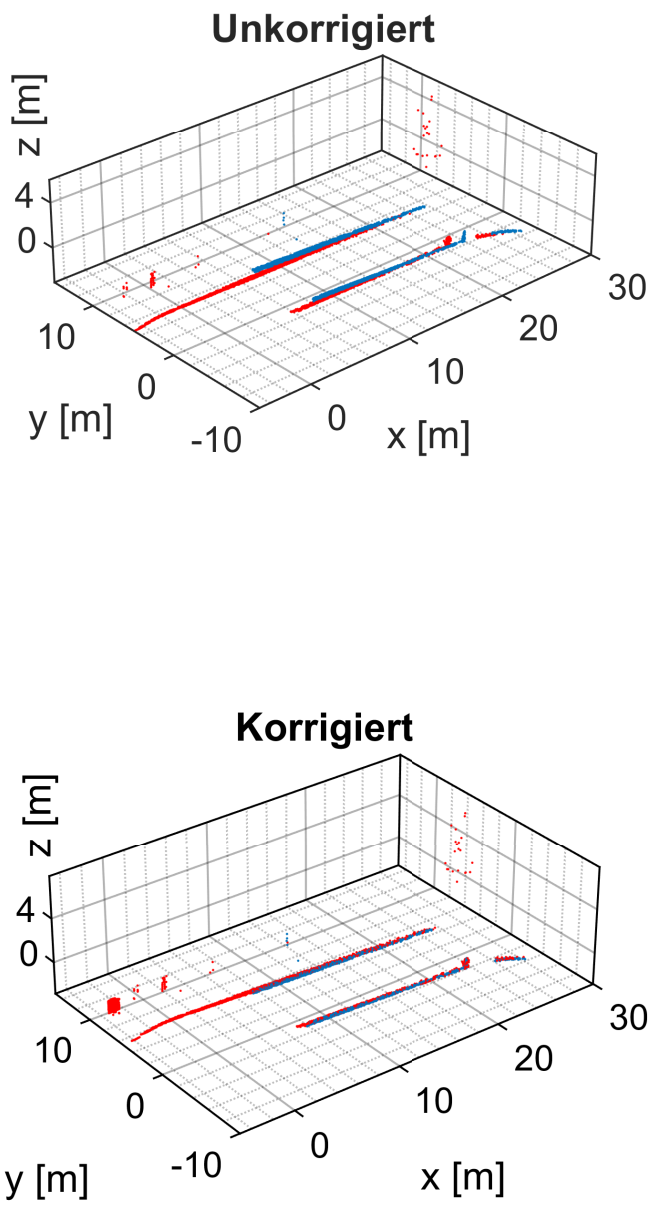


Abbildung 6.2.: Vergleich zwischen unkorrigierten (oben) und korrigierten Punktwolken (unten) des zweiten Testfalls

6.3. Vergleich und Bewertung der Algorithmen

Für die Untersuchung der Sensorik und Algorithmen wurden pro Testfall zwei Messfahrten durchgeführt, daraus ergaben sich sechs Messdateien pro Sensor, welche jeweils mit den vier verschiedenen Algorithmen ausgewertet wurden. Letzlich resultierten daraus 48 zu untersuchende theoretische Kartierungsergebnisse. Aufgrund der Schwierigkeit längere Aufzeichnungen des Ouster aufnehmen zu können, konnte für den dritten Testfall lediglich eine funktionsfähige Messdatei erzeugt werden. So reduziert sich die Anzahl der möglichen Ergebnisse auf 44. Die Kartierungsdateien wurden im .mat-Format gespeichert und können dem Datenträger entnommen werden. Die Bezeichnung der Dateien, wurde dabei foglendermaßen gestaltet: `Sensor_Algorithmus_Testfallnummer_Messungsnummer`. Die fehlgeschlagenen Kartierungen wurden mit dem Anhang `_failed` erkenntlich gemacht.

Eine tabellarische Auflistung der ausgewerteten Ergebnisse, sortiert nach Testfall, kann dem Anhang entnommen werden (siehe B.2 bis B.4). Neben den Tabellen wurden Säulendiagramme zur Veranschaulichung der Fehlerwerte angelegt. Diese wurden ebenfalls nach Testfall sortiert. Die einzelnen Messungen sind innerhalb der Diagramme nebeneinander angeordnet worden, sodass sich die folgende Reihenfolge, von links nach rechts, ergibt: erste Messung Livox, zweite Messung Livox, erste Messung Ouster und zweite Messung Ouster. Die Sensoren wurden farblich voneinander getrennt. Da jede Messdatei mit vier Algorithmen ausgewertet wurde, können pro Diagramm maximal 16 Säuleneinträge verzeichnet werden. Für den Fall, dass die Kartierung einer Messdatei fehlgeschlagen ist, wurden die Einträge an der jeweiligen Stelle der Tabelle mit Bindestrichen gefüllt. Innerhalb der Diagramme wurden diese Fehlkartierungen ausgelassen, sodass eine Lücke bestehen blieb. Die Diagramme der Fehlergrößen sind in dieser Reihenfolge hinterlegt worden: Translation, Gierwinkel, Nickwinkel, Rollwinkel und Rechenzeit.

Eine Kartierung wurde als erfolgreich bewertet, wenn die Trajektorie sowie das Gelände der Testfläche eindeutig zu erkennen und formtreu waren. Sobald die erzeugte Karte derart an Form verlor, dass eine Auswertung nicht mehr möglich war, wurde diese als fehlerhaft klassifiziert. Von den 44 möglichen Ergebnissen wurden 26 als erfolgreich bewertet. Fünf Ergebnisse des ersten sowie vier des zweiten Testfalls wurden als fehlerhaft klassifiziert. Es ergibt sich eine Fehlerquote von 31,25 % für den ersten Testfall bzw. 25,00 % für den zweiten. Von diesen neun Fehlversuchen sind sieben aus Messungen des Livox hervorgegangen, die anderen beiden aus Messungen des Ouster. So ergibt sich eine Fehlerquote von 43,75 % für den Livox und 12,50 % für den Ouster, bei Betrachtung der ersten beiden Testfälle. Die verbleibenden theoretischen zwölf Ergebnisse sind dem dritten Testfall zuzuordnen.

Während der Versuchsauswertung fiel auf, dass keiner der Algorithmen imstande war, die Messungen des Livox erfolgreich zu kartieren. Ab einer gewissen zurückgelegten Strecke verlor jede Trajektorie ihre Form und die Punktwolken wurden fehlerhaft transformiert. Des Weiteren scheinen die Punktwolken des Livox nicht kompatibel mit der SKD-Methode zu sein, weshalb es in jedem Kartierungsprozess mehrfach zu falsch detektierten Schleifen-Punkten kam. Resultierend konnten die fehlerhaften Transformationen nicht ausgeglichen werden. Außerdem wurde das Ergebnis durch die falschen Verbindungen weiter verzerrt. Es wird davon ausgegangen, dass die Punktwolken des Livox einen zu kleinen Bereich abdecken und in Folge der wenig abwechslungsreichen Testfläche, die Unterscheidung zwischen den SKDs nicht mehr funktionieren kann. So blieben für den dritten Testfall lediglich die Ergebnisse des Ouster übrig. Von diesen vier Ergebnissen wurde eins als fehlerhaft klassifiziert.

Von allen vier Registrierungsalgorithmen schaffte es nur die PzF-Metrik des ICP-Algorithmus für jede Messdatei eine erfolgreiche Kartierung zu erstellen, mit Ausnahme der Messungen des Livox des dritten Testfalls. Danach folgt der GICP-Algorithmus, welcher im zweiten Testfall nur für eine Messfahrt des Livox funktionierte. Der PzP- sowie NDT-Algorithmus konnten jeweils lediglich vier der acht möglichen Ergebnisse der ersten beiden Tesfälle erfolgreich kartieren. Wobei die Fehlerquote des NDT-Algorithmus allein auf den Messungen des Livox beruht. Die PzP-Metrik weist für beide Sensoren die gleiche Fehlerquote auf. Im Falle des NDT-Algorithmus wird vermutet, dass die Punktwolken des Livox Horizon zu klein sind, um eine zuverlässige Kartierung gewährleisten zu können, da die Punktwolken des Ouster ohne Schwierigkeiten transformiert wurden.

Algorithmus	**gemittelter Translationsfehler [m]**	
	Livox Horizon	**Ouster OS1**
GICP	4,4779	1,1470
NDT	-	1,1524
ICP-PzF	2,7292	1,3612
ICP-PzP	3,7856	1,3976

Tabelle 6.3.: Translationsfehler der Algorithmen und Sensoren

In der Tabelle 6.3 wurden die gemittelten Translationsfehler für beide Sensoren aufgelistet. Es ist zu erkennen, dass der Ouster für jeden Algorithmus ein vielfach besseres Ergebnis produziert. Im Falle des GICP-Algorithmus ist der Translationsfehler in etwa viermal kleiner als der des Livox. Die Abweichungen zwischen den Algorithmen fallen dabei gering aus, mit einem maximalen Unterschied von 0,2506 m. Betrachtet man

die Fehler des Gierwinkels fällt ein ähnliches Verhalten auf (siehe Tabelle 6.4). Hier generieren jedoch die PzP- sowie die PzF-Metrik die genaueren Werte, bei Betrachtung des Ouster OS1.

Der Nick- und Rollwinkel spielen bei der Kartierung eine geringe Rolle, solange die Umgebung annähernd planar ist. So blieben die Abweichungen hier deutlich geringer. Mit Ausnahme des Livox, dieser produzierte häufig große Abweichungen ($>2°$). Dies führte zu einem Verlust der Planarität. Die Werte können den Tabellen im Anhang entnommen werden.

Algorithmus	**gemittelter Gierwinkelfehler [°]**	
	Livox Horizon	**Ouster OS1**
GICP	4,0229	1,3230
NDT	-	1,3345
ICP-PzF	3,6659	0,9124
ICP-PzP	3,0239	0,9945

Tabelle 6.4.: Gierwinkelfehler der Algorithmen und Sensoren

Anhand des dritten Testfalls kann nachvollzogen werden, wie sich die Fehlerwerte aus den Tabellen 6.3 sowie 6.4 mit zunehmender Distanz verändern. Der Translationsfehler des PzF-Algorithmus wächst in diesem Fall auf fast 18 m, während der GICP-Algorithmus lediglich einen Fehler von 2,7598 m aufweist. Der Fehlerwert des NDT-Algorithmus liegt bei 4,8650 m. Im dritten Testfall wird annähernd die gleiche Strecke zurückgelegt, wie in den vorherigen Testfällen kombiniert. Die Fehler des GICP- bzw. NDT-Algorithmus scheinen hier linear mit der zurückgelegten Strecke zu wachsen. Der Gierwinkelfehler hingegen vervierfacht sich beim GICP-Algorithmus. Der NDT-Algorithmus weist ein besseres Ergebnis für den Gierwinkel auf, jedoch steigt hier der Nickwinkelfehler deutlich. Tatsächlich sind die Fehlerwerte nach der Schleifen-Optimierung schlechter, in einigen Fällen gleich. Obwohl die sichtbare Genauigkeit der Karte zunimmt ändern sich die Fehler, die am Ende der Kartierung bestehen, nicht. Die Abbildung 6.3 zeigt einen Vergleich der korrigierten und unkorrigierten Karte des GICP-Algorithmus. Es wird vermutet, dass die Optimierung lediglich Ausreißer korrigiert, ohne dabei einen größeren Einfluss auf das Ende der Trajektorie zu haben. Außerdem werden nur der Anfang und das Ende der Trajektorie miteinander verglichen. Es können also keine Fehlerwerte für jeden Knoten des Graphen berechnet werden.

Vergleicht man die einzelnen Fehlerwerte der Algorithmen fällt auf, dass der GICP-Algorithmus am robustesten und tatsächlich unabhängiger von Parameter- sowie Sensorkonfiguration zu sein scheint. Obwohl der NDT-Algorithmus teilweise bessere Genauigkeiten aufweist, so benötigt er dafür in etwa die doppelte Rechenzeit. Des Weiteren ist sein Ergebnis deutlich abhängiger von der Punktmenge bzw. -dichte. Mit den deutlich kleineren Punktwolken des Livox kann er keine guten Ergebnisse erzielen. Die ICP-Algorithmen scheinen hier eine größere Toleranz zu besitzen. Ebenso fällt auf, dass die Rechenzeit in direktem Zusammenhang mit der Punktwolkengröße steht, da die Rechenzeiten des Livox immer geringer sind, als die des Ouster.

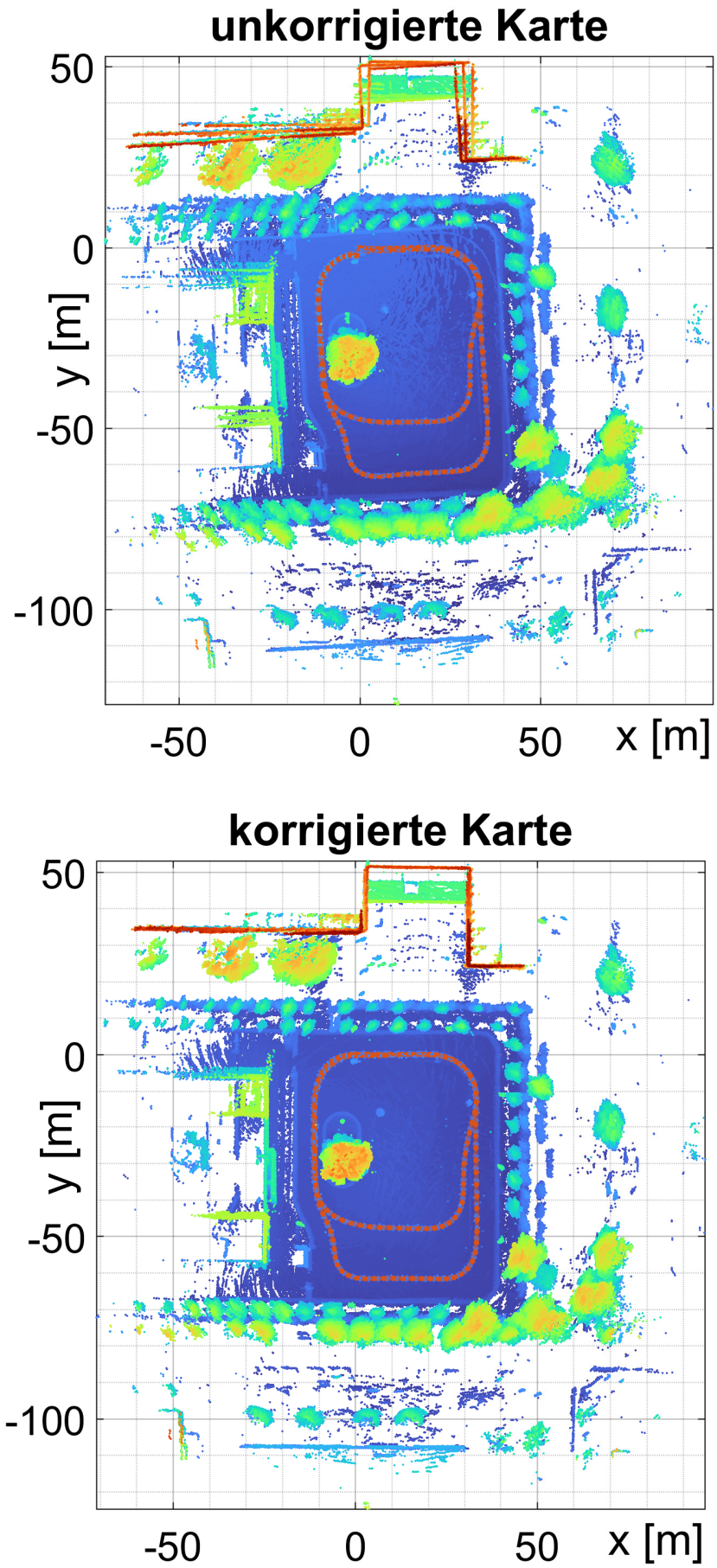


Abbildung 6.3.: Vergleich zwischen unkorrigierter und korrigierter LiDAR-Karte des dritten Testfalls (GICP)

7. Ausblick

In Anbetracht der erzielten Ergebnisse kann festgehalten werden, dass die Umsetzung eines Kartierungsalgorithmus mittels LiDAR-Sensorik innerhalb von Matlab möglich ist. Die verwendeten Sensoren von Livox und Ouster liefern hochauflösende, für den Anwendungsfall geeignete Punktwolken und lassen sich in Matlab implementieren. Zudem eignen sie sich für die Anbringung an Fahrzeugen und ermöglichen Messwertaufnahmen während der Fahrt. Des Weiteren zeigte die praktische Versuchsdurchführung, sowie die Auswertung der Ergebnisse, dass unter der alleinigen Verwendung von LiDAR genaue Kartierungen durchgeführt werden können. Die Erfassung und Kartierung von LiDAR-Daten in Echtzeit ist mit den entwickelten Programmen jedoch nicht möglich. Die Rechenzeiten innerhalb von Matlab übertreffen zum Teil die Eingangsrate der Messwerte, zudem ist die Kapazität mit speicherintensiven Dateien arbeiten zu können begrenzt. Die verwendeten Programme wurden jedoch nicht auf Anwendung in Echtzeit konzipiert. Es existieren bereits viele Entwicklungen auf diesem Gebiet, welche deutlich höhere Genauigkeiten erzielen, als es die hier vorgelegten Entwicklungen können. Das vorliegende Konzept weist Genauigkeiten im geringen Meterbereich auf. Eine Lokalisierung im Centimeterbereich ist allerdings nicht möglich. Für lokale Anwendungen auf der Testfläche, um etwa andere Versuche zu dokumentieren, scheinen die Konzepte jedoch ausreichend genau zu funktionieren. Für die Anwendung im Straßenverkehr sind die entwickelten Anwendungen allerdings nicht geeignet.

Im Hinblick auf reale Anforderungen im öffentlichen Verkehr fällt auf, dass für ein solches System viele weitere Optimierungen nötig sind. Dazu gehören Detektionsalgorithmen, welche bewegte Objekte herausfiltern, sowie die Möglichkeit Parameter abhängig von der aktuellen Situation verändern zu können. In straßentauglichen Fahrzeugen befinden sich zudem eine große Menge weiterer Sensoren, welche in ein solches System eingebunden werden müssen. So bestünde die Möglichkeit, dass auf Größen wie Geschwindigkeit, Beschleunigung und Gierwinkel direkt zugegriffen werden kann, ohne diese schätzen zu müssen. Des Weiteren können die Ergebnisse der LiDAR-Sensoren durch weitere Abstandssensorik, gerade im Nahbereich unterstützt werden. Durch die Versuche fiel auf, dass in den meisten Fällen ein Blickfeld von 360° einem eingeschränkten Sichtbereich überlegen ist. Ausgehend davon muss überlegt werden, wie und welche Sensoren in einem Fahrzeug verbaut werden können. Es könnten Sensoren der Art des Livox Horizon mehrfach an beliebigen Stellen verbauen, so dass sie bündig mit der Fahrzeugkarosserie sind und ebenfalls eine Abdeckung von 360° erzeugen.

Rotierende Sensoren lassen sich nur begrenzt gut positionieren. Die Montage auf dem Dach, könnte in vielen Situationen zu Problemen führen. Ein weiterer Aspekt, der beachtet werden muss ist, dass LiDAR-Sensoren sehr teuer sind. Der Ouster

OS1, in einer ähnlichen Konfiguration wie er in dieser Arbeit verwendet wurde, liegt preislich bei ungefähr 20.000 € [86]. Solch ein Sensor kann nicht in Serienfahrzeugen verbaut werden. Sollte sich die Rentabilität dieser Messtechnik zukünfitg verbessern, könnten die vorgestellten Konzepte jedoch erfolgreich implementiert werden und so zur weiteren Automatisierung und Sicherheit der Fahrzeuge beitragen.

Literatur- und Quellenverzeichnis

[1] o.V.:
Tesla Autopilot: Fahren in der Zukunft. Webseite. o.J. URL: https://www.tesla.com/de_DE/autopilot (besucht am 14. 06. 2023).

[2] F. Petit:
Entmystifizierung von LiDAR – Ein Überblick über die LiDAR – Technologie. Webseite. 2020. URL: https://www.blickfeld.com/de/blog/was-ist-lidar/ (besucht am 09. 02. 2023).

[3] H. Winner und S. Hakuli und G. Wolf:
Handbuch Fahrerassistenzsysteme. Hrsg. von H. Winner und S.Hakuli und F. Lotz und C.Singer. 3. Aufl. ATZ/MTZ-Fachbuch. Springer Vieweg Wiesbaden, 2015. ISBN: 978-3-658-05733-6. DOI: https://doi.org/10.1007/978-3-658-05734-3.

[4] G. Vosselman und H.G. Maas:
Airborne and terrestrial laser scanning. eng. Dunbeath, Caithness: Whittles Publishing, 2010. ISBN: 190444587X.

[5] Z. Liu und F. Zhang und X. Hong:
„Low-Cost Retina-Like Robotic Lidars Based on Incommensurable Scanning“. In: *IEEE/ASME Transactions on Mechatronics* 27.1 (2022), S. 58–68. DOI: 10.1109/TMECH.2021.3058173.

[6] M. Müller:
Die wichtigsten LiDAR-Parameter – Den Durchblick behalten. Webseite. 2020. URL: https://www.blickfeld.com/de/blog/lidar-parameter-verstehen/ (besucht am 10. 02. 2023).

[7] L. Moussy:
What are the different scan patterns of LiDAR systems? Webseite. 2021. URL: https://www.yellowscan-lidar.com/knowledge/what-are-the-different-patterns-of-lidar-scanners/ (besucht am 13. 02. 2023).

[8] o.V.:
3D LiDAR Technology Brought to Mass-Market with a $599 Livox Sensor. Webseite. 2019. URL: https://www.livoxtech.com/news/1 (besucht am 13. 02. 2023).

[9] o.V.:
LIVOX Wiki. 2. Introduction to LIVOX Scanning Patterns. Webseite. 2020. URL: https://livox-wiki-en.readthedocs.io/en/latest/introduction/livox_scanning_pattern.html (besucht am 09. 02. 2023).

[10] o.V.:
Livox Horizon User Manual. PDF (online). o.J. URL: https://www.livoxtech.com/3296f540ecf5458a8829e01cf429798e/assets/horizon/Livox%20Horizon%20user%20manual%20v1.0.pdf (besucht am 16. 02. 2023).

[11] o.V.:
IMU: BMI088 - High performance. Optimized for drones and robots. Webseite. 2022. URL: https://www.bosch-sensortec.com/products/motion-sensors/imus/bmi088/ (besucht am 20. 02. 2023).

[12] 0.V.
OS1 Gen 1: Mid-Range High-Resolution Imaging Lidar. PDF (Michael Swisher, Ouster Technical Support). 2022.

[13] 0.V.
ICM-20948: World's Lowest-Power 9-Axis MEMS MotionTracking® Device. Webseite. o.J. URL: https://invensense.tdk.com/products/motion-tracking/9-axis/icm-20948/ (besucht am 04. 07. 2023).

[14] A. Borkar:
SLAM für autonome Fahrzeuge. Webseite. 2020. URL: https://www.elektroniknet.de/automotive/assistenzsysteme/slam-fuer-autonome-%20fahrzeuge.172786.html (besucht am 18. 02. 2023).

[15] o.V.:
Cadence Extends Popular Tensilica Vision and AI DSP IP Product Line with New DSPs Targeting High-End and Always-On Applications. Vision Q8 and Vision P1 DSPs Broaden Support for Automotive, Mobile and Consumer Markets. Webseite. 2021. URL: https://www.cadence.com/en_US/home/company/newsroom/press-releases/pr/2021/cadence-extends-popular-tensilica-vision-and-ai-dsp-ip-product-l.html (besucht am 19. 04. 2023).

[16] H. Durrant-Whyte und T. Bailey:
„Simultaneous localization and mapping: part I“. In: *IEEE Robotics & Automation Magazine* 13.2 (2006), S. 99–110. DOI: 10.1109/MRA.2006.1638022.

[17] o.V.:
IEEE Xplore. Webseite. URL: https://ieeexplore.ieee.org/Xplore/home.jsp (besucht am 19. 04. 2023).

[18] K. Ćwianand und M. Nowicki und J. Wietrzykowski und P. Skrzypczyński:
„Large-Scale LiDAR SLAM with Factor Graph Optimization on High-Level Geometric Features“. In: *Sensors* 21.10 (2021). ISSN: 1424-8220. DOI: 10.3390/s21103445. URL: https://www.mdpi.com/1424-8220/21/10/3445.

[19] T. Bailey. und H. Durrant-Whyte:
„Simultaneous localization and mapping (SLAM): part II“. In: *IEEE Robotics & Automation Magazine* 13.3 (2006), S. 108–117. DOI: 10.1109/MRA.2006.1678144.

[20] J. Zhang und S. Singh:
„LOAM : Lidar Odometry and Mapping in real-time“. In: *Robotics: Science and Systems Conference (RSS)* (Jan. 2014), S. 109–111.

[21] T. Shan und B. Englot:
„LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain“. In: (2018), S. 4758–4765. DOI: 10.1109/IROS.2018.8594299.

[22] A. Geiger u.a.:
The KITTI Vision Benchmark Suite. Visual Odometry / SLAM Evaluation 2012. Webseite. 2023. URL: https://www.cvlibs.net/datasets/kitti/eval_odometry.php (besucht am 24. 04. 2023).

[23] I. Vizzo u.a.:
„KISS-ICP: In Defense of Point-to- Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way“. In: *IEEE Robotics and Automation Letters (RA-L)* 8.2 (2023), S. 1–8. DOI: 10.1109/LRA.2023.3236571.

[24] C. Stachniss:
Graph-Based SLAM in 90 Minutes. PDF (online). 2015. URL: https://www.unravel.rwth-aachen.de/global/show_document.asp?id=aaaaaaaaabebgwr%5C&download=1 (besucht am 18. 04. 2023).

[25] MATLAB und B. Douglas:
Understanding SLAM Using Pose Graph Optimization | Autonomous Navigation, Part 3. YouTube. 2020. URL: https://www.youtube.com/watch?v=saVZtgPyyJQ%5C&t=16s (besucht am 18. 02. 2023).

[26] J. Będkowski:
Large-Scale Simultaneous Localization and Mapping. 1. Aufl. Springer Singapore, 2022. ISBN: 978-981-19-1972-5. DOI: https://doi.org/10.1007/978-981-19-1972-5.

[27] o.V.:
3D lidar SLAM: The Basics. Webseite. 2022. URL: https://www.kudan.io/blog/3d-lidar-slam-the-basics/ (besucht am 22. 04. 2023).

[28] o.V.:
Motion Compensation in 3-D Lidar Point Clouds Using Sensor Fusion. Webseite. o.J. URL: https://de.mathworks.com/help/lidar/ug/motion-compensation-in-lidar-point-cloud.html (besucht am 22. 04. 2023).

[29] G. Kim und A. Kim:
„Scan Context: Egocentric Spatial Descriptor for Place Recognition Within 3D Point Cloud Map“. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Okt. 2018, S. 4802–4809. DOI: 10.1109/IROS.2018.8593953.

[30] G. Kim, S. Choiand und A. Kim:
„Scan Context++: Structural Place Recognition Robust to Rotation and Lateral Variations in Urban Environments“. In: *IEEE Transactions on Robotics* 38.3 (2022), S. 1856–1874. DOI: 10.1109/TRO.2021.3116424.

[31] G. Kim:
SC-LeGO-LOAM. GitHub. 2020. URL: https://github.com/irapkaist/SC-LeGO-LOAM (besucht am 26. 04. 2023).

[32] o.V.:
scanContextDescriptor: Extract scan context descriptor from point cloud. o.J. URL: https://de.mathworks.com/help/vision/ref/scancontextdescriptor.html (besucht am 26. 04. 2023).

[33] P.J. Besl und N.D. McKay:
„A method for registration of 3-D shapes“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), S. 239–256. DOI: 10.1109/34.121791.

[34] Y. Chen und G. Medioni:
„Object modeling by registration of multiple range images“. In: *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. 1991, 2724–2729 vol.3. DOI: 10.1109/ROBOT.1991.132043.

[35] Z. Zhang:
„Iterative point matching for registration of free-form curves and surfaces“. In: *International Journal of Computer Vision* 13 (1994), S. 119–152.

[36] D. Lawrence:
Summary of LIDAR scan matching algorithms. Webseite. 2014. URL: https://daniel.lawrence.lu/blog/y2014m11d25/ (besucht am 16. 02. 2023).

[37] C. Stachniss:
ICP & Point Cloud Registration - Part 2: Unknown Data Association. YouTube. 2021. URL: https://www.youtube.com/watch?v=ktRqKxddjJk (besucht am 24. 02. 2023).

[38] C. Stachniss:
Prof. Dr. Cyrill Stachniss. Online Register Uni Bonn. o.J. URL: https://www.ipb.uni-bonn.de/people/cyrill-stachniss/ (besucht am 01. 05. 2023).

[39] C. Stachniss:
ICP & Point Cloud Registration - Part 1: Known Data Association & SVD. YouTube. 2021. URL: https://www.youtube.com/watch?v=dhzLQfDBx2Q%5C&t=661s (besucht am 24. 02. 2023).

[40] o.V.:
rigidtform3d: 3-D rigid geometric transformation. o.J. URL: https://de.mathworks.com/help/images/ref/rigidtform3d.html#d124e268471 (besucht am 04. 05. 2023).

[41] A. Segal, D. Hähnel und S. Thrun:
„Generalized-ICP". In: *Proc. of Robotics: Science and Systems*. Juni 2009. DOI: 10.15607/RSS.2009.V.021.

[42] C. Stachniss:
Point-to-Plane and Generalized ICP – 5 Minutes with Cyrill. YouTube. 2021. URL: https://www.youtube.com/watch?v=2hC9IG6MFD0 (besucht am 10. 03. 2023).

[43] I. Bogoslavskyi:
ICP. Jupyter Notebook (online). o.J. URL: https://nbviewer.org/github/niosus/notebooks/blob/master/icp.ipynb#Using-point-to-plane-metric-with-Least-Squares-ICP (besucht am 03. 03. 2023).

[44] P. Biber und W. Straßer:
„The normal distributions transform: a new approach to laser scan matching". In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)* 3 (2003), 2743–2748 vol.3.

[45] o.V.:
MATLAB - Übersicht. Webseite. o.J. URL: https://de.mathworks.com/products/matlab.html (besucht am 06. 06. 2023).

[46] o.V.:
Lidar Toolbox: Entwurf, Analyse und Testen von LiDAR-Verarbeitungssystemen. Webseite. o.J. URL: https://de.mathworks.com/products/lidar.html (besucht am 06. 06. 2023).

[47] o.V.:
pointCloud: Object for storing 3-D point cloud. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ref/pointcloud.html (besucht am 09. 06. 2023).

[48] o.V.:
findPointsInROI: Find points within a region of interest in the point cloud. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ref/pointcloud.findpointsinroi.html (besucht am 09. 06. 2023).

[49] o.V.:
pcregistericp: Register two point clouds using ICP algorithm. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ref/pcregistericp.html (besucht am 09. 06. 2023).

[50] o.V.:
pcregisterndt: Register two point clouds using NDT algorithm. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ref/pcregisterndt.html (besucht am 09. 06. 2023).

[51] o.V.:
pctransform: Transform 3-D point cloud. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ref/pctransform.html (besucht am 09. 06. 2023).

[52] o.V.:
pcshowpair: Visualize difference between two point clouds. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ref/pcshowpair.html (besucht am 09. 06. 2023).

[53] o.V.:
Lidar Viewer: Visualize and analyze lidar data. Webseite. o.J. URL: https://de.mathworks.com/help/lidar/ref/lidarviewer-app.html (besucht am 09. 06. 2023).

[54] o.V.:
LVX Specifications v1.1.0.0. PDF (online). 2019. URL: https://www.livoxtech.com/3296f540ecf5458a8829e01cf429798e/downloads/Livox%20Viewer/LVX%20Specifications%20EN_20190924.pdf (besucht am 16. 02. 2023).

[55] o.V:
Livox Viewer User Manual v1.0. PDF (online). 2020. URL: https://www.livoxtech.com/3296f540ecf5458a8829e01cf429798e/downloads/Livox%20Viewer/Livox%20Viewer%20User%20Manual.pdf (besucht am 16. 02. 2023).

[56] F. Blechschmidt:
„Entwicklung einer Verkehrszeichenerkennung mittels Lidar-Sensorik“. HTW Dresden / Mechlab. 2022.

[57] o.V.:
Timetables: Time-stamped data in tabular form. Webseite. o.J. URL: https://de.mathworks.com/help/matlab/timetables.html (besucht am 09. 06. 2023).

[58] o.V.:
Lidar Toolbox Support Package for Ouster Lidar Sensors: Design, analyze, and test lidar processing systems. Webseite. o.J. URL: https://de.mathworks.com/help/supportpkg/ousterlidar/index.html (besucht am 09. 06. 2023).

[59] o.V.:
Ouster Studio: Digital Lidar Visualizer. Webseite. o.J. URL: https://ouster.com/products/software/ouster-studio-visualizer/ (besucht am 12. 06. 2023).

[60] o.V.:
What are Organized and Unorganized Point Clouds? Webseite. o.J. URL: https://de.mathworks.com/help/lidar/ug/organized-and-unorganized-point-clouds.html (besucht am 17. 06. 2023).

[61] o.V.:
HTW Dresden K-Gebäude. Google Maps. o.J. URL: https://www.google.de/maps (besucht am 14. 06. 2023).

[62] o.V.:
OpenStretMap - Deutschland. OpenStreetMap® sind offene Daten, lizenziert unter der Open Data Commons Open Database License (ODbL) von der OpenStreetMap Stiftung (OSMF). 2023. URL: https://www.openstreetmap.org/copyright (besucht am 23. 01. 2023).

[63] o.V.:
JOSM is an extensible editor for OpenStreetMap (OSM) for Java 8+. Software. o.J. URL: https://josm.openstreetmap.de/ (besucht am 17. 06. 2023).

[64] o.V.:
gpxread: Read GPX file. Webseite. o.J. URL: https://de.mathworks.com/help/map/ref/gpxread.html (besucht am 17. 06. 2023).

[65] o.V.:
Mapping Toolbox: Analyze and visualize geographic information. Webseite. o.J. URL: https://de.mathworks.com/help/map/index.html?s_tid=CRUX_lftnav (besucht am 17. 06. 2023).

[66] prakhar7. Haversine-Formel zum Ermitteln des Abstands zwischen zwei Punkten auf einer Kugel. Webseite. 2022. URL: https://de.acervolima.com/haversine-formel-zum-ermitteln-des-abstands-zwischen-zwei-punkten-auf-einer-kugel/ (besucht am 18. 06. 2023).

[67] o.V.:
Accelerating Urban Logistics: Eine gemeinsame Studie von BVL.digital und HERE Technologies. PDF (online. o.J. URL: https://go.engage.here.com/Accelerating-Urban-Logistics.html (besucht am 20. 06. 2023).

[68] o.V.:
Build a Map from Lidar Data Using SLAM. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ug/build-a-map-from-lidar-data-using-slam.html (besucht am 08. 02. 2023).

[69] o.V.:
Build a Map from Lidar Data. Webseite. o.J. URL: https://de.mathworks.com/help/driving/ug/build-a-map-from-lidar-data.html (besucht am 08. 02. 2023).

[70] o.V.:
segmentGroundFromLidarData: Segment ground points from organized lidar data. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ref/segmentgroundfromlidardata.html (besucht am 29. 06. 2023).

[71] o.V.:
pcfitplane: Fit plane to 3-D point cloud. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ref/pcfitplane.html (besucht am 29. 06. 2023).

[72] o.V.:
pcdownsample: Downsample a 3-D point cloud. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ref/pcdownsample.html (besucht am 02. 07. 2023).

[73] o.V.:
Computer Vision Toolbox: Entwickeln und Testen von Computer Vision-, 3D-Vision- und Videoverarbeitungssystemen. Webseite. o.J. URL: https://de.mathworks.com/products/computer-vision.html (besucht am 26. 06. 2023).

[74] o.V.:
Image Processing Toolbox: Durchführung der Bildverarbeitung, Visualisierung und Analyse. Webseite. o.J. URL: https://de.mathworks.com/products/image.html (besucht am 26. 06. 2023).

[75] o.V.:
Navigation Toolbox: Entwerfen, Simulieren und Bereitstellen von Algorithmen zur autonomen Navigation. Webseite. o.J. URL: https://de.mathworks.com/products/navigation.html (besucht am 26. 06. 2023).

[76] o.V.:
Sensor Fusion and Tracking Toolbox: Entwurf, Simulation und Test von Multisensor-Tracking- und Navigationssystemen. Webseite. o.J. URL: https://de.mathworks.com/products/sensor-fusion-and-tracking.html (besucht am 26. 06. 2023).

[77] o.V.:
pcviewset: Manage data for point cloud based visual odometry and SLAM. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ref/pcviewset.html (besucht am 27. 06. 2023).

[78] o.V.:
scanContextLoopDetector: Detect loop closures using scan context descriptors. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ref/scancontextloopdetector.html (besucht am 27. 06. 2023).

[79] o.V.:
createPoseGraph: Create pose graph. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ref/imageviewset.createposegraph.html (besucht am 27. 06. 2023).

[80] o.V.:
optimizePoseGraph: Optimize nodes in pose graph. Webseite. o.J. URL: https://de.mathworks.com/help/nav/ref/optimizeposegraph.html (besucht am 27. 06. 2023).

[81] o.V.:
updateView: Update view in view set. Webseite. o.J. URL: https://de.mathworks.com/help/vision/ref/imageviewset.updateview.html (besucht am 27. 06. 2023).

[82] J. E. Dennis:
Numerical methods for unconstrained optimization and nonlinear equations. eng. Classics in applied mathematics 16. Philadelphia, Pa: SIAM, 1996 - 1983. ISBN: 9780898713640.

[83] 0.V.
Präzise Positionsbestimmung mit Hilfe von GPS / GNSS. Webseite. o.J. URL: https://www.magicmaps.de/gnss-wissen/praezise-gps-messungen-mit-hilfe-von-dgps-und-rtk/?L=0 (besucht am 15. 07. 2023).

[84] o.V.:
Fitting an Orthogonal Regression Using Principal Components Analysis. Webseite. o.J. URL: https://de.mathworks.com/help/stats/fitting-an-orthogonal-regression-using-principal-components-analysis.html (besucht am 11. 07. 2023).

[85] o.V.:
Schnittwinkel zwischen Geraden und/oder Ebenen. Webseite. 2018. URL: https://abiturma.de/mathe-lernen/geometrie/lagebeziehungen-und-schnitt/schnittwinkel-zwischen-geraden-undoder-ebenen (besucht am 11. 07. 2023).

[86] o.V.:
Ouster OS1 64 Rev 6 Mid Range Lidar Sensor. Webseite. o.J. URL: https://eu.robotshop.com/products/ouster-os1-64-rev-6-mid-range-lidar-sensor (besucht am 18. 07. 2023).

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. Dirk Engert angefertigt. Hiermit erkläre ich, dass ich die vorliegende Arbeit zum Thema

„Entwicklung eines Algorithmus zur Bestimmung der Fahrzeugtrajektorie und Kartierung der Umgebung mittels LiDAR-Sensorik“

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

Tristan Weiland

Verzeichnis der Anhänge

A Mathematische Erläuterungen **91**

A.1 Optimierung der PzP-Zielfunktion mittels SWZ 91

A.2 Optimierung der PzF-Zielfunktion mittels Gauß-Newton-Verfahren . . 93

A.3 Herleitung des GICP-Algorithmus 95

B Tabellen und Diagramme der Auswertung **97**

B.1 Analyse des `gridSize`-Parameters 97

B.2 Testfall 1 100

B.3 Testfall 2 104

B.4 Testfall 3 108

C Datenträger **112**

A. Mathematische Erläuterungen

A.1. Optimierung der PzP-Zielfunktion mittels SWZ

Den mathematischen Ablauf einer Optimierung der Zielfunktion 3.11 wird folgend nach Cyrill Stachniss erläutert [39]. Um die Notation zu erleichtern, werden die Punktwolken P und Q auf die durch C vorgegebenen assoziierten Punktepaare reduziert. Die Punkte werden nun nach dem Index n sortiert, sodass der erste Punkt aus P in Zusammenhang mit dem ersten Punkt aus Q steht (q_n und p_n mit $n = 1, \ldots, N$). Des Weiteren wird davon ausgegangen, dass P und Q zweidimensionale Punktwolken sind. Dadurch verändern sich die vorher genannten Definitionen von R und t und werden für das folgende Beispiel durch die Formeln A.1 und A.2 beschrieben.

$$t = \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{A.1}$$

$$R = \begin{pmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{pmatrix} \tag{A.2}$$

Die Zielfunktion der Punkt-zu-Punkt Metrik (3.11) kann, um den Winkel der Rotation bestimmen zu können, mittels einer Singulärwertzerlegung (SWZ) minimiert werden [39]. Dabei sind die folgenden Rechenschritte nötig, um die gesuchten Variablen bestimmen zu können:

1) Berechnung der Flächenschwerpunkte von P und Q

2) Bilden der Kreuzkovarianzmatrix K

3) Bilden der SWZ von K

4) Berechnung der Rotation R aus den Werten der SWZ

5) Berechnung der Translation t

Die Flächenschwerpunkte q_0 und p_0 werden bestimmt, indem alle Punkte aufsummiert und durch die Anzahl dieser geteilt werden, siehe Formel (A.3).

$$q_0 = \frac{\sum q_n}{N} \qquad p_0 = \frac{\sum p_n}{N} \tag{A.3}$$

Mit den Schwerpunkten kann die Kreuzkovarianzmatrix K gebildet werden, welche den räumlichen Zusammenhang zwischen beiden Punktwolken beschreibt. Die Matrix K wird nach Formel (A.4) gebildet und weist, aufgrund der zweidimensionalen Punktwolken, die durch Gleichung (A.5) beschriebene Form auf.

$$K = \sum_{n=1}^{N} (q_n - q_0)(p_n - p_0)^T \tag{A.4}$$

$$K = \begin{pmatrix} cov(q_{nx}, p_{nx}) & cov(q_{nx}, p_{ny}) \\ cov(q_{ny}, p_{nx}) & cov(q_{ny}, p_{ny}) \end{pmatrix} \tag{A.5}$$

Auf Basis der Matrix K wird die SWZ gebildet (A.6). Durch die Zerlegung entstehen die drei Matrizen U, S und V^T [82]. Die gesuchte Rotation R, ergibt sich dabei aus dem Produkt der Transponierten von U und der Transponierten von V^T (A.7). Die Translation t kann dann aus dem Abstand des Flächenschwerpunkt q_0 zum rotierten Schwerpunkt $R \cdot p_0$ bestimmt werden (A.8).

$$\mathrm{SWZ}(K) = \mathrm{USV}^T \tag{A.6}$$

$$R = \mathrm{VU}^T \tag{A.7}$$

$$t = q_0 - R \cdot p_0 \tag{A.8}$$

Mithilfe der berechneten Werte für R und t kann anschließend die Punktwolke P transformiert werden (A.9) und der RMSE-Wert erneut bestimmt werden (A.10). Anhand von vorher bestimmten Konvergenzkriterien wird entschieden, ob der Algorithmus beendent oder erneut iteriert wird.

$$\bar{p_n} = R \cdot p_n + t \tag{A.9}$$

$$e^2 = \sum_{n=1}^{N} || \, q_n - \bar{p_n} \, ||^2 \tag{A.10}$$

A.2. Optimierung der PzF-Zielfunktion mittels Gauß-Newton-Verfahren

Da die Parameter ϕ, t_x und t_y gesucht werden, bilden sie zusammengefasst als Vektor x das Funktionsargument der Fehlerfunktion e^2. Durch Einbinden der gesuchten Parameter wird die Fehlerfunktion (3.12) entsprechend erweitert und nun mit E betitelt, siehe Formel (A.11). Für ein einzelnes Punktepaar wird die Funktion, wie in Formel (A.11) definiert gebildet.

$$\begin{aligned} E(\phi, t_x, t_y) &= \sum_{n=1}^{N} ((R(\phi) \cdot p_n + [t_x, t_y]^T - q_n) \cdot n_n)^2 \\ E(x) &= \sum_{n=1}^{N} ((\bar{p}_n(x) - q_n) \cdot n_n)^2 \\ E(x) &= \sum_{n=1}^{N} (e_n(x))^2 \end{aligned} \tag{A.11}$$

Da die Funktion (A.11) eine nichtlineare Funktion ist, wird das Gauß-Newton-Verfahren angewendet, welches die Funktion an der Stelle x linearisiert [82]. Dadurch kann das Problem als folgendes angenähertes Gleichungssystem (A.12) gelöst werden [43], [26], [82].

$$H \Delta x = -g \tag{A.12}$$

Δx definiert die angenäherte Änderung der Parameter in Richtung des Funktionsminimum. H ist die Hesse-Matrix der Zielfunktion E und g deren Gradient. H und g werden im Gauß-Newton Verfahren approximiert und nicht gemäß ihrer Definitionen berechnet. Dabei wird der Gradient g aus dem Produkt der Jacobi-Matrix J und der expliziten Funktion e_n, an der Stelle x bestimmt, siehe Formel (A.14). Die Hesse-Matrix H wird aus dem Produkt der transponierten Jacobi-Matrix mit sich selbst angenähert, siehe Formel (A.13). [82]

$$H = J^T(x) J(x) \tag{A.13}$$

$$g = J^T(x) e(x) \tag{A.14}$$

Die Jacobi-Matrix J beinhaltet die ersten partiellen Ableitungen von $e(x)$, nach den gesuchten Variablen. Da in der PzF-Metrik der Fehler $e(x)$ durch das Skalarprodukt mit dem Normalvektor n eindimensional wird, ist die Jacobi-Matrix eine 1x3 Matrix. Sie wird wie in Formel (A.15) beschrieben aufgebaut.

$$J_n(\phi, t_x, t_y) = \begin{bmatrix} \frac{\partial e_n}{\partial \phi} & \frac{\partial e_n}{\partial t_x} & \frac{\partial e_n}{\partial t_y} \end{bmatrix} \tag{A.15}$$
$$= \begin{bmatrix} n_x \cdot (-p_{nx}\sin(\phi) - p_{ny}\cos(\phi)) + n_{ny} \cdot (p_{nx}\cos(\phi) - p_{ny}\sin(\phi)) \\ n_{nx} \\ n_{ny} \end{bmatrix}^T$$

Mithilfe der Jacobi-Matrix J können die Hesse-Matrix H und der Gradient g gebildet und über alle Punktepaare q_n und p_n aufsummiert werden. Anschließend kann das Gleichungssystem aufgestellt und durch Umstellen nach Δx entsprechend gelöst werden. Dabei ergibt sich der neue Parametervektor x_{i+1} aus dem vorherigen Argument x_i und dem ermittelten Inkrement Δx. Das Verfahren wird mit den jeweils zuletzt bestimmten Parametern iteriert, bis ein Konvergenzkriterium erfüllt oder die maximale Anzahl an Iterationen erreicht wird.

$$H\Delta x = -g \rightarrow \Delta x = -H^{-1}g \tag{A.16}$$

$$x_{i+1} \leftarrow x_i + \Delta x \tag{A.17}$$

A.3. Herleitung des GICP-Algorithmus

Im wahrscheinlichkeitstheoretischen Modell von GICP wird angenommen, dass die gemessenen Punkte einer Normalverteilung von theoretischen Punkten entspringen. Die bereits erwähnten Punktwolken Q und P wären demnach ein Resultat der Normalverteilungen, der theoretischen Punktwolken $\hat{Q}$ und $\hat{P}$. Aufgrund dessen, dass es sich um eine multivariate Verteilung handelt, existieren mehrere Werte für die Varianz der Verteilung, weshalb die Kovarianzmatrizen C_n^Q und C_n^P benötigt werden, welche diese Parameter der Varianz σ^2 beinhalten. So ergibt sich für die Zusammensetzung der Punktwolken der folgende Zusammenhang.

$$q_n \sim \mathcal{N}(\hat{q}_n, C_n^Q) \qquad p_n \sim \mathcal{N}(\hat{p}_n, C_n^P) \tag{A.18}$$

Der Abstand zwischen den Punkten einer Paarung wird weiterhin, wie in der Zielfunktion der PzP-Metrik (3.11) über die euklidischen Abstände bestimmt und wird nach Formel (A.19) berechnet. Aufgrund dessen, dass die Punkte q_n und p_n jedoch aus unterschiedlichen Normalverteilungen stammen, unterliegt der Abstand zwischen ihnen der in Formel (A.20) beschriebenen, kombinierten Normalverteilung. Um eine Verwechslung zu vermeiden, wird dieser Abstand im Folgenden mit d_n betitelt. T^* beschreibt hier die Transformation T, welche den Punkt p_n fehlerfrei zu q_n transformiert, sodass der Abstand zwischen beiden Punkten null wird, siehe Formel (A.19).

$$d_n(T) = q_n - Tp_n \quad \rightarrow \quad \hat{q}_n = (T^*)\hat{p}_n \tag{A.19}$$

$$d_n(T^*) \sim \mathcal{N}(\hat{q}_n - (T^*)\hat{p}_n, \quad C_n^Q + (T^*)C_i^P(T^*)^T) \tag{A.20}$$

Aus den Normalverteilungen (A.18) und (A.20) leiten die Autoren von GICP die Summe (A.21) ab. Über Näherungsverfahren wird iterativ die Transformation T bestimmt, die diese Summe minimiert, analog zur Punkt-zu-Fläche Metrik.

$$T = \arg\min_T \sum_{n=1}^{n} {d_n}^T (C_n^Q + TC_n^P T^T)^{-1} d_n \tag{A.21}$$

Im Gegensatz zu den bisherigen Funktionen, ändert sich die Zielfunktion in Abhängigkeit der entstehenden Kovarianzmatrizen. Die PzP- und PzF-Metrik sind dadurch Spezialfälle der Summe, welche bei bestimmten Kovarianzmatrizen entstehen. Der eigentliche Vorteil von GICP liegt in der weiteren dritten Metrik, die sog. FzF-Metrik. Um ausgehend von der PzF-Metrik die Genauigkeit zu steigern, wird dieses Modell genutzt, um Informationen von beiden Punktwolken in den Registrierungsprozess miteinzubeziehen. Mit den bestehenden Annahmen, dass zweidimensionale Flächen im dreidimensionalen Raum gescannt werden und dass durch das Scannen aus zwei Per-

spektiven nicht die gleichen Punkte aufgezeichnet werden, wird ein neues Modell abgeleitet. Da die realen Flächen bzw. die Punktwolken lokal planar sind, wird angenommen, dass jeder Punkt mit hoher Kovarianz entlang seiner lokalen Fläche und mit sehr geringer Kovarianz entlang der Richtung seines Normalenvektors, verteilt werden kann. Dafür wird die folgende Einheitsmatrix (A.22) eingeführt.

$$\begin{pmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{A.22}$$

ϵ ist hierbei eine kleine Konstante, mit welcher die Kovarianz entlang der Flächennormalen festgelegt wird. Die Matrizen C_n^Q und C_n^P werden dann modelliert, indem die Einheitsmatrix (A.22) so rotiert wird, dass ϵ tatsächlich die Kovarianz entlang der Flächennormalen darstellt. Für die Punkte p_n und q_n, mit den zugehörigen Normalvektoren ν_n und μ_n werden die finalen Matrizen entsprechend den Formeln (A.23) und (A.24) gebildet.

$$C_n^P = R_{\mu_n} \cdot \begin{pmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot {R_{\mu_n}}^T \tag{A.23}$$

$$C_n^Q = R_{\nu_n} \cdot \begin{pmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot {R_{\nu_n}}^T \tag{A.24}$$

B. Tabellen und Diagramme der Auswertung

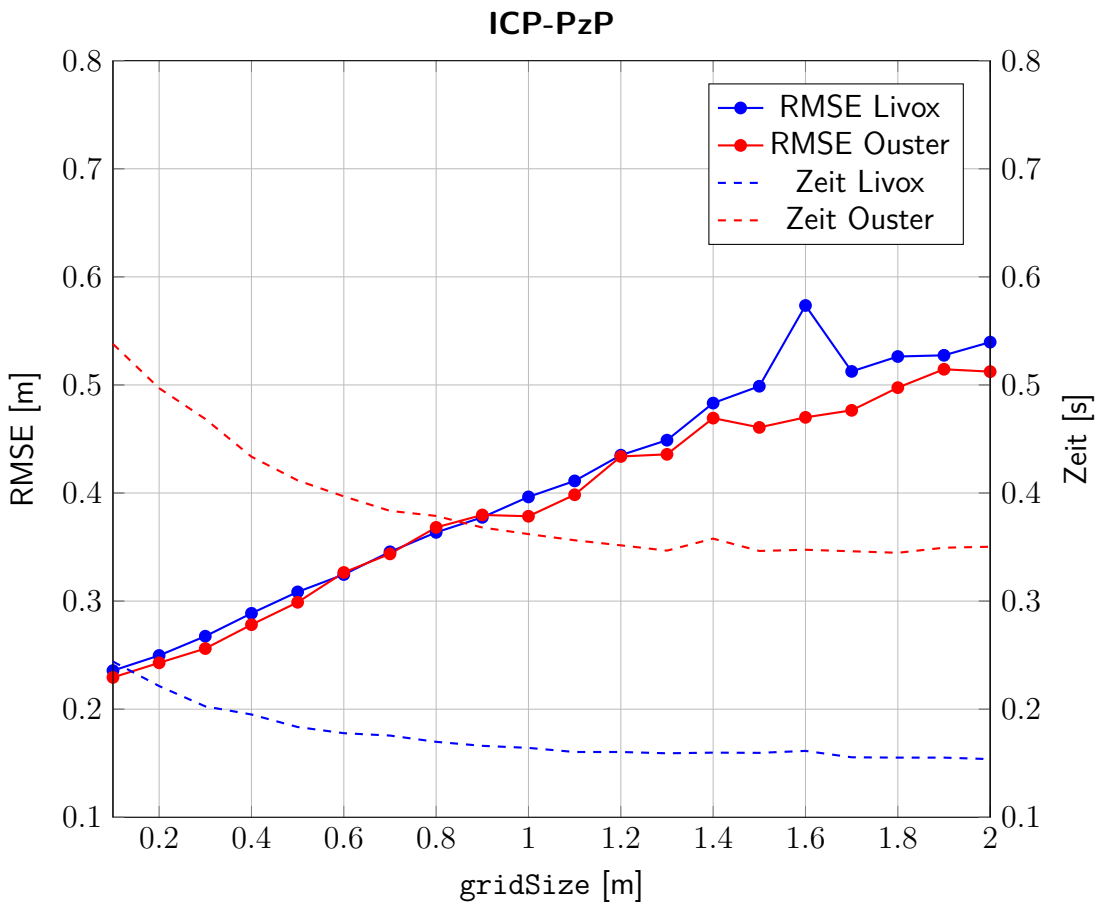
B.1. Analyse des `gridSize`-Parameters

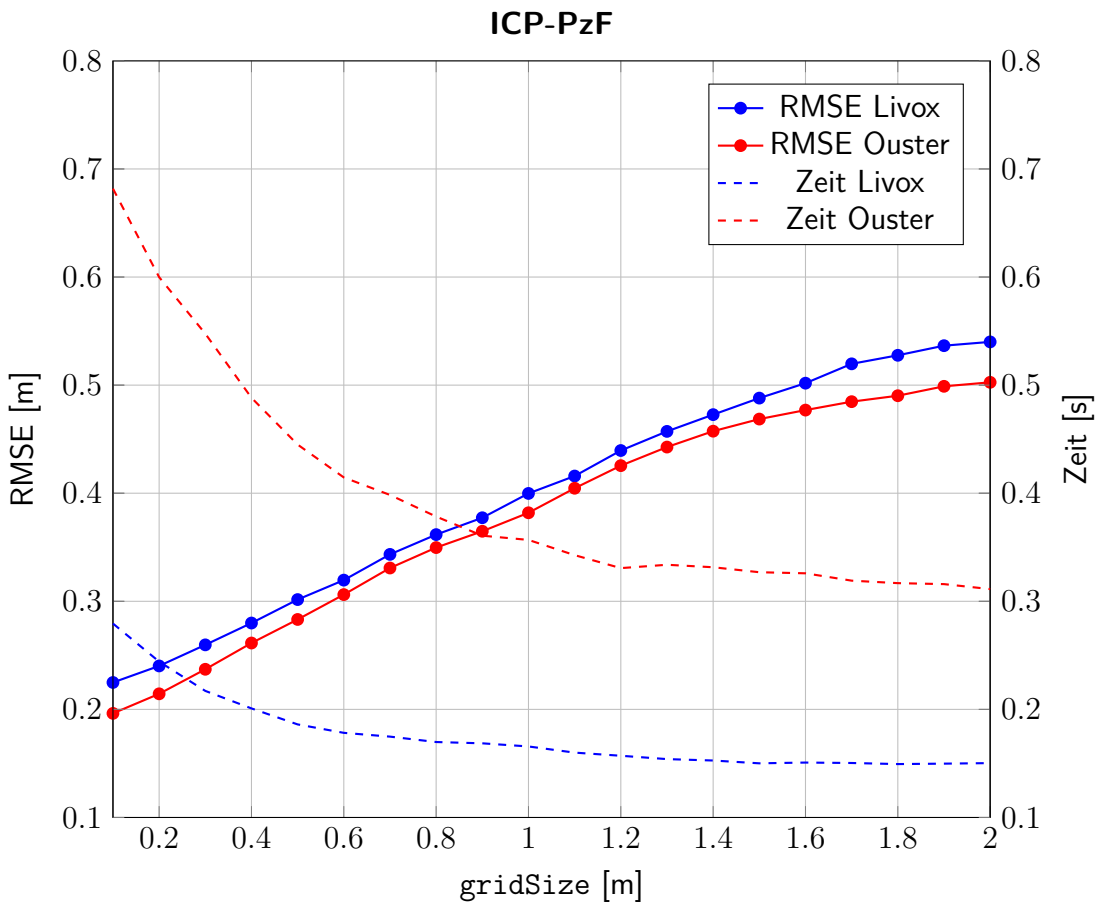
LIVOX Horizon								
gridSize [m]	**ICP-PzP**		**ICP-PzF**		**GICP**		**NDT**	
	Zeit [s]	RMSE [m]	Zeit [s]	RMSE [m]	Zeit [s]	RMSE [m]	Zeit [s]	RMSE [m]
0,1	0,2441	0,2357	0,2794	0,2249	0,3308	0,2251	0,4030	1,5727
0,2	0,2214	0,2497	0,2442	0,2402	0,2677	0,2410	0,3313	1,3952
0,3	0,2026	0,2675	0,2171	0,2597	0,2300	0,2606	0,2744	1,4890
0,4	0,1951	0,2887	0,2009	0,2799	0,2055	0,2810	0,2538	1,4577
0,5	0,1835	0,3085	0,1861	0,3016	0,1891	0,3025	0,2275	1,5890
0,6	0,1778	0,3246	0,1783	0,3196	0,1766	0,3212	0,2128	1,5653
0,7	0,1756	0,3456	0,1748	0,3434	0,1703	0,3445	0,2076	1,5246
0,8	0,1698	0,3636	0,1698	0,3617	0,1650	0,3635	0,1959	1,5158
0,9	0,1661	0,3775	0,1686	0,3773	0,1574	0,3794	0,1914	1,7226
1	0,1642	0,3964	0,1657	0,3998	0,1552	0,4008	0,1857	2,0481
1,1	0,1604	0,4112	0,1600	0,4159	0,1526	0,4186	0,1809	1,6654
1,2	0,1604	0,4350	0,1571	0,4395	0,1499	0,4413		1,6584
1,3	0,1592	0,4489	0,1540	0,4572	0,1484	0,4610		
1,4	0,1598	0,4832	0,1527	0,4727	0,1468	0,4738		
1,5	0,1596	0,4988	0,1501	0,4880	0,1468	0,4915		
1,6	0,1614	0,5735	0,1508	0,5018	0,1436	0,5051		
1,7	0,1555	0,5125	0,1504	0,5197	0,1438	0,5218		
1,8	0,1552	0,5263	0,1494	0,5276	0,1426	0,5560		
1,9	0,1552	0,5274	0,1498	0,5365	0,1415	0,5397		
2	0,1538	0,5396	0,1503	0,5400	0,1417	0,5615		

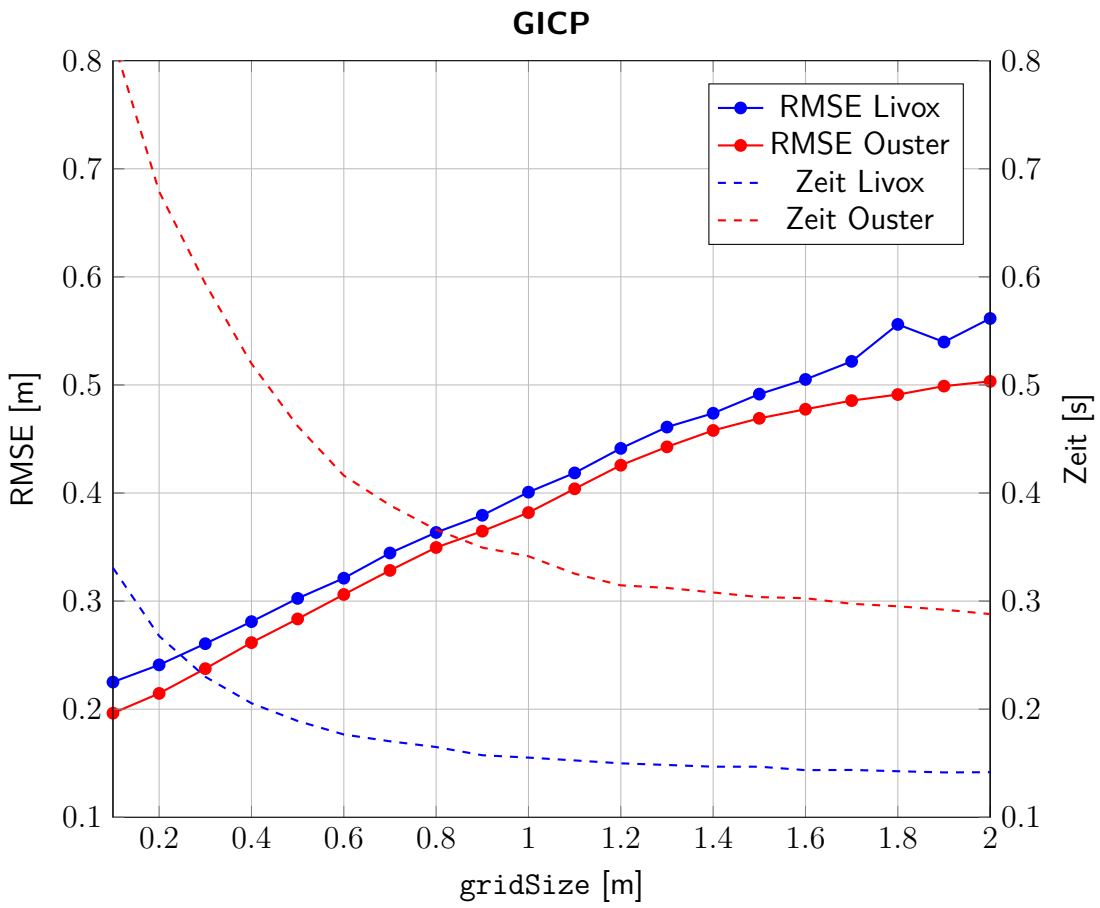
Tabelle B.1.: `gridSize`-Ergebnisse des Livox Horizon

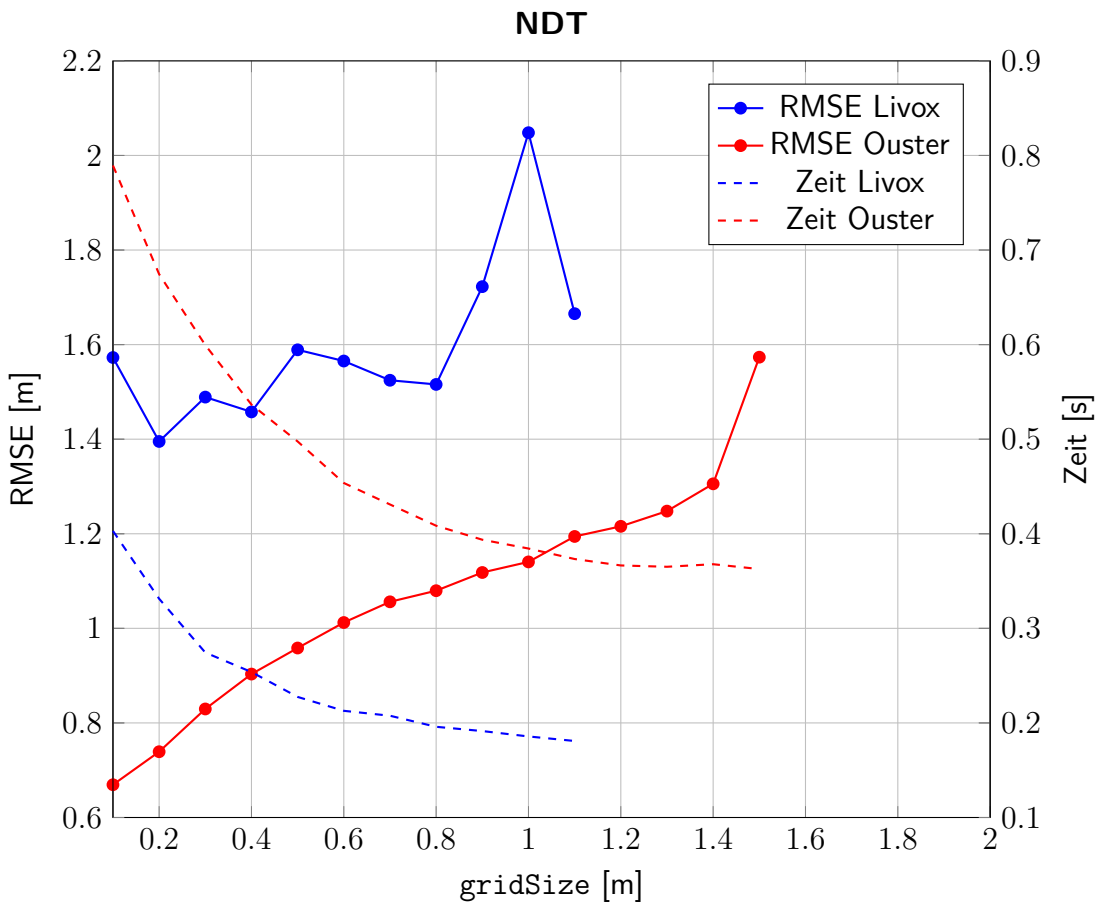
Ouster OS1								
gridSize [m]	**ICP-PzP**		**ICP-PzF**		**GICP**		**NDT**	
	Zeit [s]	RMSE [m]	Zeit [s]	RMSE [m]	Zeit [s]	RMSE [m]	Zeit [s]	RMSE [m]
0,1	0,5379	0,2294	0,6818	0,1963	0,8196	0,1963	0,7894	0,6693
0,2	0,4970	0,2429	0,5999	0,2144	0,6789	0,2146	0,6745	0,7391
0,3	0,4685	0,2561	0,5476	0,2371	0,5937	0,2375	0,5995	0,8295
0,4	0,4336	0,2782	0,4883	0,2614	0,5197	0,2616	0,5366	0,9033
0,5	0,4117	0,2989	0,4450	0,2832	0,4619	0,2835	0,4976	0,9584
0,6	0,3969	0,3265	0,4146	0,3062	0,4162	0,3061	0,4537	1,0123
0,7	0,3835	0,3437	0,3984	0,3308	0,3887	0,3284	0,4310	1,0561
0,8	0,3788	0,3682	0,3784	0,3497	0,3661	0,3496	0,4085	1,0796
0,9	0,3681	0,3797	0,3607	0,3648	0,3495	0,3647	0,3937	1,1181
1	0,3620	0,3785	0,3568	0,3819	0,3415	0,3819	0,3844	1,1405
1,1	0,3563	0,3984	0,3427	0,4046	0,3254	0,4039	0,3733	1,1943
1,2	0,3516	0,4338	0,3307	0,4255	0,3146	0,4257	0,3665	1,2157
1,3	0,3467	0,4358	0,3338	0,4427	0,3122	0,4427	0,3651	1,2479
1,4	0,3578	0,4693	0,3315	0,4574	0,3080	0,4579	0,3678	1,3056
1,5	0,3464	0,4607	0,3269	0,4686	0,3037	0,4691	0,3627	1,5734
1,6	0,3475	0,4700	0,3259	0,4769	0,3027	0,4775		4,9552
1,7	0,3461	0,4765	0,3191	0,4847	0,2975	0,4855		
1,8	0,3447	0,4975	0,3168	0,4902	0,2952	0,4911		
1,9	0,3494	0,5145	0,3159	0,4989	0,2921	0,4989		
2	0,3503	0,5123	0,3113	0,5026	0,2880	0,5033		

Tabelle B.2.: `gridSize`-Ergebnisse des Ouster OS1

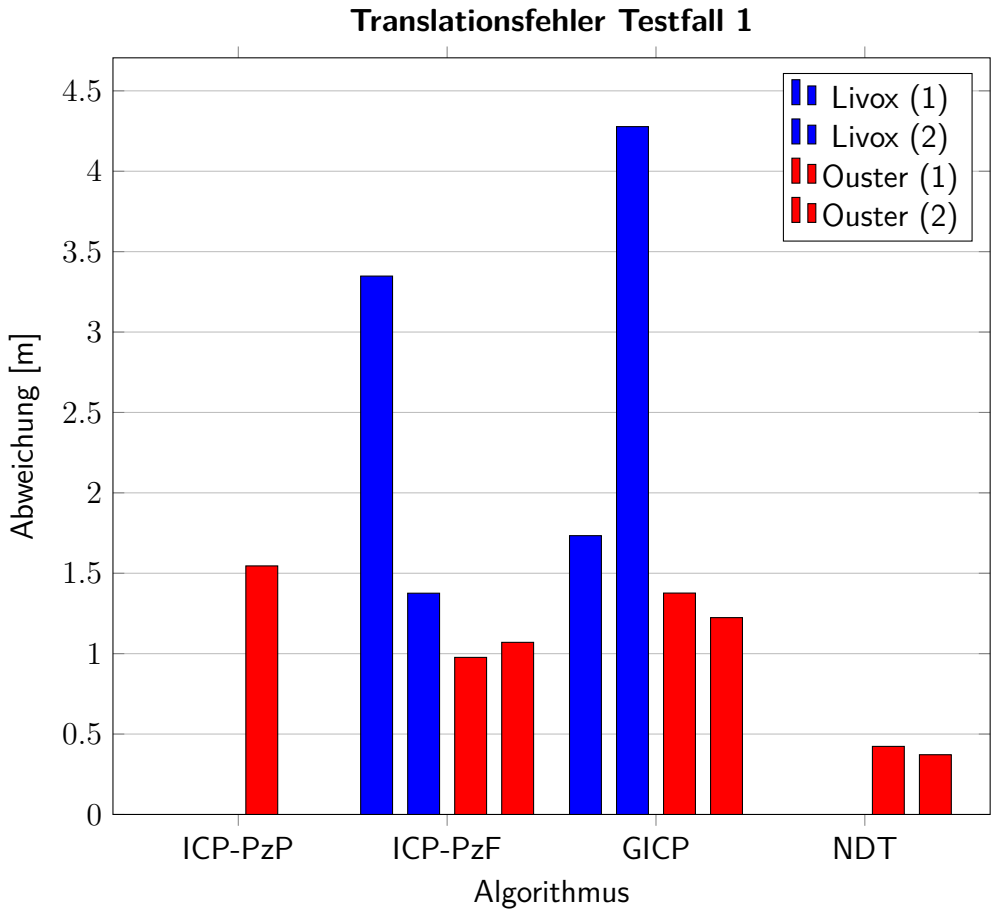

ICP-PzP
RMSE Livox
RMSE Ouster
Zeit Livox
Zeit Ouster
RMSE [m]
Zeit [s]
gridSize [m]
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

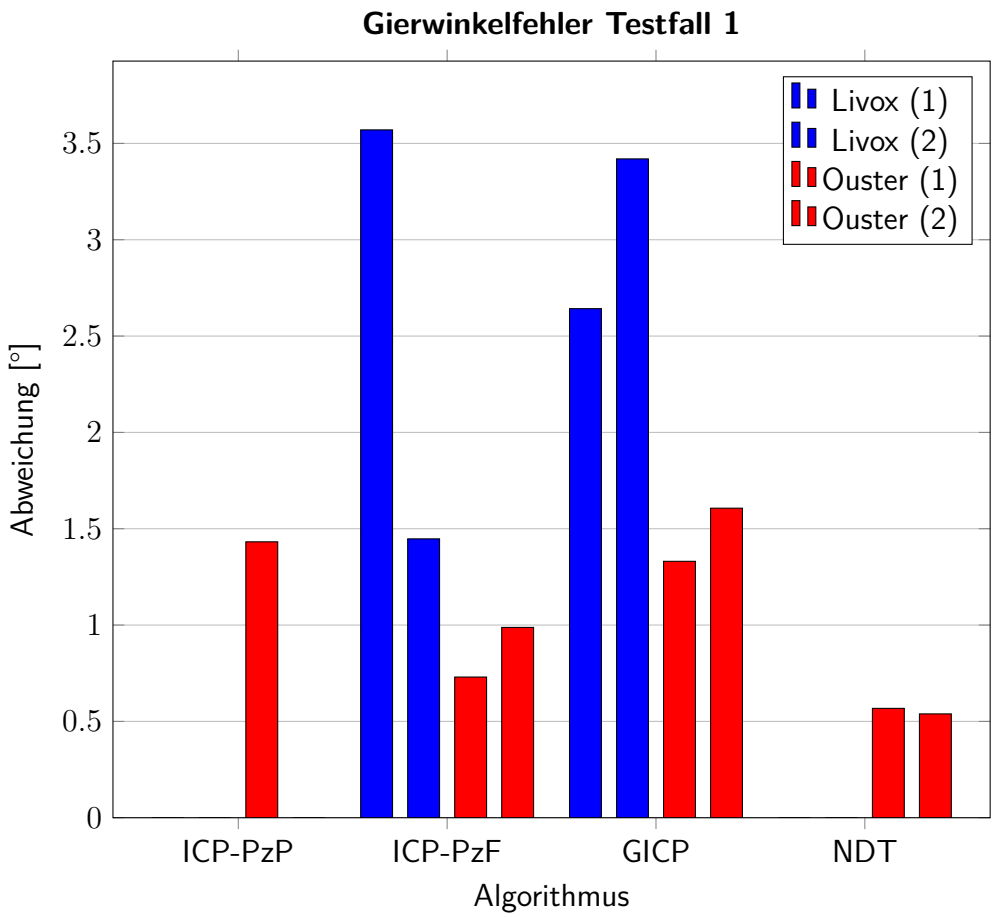

ICP-PzF
RMSE Livox
RMSE Ouster
Zeit Livox
Zeit Ouster
RMSE [m]
Zeit [s]
gridSize [m]
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

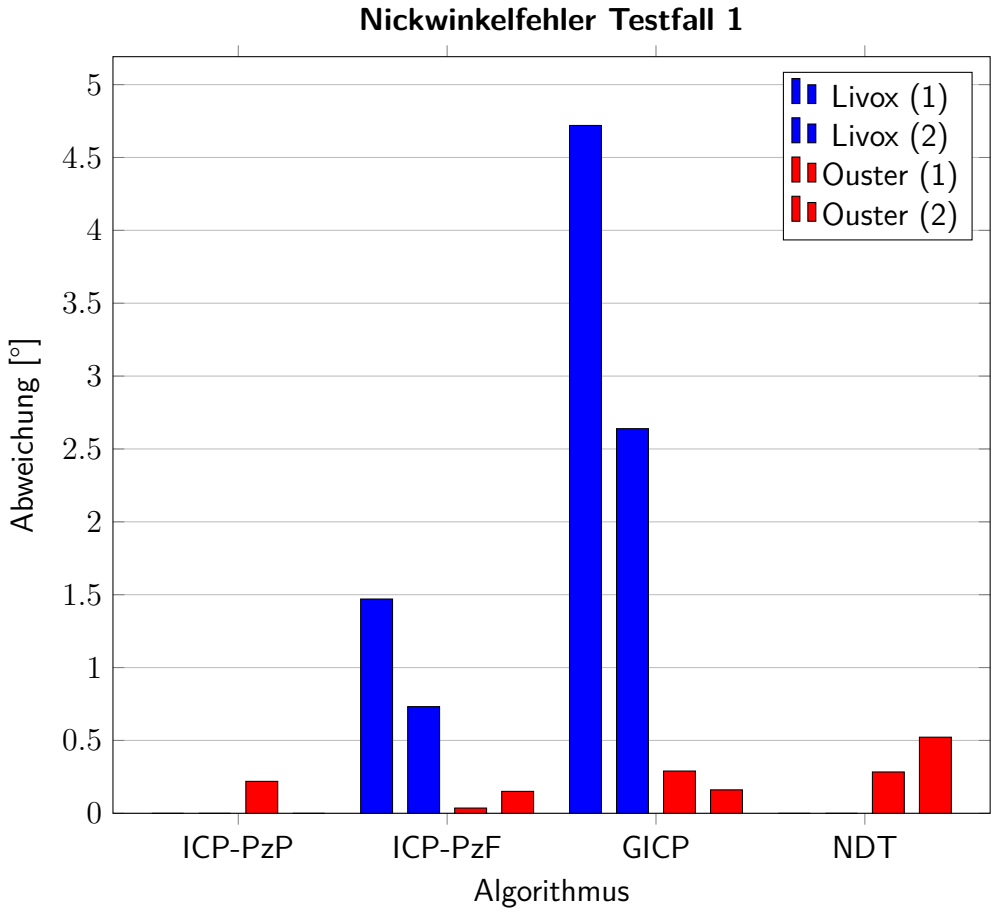

GICP
RMSE Livox
RMSE Ouster
Zeit Livox
Zeit Ouster
RMSE [m]
Zeit [s]
gridSize [m]
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

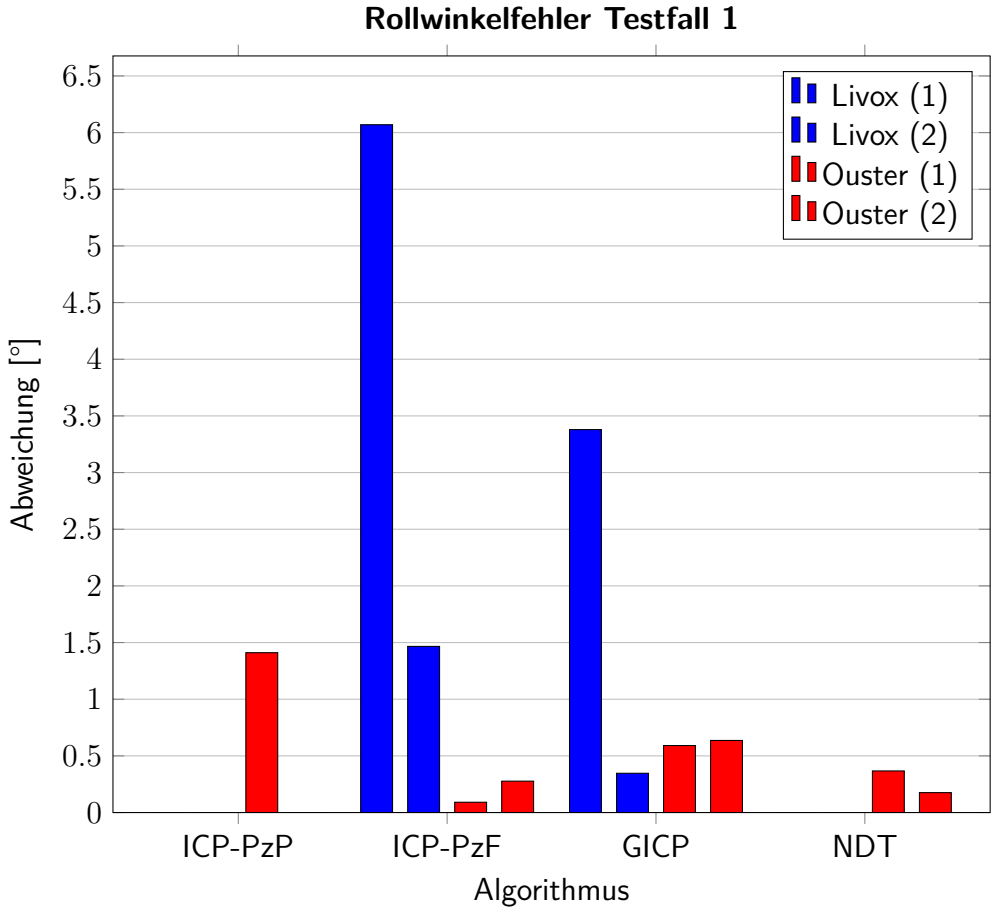

NDT
RMSE Livox
RMSE Ouster
Zeit Livox
Zeit Ouster
RMSE [m]
Zeit [s]
gridSize [m]
0.6
0.8
1
1.2
1.4
1.6
1.8
2
2.2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

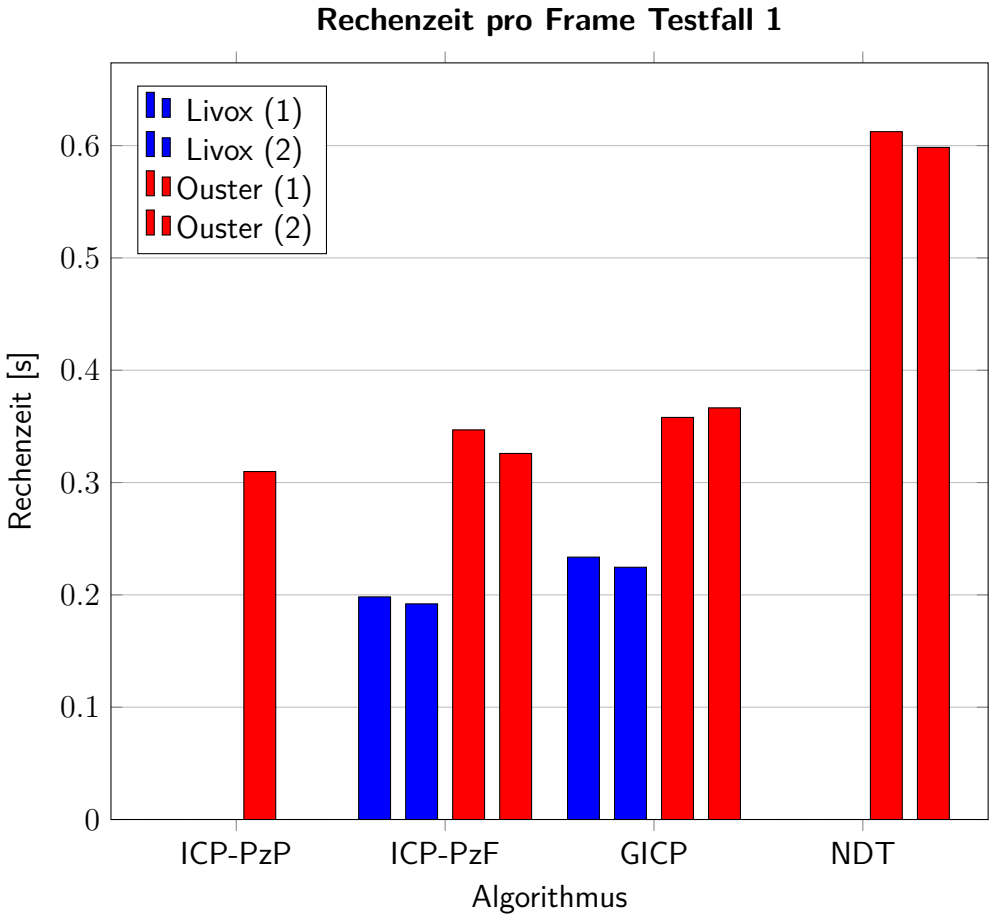
B.2. Testfall 1






Nickwinkelfehler Testfall 1
Livox (1)
Livox (2)
Ouster (1)
Ouster (2)
Abweichung [°]
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
ICP-PzP
ICP-PzF
GICP
NDT
Algorithmus

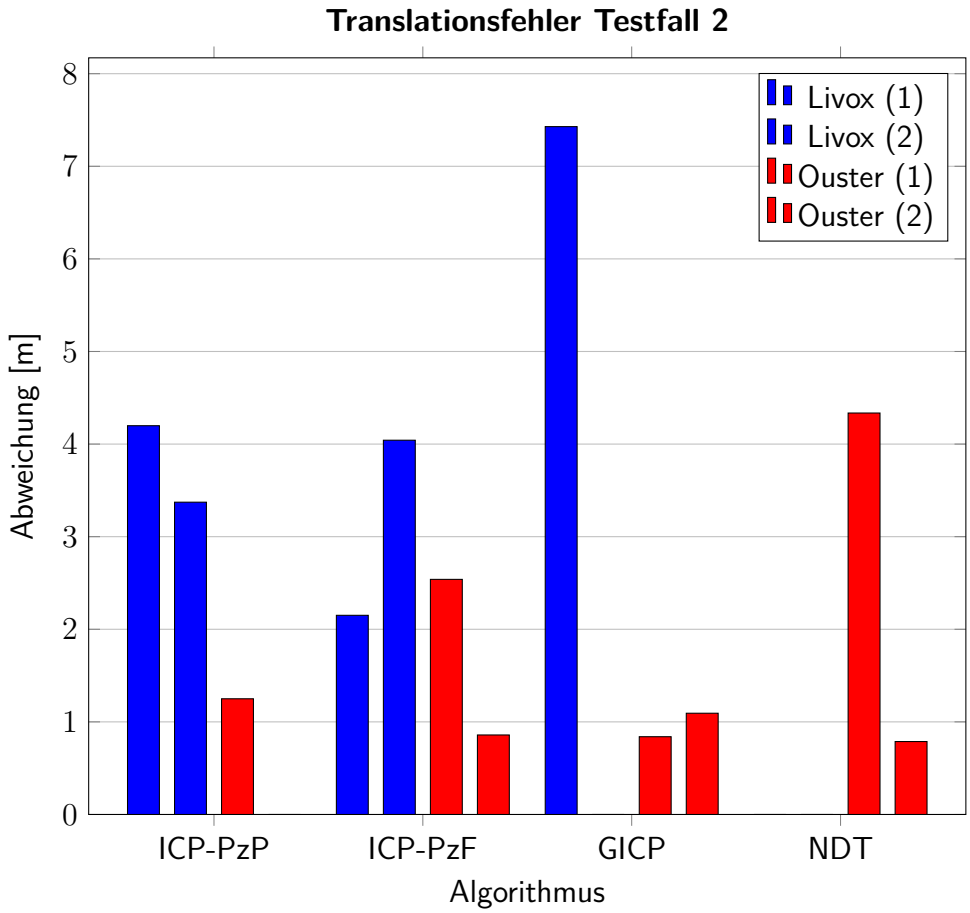

Rollwinkelfehler Testfall 1
Livox (1)
Livox (2)
Ouster (1)
Ouster (2)
Abweichung [°]
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6
6.5
ICP-PzP
ICP-PzF
GICP
NDT
Algorithmus

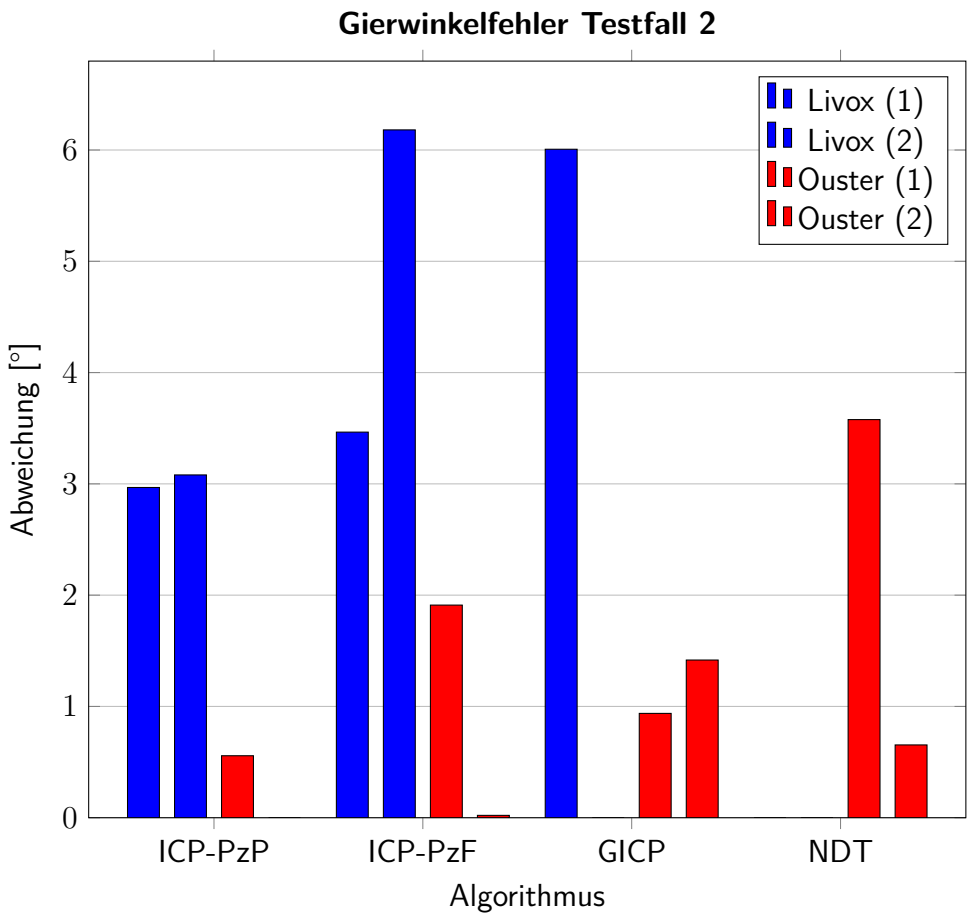

Rechenzeit pro Frame Testfall 1
Livox (1)
Livox (2)
Ouster (1)
Ouster (2)
Rechenzeit [s]
0
0.1
0.2
0.3
0.4
0.5
0.6
ICP-PzP
ICP-PzF
GICP
NDT
Algorithmus

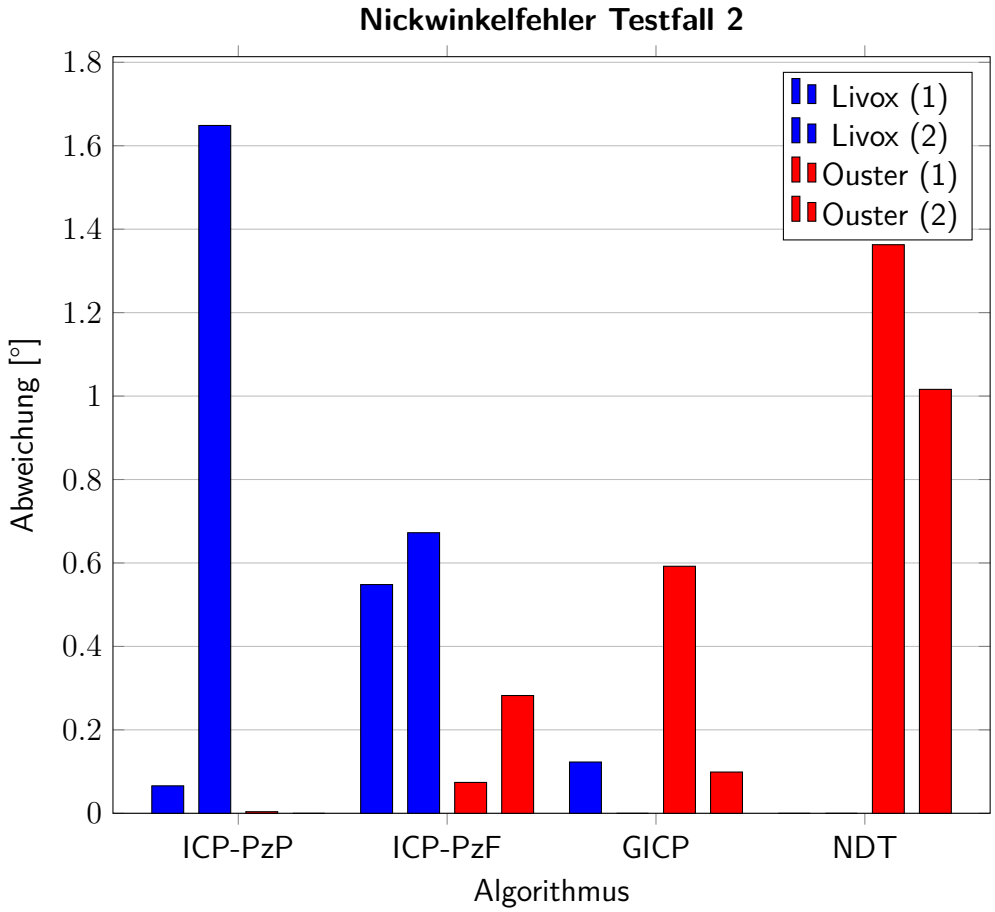
Testfall 1								
Algorithmus	**Sensor**	**Messung**	**RMSE [m]**	**Rechenzeit [s]**	**Absoluter Fehler**			
					Translation (x, y, z) [m]	Gierwinkel [°]	Nickwinkel [°]	Rollwinkel [°]
ICP-PzP	Livox	1	-	-	-	-	-	-
		2	-	-	-	-	-	-
	Ouster	1	0,3178	0,3098	1,2197; 0,9485; 0,0447	1,4320	0,2192	1,4106
		2	-	-	-	-	-	-
ICP-PzF	Livox	1	0,3214	0,1982	1,4222; 1,0401; 2,8470	3,5702	1,4703	6,0695
		2	0,3251	0,1920	0,9280; 0,4464; 0,9127	1,4474	0,7320	1,4662
	Ouster	1	0,3491	0,3469	0,4366; 0,5678; 0,6641	0,7302	0,0360	0,0910
		2	0,3462	0,3259	0,5963; 0,5895; 0,6654	0,9878	0,1505	0,2770
GICP	Livox	1	0,2805	0,2336	1,2510; 0,8661; 0,8313	2,6423	4,7197	3,3792
		2	0,2825	0,2246	0,9891; 0,4836; 4,1334	3,4196	2,6392	0,3466
	Ouster	1	0,3491	0,3580	0,6435; 1,0920; 0,5377	1,3309	0,2900	0,5911
		2	0,3458	0,3665	0,7772; 0,9104; 0,2560	1,6066	0,1609	0,6363
NDT	Livox	1	-	-	-	-	-	-
		2	-	-	-	-	-	-
	Ouster	1	0,6693	0,6124	0,1387; 0,3726; 0,1453	0,5676	0,2836	0,3672
		2	0,8627	0,5984	0,0216; 0,3694; 0,0326	0,5389	0,5223	0,1758

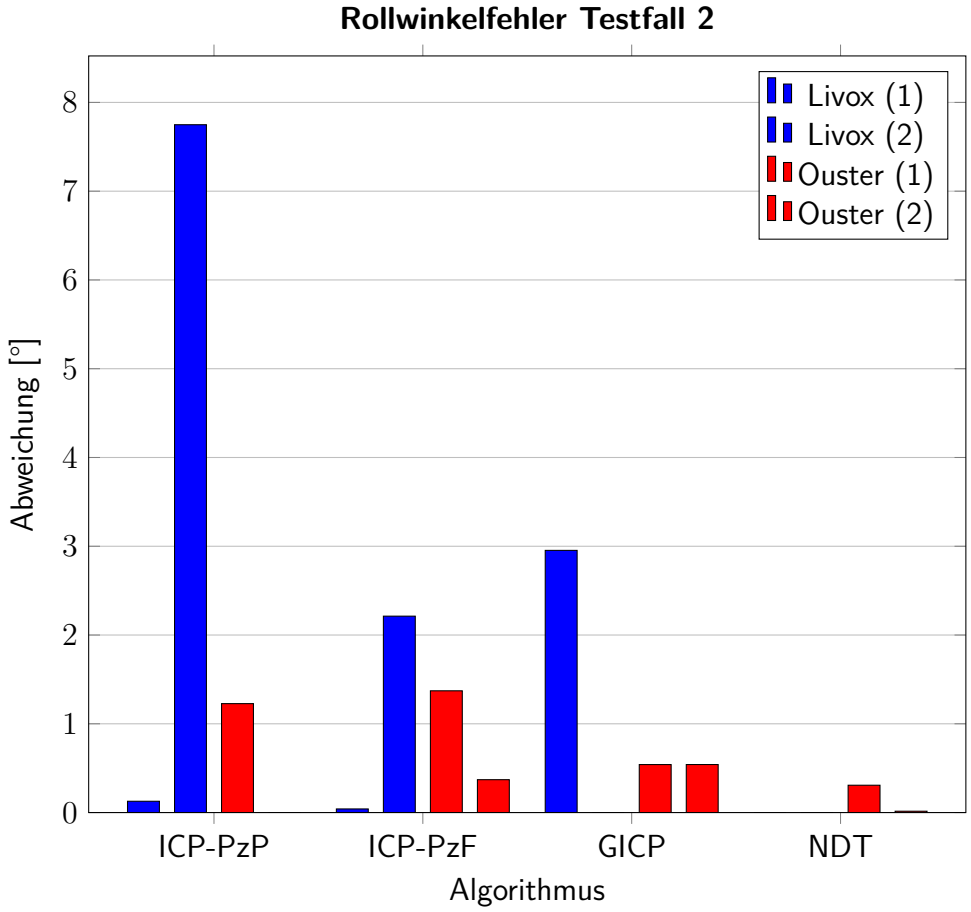
Tabelle B.3.: Ermittelte Fehlerwerte des ersten Testfalls

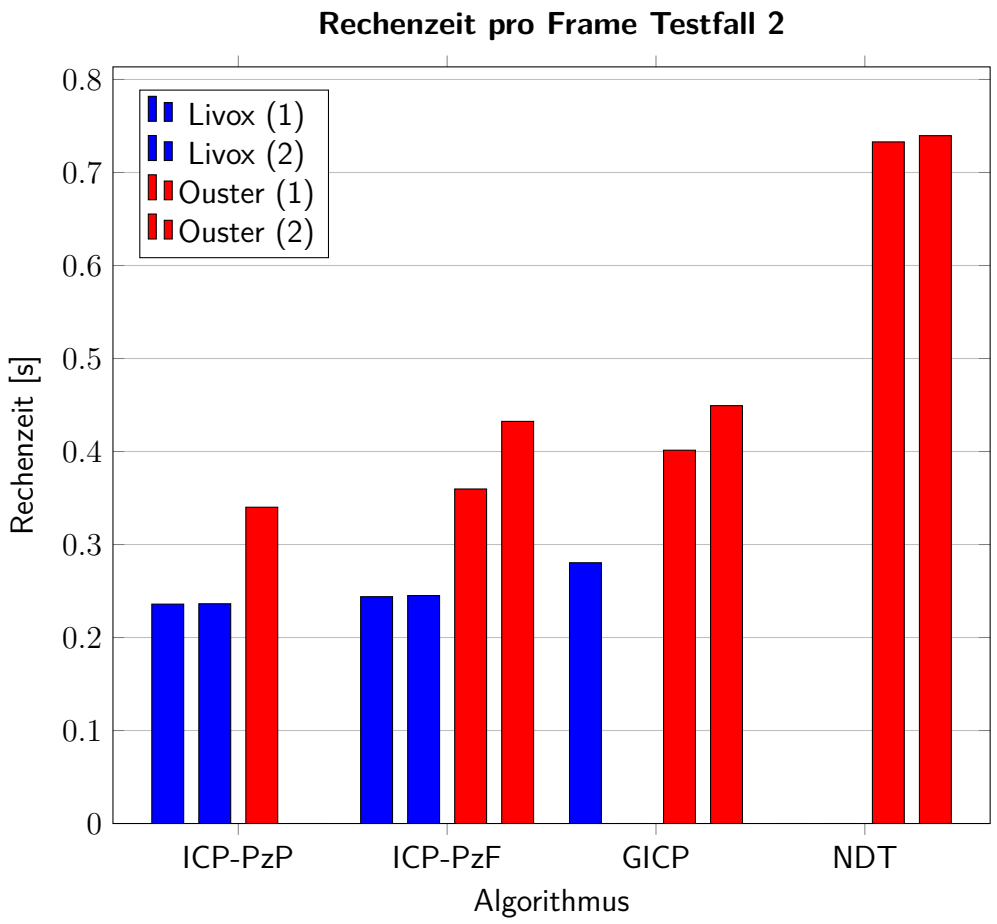
B.3. Testfall 2






Nickwinkelfehler Testfall 2
Livox (1)
Livox (2)
Ouster (1)
Ouster (2)
Abweichung [°]
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
ICP-PzP
ICP-PzF
GICP
NDT
Algorithmus

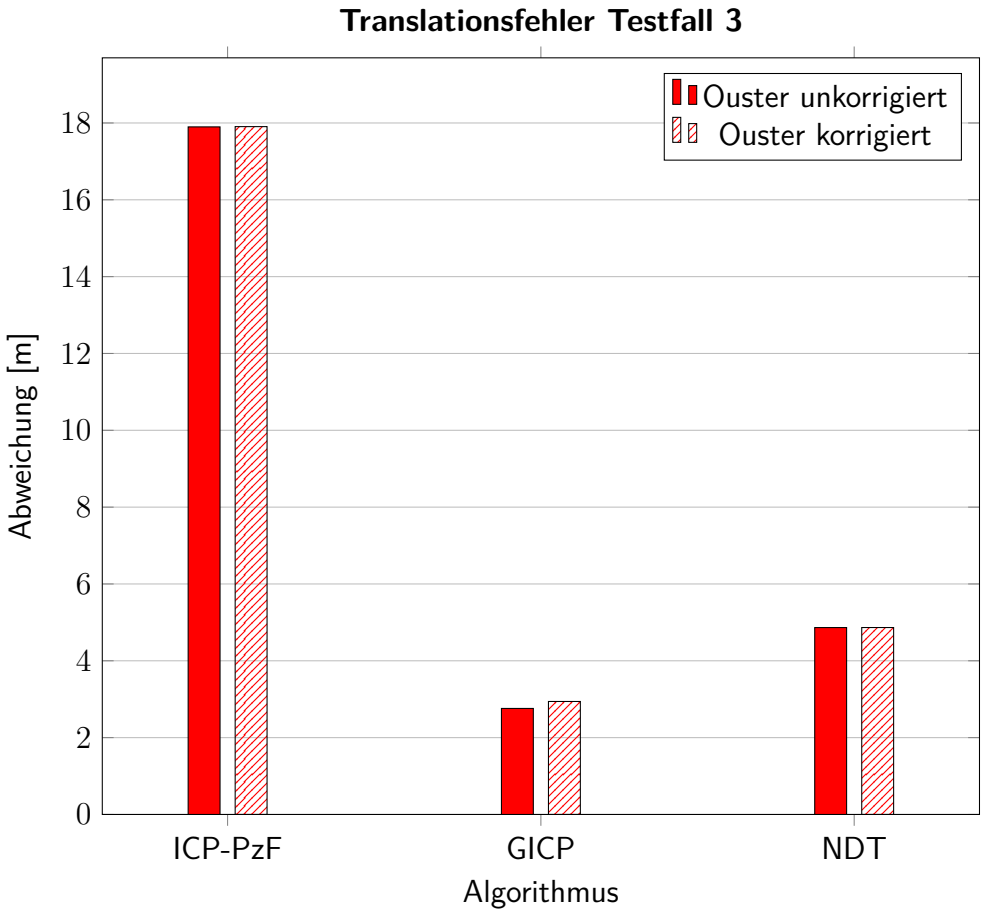

Rollwinkelfehler Testfall 2
Livox (1)
Livox (2)
Ouster (1)
Ouster (2)
Abweichung [°]
0
1
2
3
4
5
6
7
8
ICP-PzP
ICP-PzF
GICP
NDT
Algorithmus

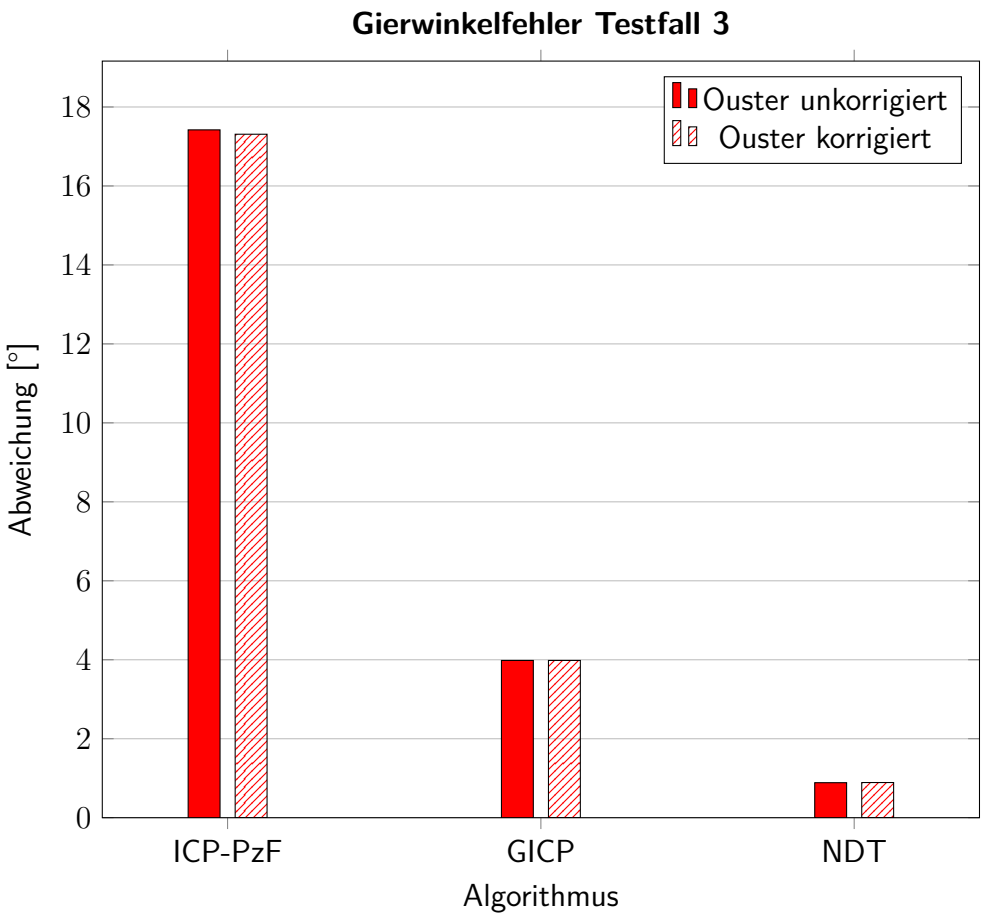

Rechenzeit pro Frame Testfall 2
Livox (1)
Livox (2)
Ouster (1)
Ouster (2)
Rechenzeit [s]
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
ICP-PzP
ICP-PzF
GICP
NDT
Algorithmus

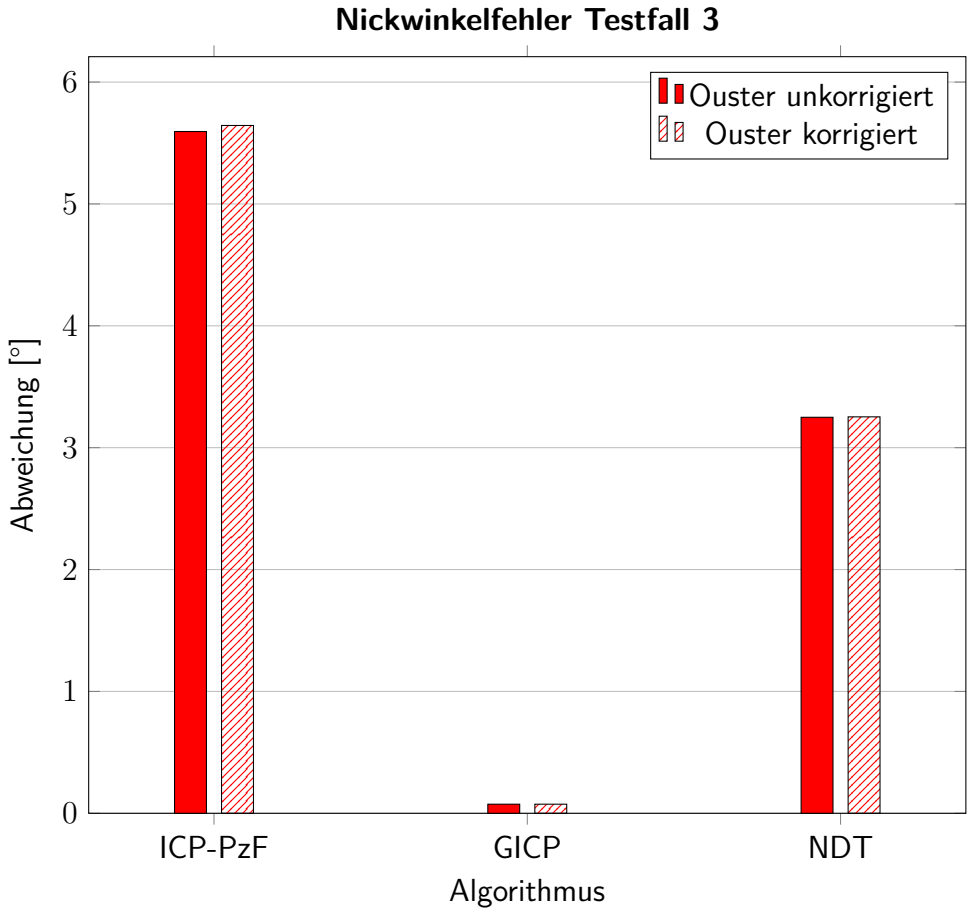
Testfall 2								
Algorithmus	**Sensor**	**Messung**	**RMSE [m]**	**Rechenzeit [s]**	**Absoluter Fehler**			
					Translation (x, y, z) [m]	Gierwinkel [°]	Nickwinkel [°]	Rollwinkel [°]
ICP - PzP	Livox	1	0,3257	0,2359	1,1260; 0,5551; 4,0064	2,9675	0,0658	0,1274
		2	0,3261	0,2363	1,6380; 0,7161; 2,8598	3,0803	1,6486	7,7487
	Ouster	1	0,3226	0,3401	1,1987; 0,2115; 0,2824	0,5570	0,0038	1,2272
		2	-	-	-	-	-	-
ICP - PzF	Livox	1	0,3278	0,2439	1,8202; 0,1281; 1,1387	3,4650	0,5482	0,0408
		2	0,3277	0,2451	3,4802; 0,6588; 1,9468	6,1808	0,6725	2,2126
	Ouster	1	0,3682	0,3597	2,2227; 1,2152; 0,1715	1,9107	0,0740	1,3716
		2	0,3646	0,4324	0,2023; 0,1568; 0,8196	0,0212	0,2823	0,3704
GICP	Livox	1	0,2862	0,2804	3,5895; 1,5036; 6,3274	6,0069	0,1229	2,9536
		2	-	-	-	-	-	-
	Ouster	1	0,3664	0,4014	0,5215; 0,5456; 0,4787	0,9375	0,5921	0,5407
		2	0,3647	0,4493	0,7790; 0,5825; 0,4994	1,4170	0,0989	0,5411
NDT	Livox	1	-	-	-	-	-	-
		2	-	-	-	-	-	-
	Ouster	1	0,7152	0,7329	3,6725; 1,8041; 1,4334	3,5775	1,3628	0,3080
		2	0,6526	0,7396	0,7767; 0,0559; 0,1183	0,6541	1,0161	0,0152

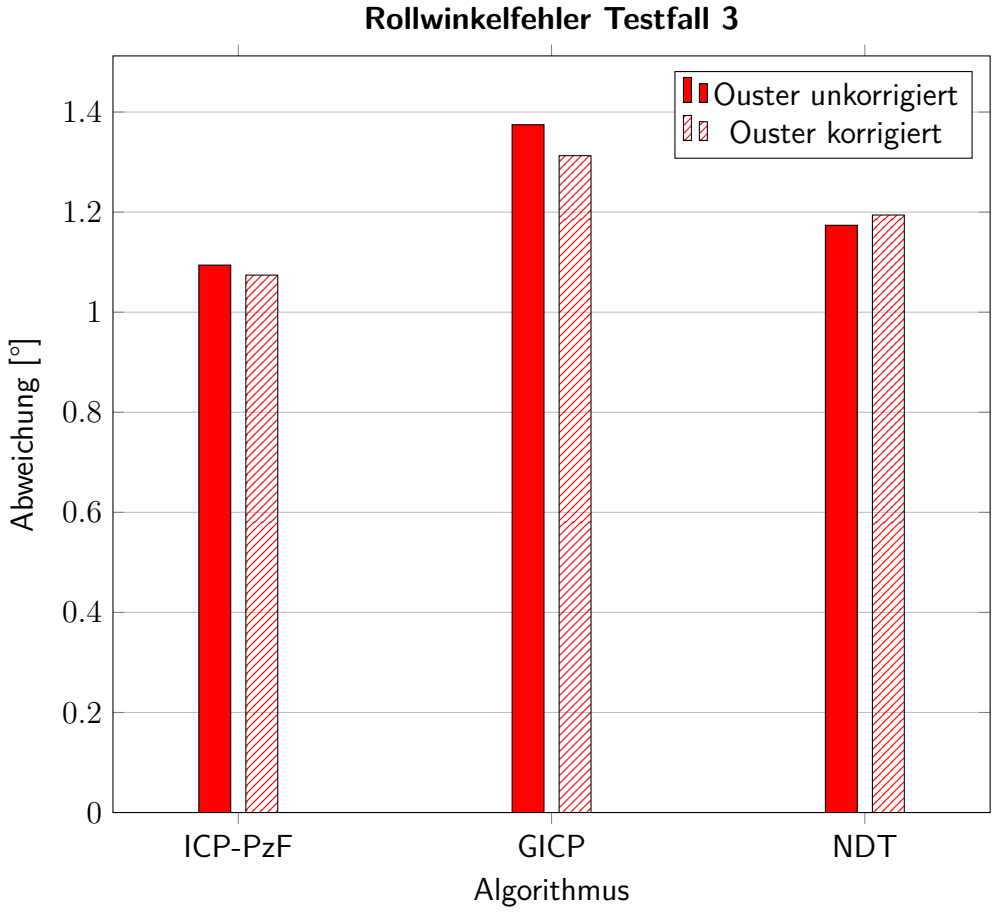
Tabelle B.4.: Ermittelte Fehlerwerte des zweiten Testfalls

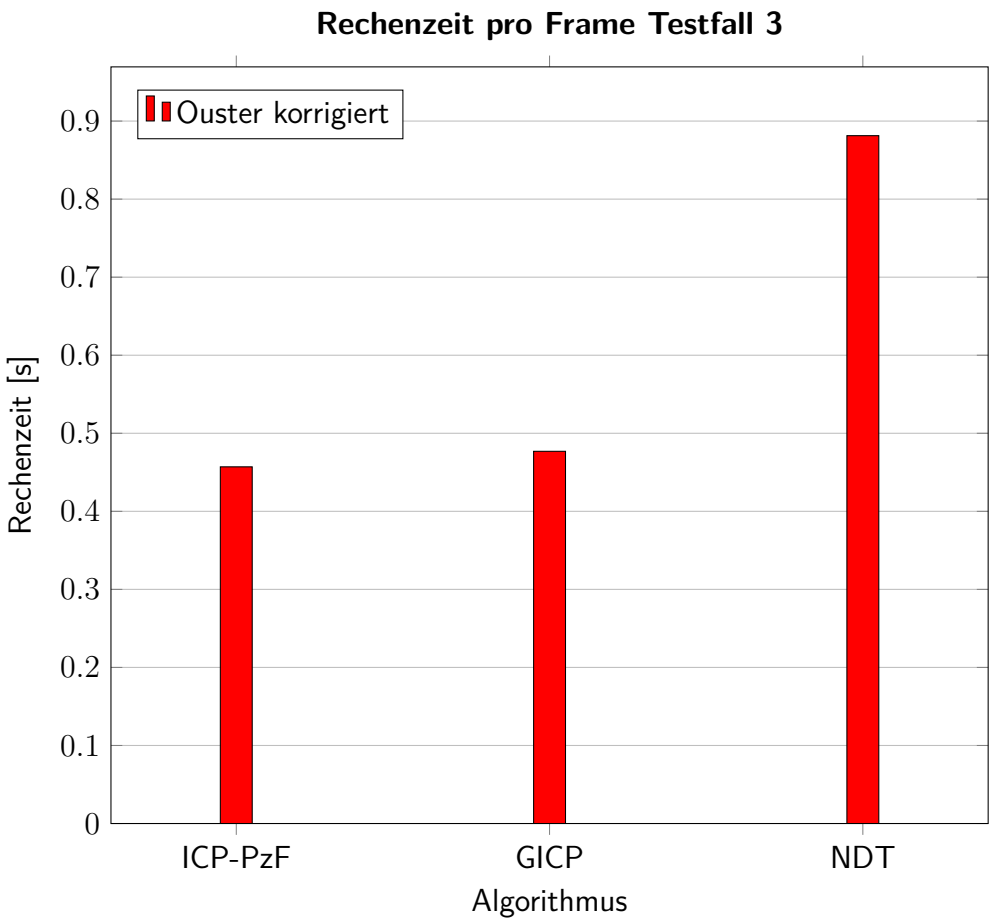
B.4. Testfall 3






Nickwinkelfehler Testfall 3
Ouster unkorrigiert
Ouster korrigiert
Abweichung [°]
0
1
2
3
4
5
6
ICP-PzF
GICP
NDT
Algorithmus


Rollwinkelfehler Testfall 3
Ouster unkorrigiert
Ouster korrigiert
Abweichung [°]
0
0.2
0.4
0.6
0.8
1
1.2
1.4
ICP-PzF
GICP
NDT
Algorithmus


Rechenzeit pro Frame Testfall 3
Ouster korrigiert
Rechenzeit [s]
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
ICP-PzF
GICP
NDT
Algorithmus

Testfall 3 (Ouster)							
Algorithmus	**Korrektur**	**RMSE [m]**	**Rechenzeit [s]**	**Absoluter Fehler**			
				Translation (x y z) [m]	Gierwinkel [°]	Nickwinkel [°]	Rollwinkel [°]
ICP - PzP	Nein	-	-	-	-	-	-
	Ja	-	-	-	-	-	-
ICP - PzF	Nein	0,3937	0,4570	17,2572; 4,7344; 0,3578	17,420	5,5943	1,0941
	Ja	-	-	17,2431; 4,8136; 0,3200	17,310	5,6439	1,0741
GICP	Nein	0,3872	0,4769	1,2659; 1,9787; 1,4487	3,9825	0,0747	1,3747
	Ja	-	-	1,6258; 1,9786; 1,4463	3,9824	0,0747	1,3128
NDT	Nein	1,0456	0,8813	2,3802; 3,6943; 2,0868	0,8867	3,2494	1,1737
	Ja	-	-	2,3799; 3,6953, 2,0870	0,8905	3,2530	1,1942

Tabelle B.5.: Ermittelte Fehlerwerte des dritten Testfalls

C. Datenträger

```
USB-Stick
├ LaTeX
├ DA.pdf
├ Skripts
│  ├ Kartierung
│  ├ Livox_Konvertierung
│  ├ Ouster_Aufzeichnen
│  ├ Ermittlung_gridSize
│  └ Fehlerermittlung
├ Messaufzeichnungen
│  ├ Testfall_1
│  ├ Testfall_2
│  └ Testfall_3
├ Kartierungsergebnisse
│  ├ Testfall_1
│  ├ Testfall_2
│  └ Testfall_3
├ GPS
│  ├ OSM_Files
│  └ GPX_Files
└ Excel_Auswertungen
```