Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Tobias Vater

Inline-Auswertung und -Korrektur multipler, komplexer, fusionierter, bildgebender Sensoren für das automatisierte Fahren

Fakultät Technik und Informatik *Faculty of Engineering and Computer Science*

Tobias Vater

Inline-Auswertung und -Korrektur multipler, komplexer, fusionierter, bildgebender Sensoren für das automatisierte Fahren

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Mechatronik
an der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Rasums Rettig
Zweitgutachter : Dr. Ralf Wendel

Abgegeben am 11. April 2018

Tobias Vater

Thema der Bachelorthesis

Inline-Auswertung und -Korrektur multipler, komplexer, fusionierter, bildgebender Sensoren für das automatisierte Fahren

Stichworte

Autonomes Fahren, Sensorik, Kartografie, SLAM, ROS, Lidar, Kamerasysteme

Kurzzusammenfassung

Diese Arbeit umfasst die Analyse, Design und Realisierung eines fahrzeugintegrierten Systems für die Auswertung von Sensordaten während der Fahrt. Die Auswertung realisiert eine Positionsbestimmungs- und Kartografierungs-Lösung.

Tobias Vater

Title of the paper

Inline-evaluation and -correction of multiple, complex and merged sensors based on imaging methods for autonomous driving

Keywords

Autonomous driving, sensors, mapping, SLAM, ROS, Lidar, imaging

Abstract

Inside this report is the analyses, design and implemention of a vehicle-integrated system described that evaluates sensor data and focused on autonomous driving. The evaluation is implemented as a positioning and mapping solution.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

1.1. Motivation

Die Entwicklung des automatisierten Fahrens hat in den letzten Jahren stark an Relevanz gewonnen. Alle OEMs und Zulieferer befassen sich mit diesem Thema. Unter ihnen befinden sich zum Beispiel Google, Audi, BMW und Daimler, aber auch der Automobilzulieferer Bosch. Das Ziel ist eine Zukunft, in welcher dem Autofahrer das konzentrierte Fahren komplett abgenommen wird und der vollständig autonome Betrieb von Fahrzeugen gemäß SAE-Level 5 stattfindet (vgl. Maurer u. a., 2015, S. 201 f.).

Maßgeblich angetrieben wird die Entwicklung durch die hohen Unfallzahlen aufgrund menschlichen Versagens. Zwar hat sich die Anzahl er tödlichen Verkehrsunfälle in den letzten Jahren immer weiter reduziert, jedoch sind 2016 immer noch mehr als 3000 Menschen bei einem Unfall ums leben gekommen (vgl. Statistisches Bundesamt). Die Konzentrations- und Leistungsfähigkeit des Menschen wird durch viele Faktoren beeinflusst. Beispielsweise stellt Müdigkeit eine große Gefahr dar, aber auch unterschätzter Einfluss von Medikamenten oder Alkohol. Durch einen vollständig autonomen Betrieb auf den Straßen, könnten Unfälle aufgrund menschlichen Versagens vermieden werden.

Abgesehen von der Verringerung der Unfallzahlen spielt auch der Komfortgewinn eine Rolle. So ist vorstellbar, dass man auf dem Weg zur Arbeit entspannt liest anstatt konzentriert über die Autobahn zu fahren. Ebenso könnten in der Zukunft körperlich kranke und ältere Menschen, die nicht mehr in der Lage sind ein Auto selbst zu fahren, vom autonomen Fahren profitieren. Man könnte ihre Mobilität und somit auch Lebensqualität länger aufrechterhalten (vgl. Maurer u. a., 2015, S.4 f.f.).

Ein weiterer positiver Effekt ist die Verbesserung des Verkehrsflusses. Die schnellen Reaktionszeiten der Systeme lassen einen kürzeren Abstand zwischen den Fahrzeugen zu. Durch „car-to-car“-Kommunikation könnte eine Kolonnenbildung und Vorrausplanung autonomer Fahrzeuge die Durchschnittsgeschwindigkeit auf stärker befahrenen Straßen erhöht werden. Die dadurch hervorgerufene Verringerung von Stau- und Verlustzeiten führt wiederum zu einer Verringerung der benötigten Energie der Verkehrsteilnehmer und einer Steigerung der Qualität des Verkehrsabflaufs. Voraussetzung ist allerdings ein rein autonomer

Betrieb auf den Straßen. Auch wenn in der nahen Zukunft von einem Mischbetrieb aus autonomen und manuell gesteuerten Fahrzeugen ausgegangen wird, liegt viel Potenzial in der Optimierung des Verkehrs. (vgl. Maurer u. a., 2015, S.347 f.f.)

1.2. Aufgabenstellung

Systematische Entwicklung von Anforderungen und Implementierung eines modularen, Fahrzeug-integrierten Systems zur Auswertung und Korrektur von bildgebenden Sensoren zur Anwendung für das automatisierte und autonome Fahren.

1.3. Ziel der Arbeit

Ziel der Arbeit ist es, die Voraussetzungen für Untersuchungen bezüglich des autonomen Fahrens an einem Versuchsfahrzeug zu schaffen, in dem eine Auswertung der auf dem Fahrzeug befindlichen bildgebenden Sensoren während der Fahrt realisiert wird.

Hierzu wird durch Anforderungsanalysen geeignete Hardware für die Auswertung ausgewählt und die Integration in das Fahrzeug konzeptioniert und realisiert.

Für die Softwareauswertung der Sensordaten werden Anforderungen entwickelt, Konzepte erstellt und implementiert. Die Auswertung soll einen Einstieg in die Problemstellungen und aktuelle Thematik des autonomen Fahrens darstellen. Zu dieser Thematik zählt die zuverlässige und präzise Lokalisierung von Fahrzeugen in einem urbanen Umfeld sowie das Erstellen sehr genauer Umgebungskarten, die ihrerseits wieder für die Lokalisation und Navigation genutzt werden können.

Aufgrund dessen soll mit den bildgebenden Sensoren eine gleichzeitige Lokalisierung des Fahrzeugs und Kartografie der Umgebung während der Fahrt erfolgen.

In abschließenden Tests wird das Gesamtsystem beurteilt. Es wird dabei insbesondere auf die Eignung der automatischen Lokalisation und Kartenerstellung für die Anwendung beim autonomen Fahren eingegangen.

1.4. Struktur der Arbeit

Die Arbeit unterteilt sich in fünf Bereiche. Der erste befasst sich mit den Grundlagen und dem Stand der Technik. Es wird die Technik des autonomen Fahrens und die Grundlagen bildgebender Sensoren erläutert. Ebenfalls wird die Positionsbestimmung und Kartografie thematisiert.

Der zweite Bereich stellt die Analyse und Design und Realisierung der Hardware dar. Es erfolgt zunächst eine Anforderungsanalyse und Design des Gesamtsystems und anschließend die Anforderungsanalyse des Computersystems. Nach der Auswahl eines geeigneten

Systems wird die Integration der Hardware in das Versuchsfahrzeug geplant. Abschließend erfolgt eine Beschreibung der Realisierung.

Im dritten Teil wird die Software des Systems analysiert, designet und realisiert. Dies schließt die Anforderungsanalyse einer geeigneten Softwareplattform und ihre Auswahl ein. Danach werden die Anforderungen an die Software für die ausgewählte Plattform analysiert und ausgewählt. Im Design-Teil wird das Konzept anhand der Auswahl erläutert und schließlich realisiert.

Nach dem die Hard- und Software realisiert ist, werden im nächsten Teil Tests durchgeführt, um die Leistungsfähigkeit des Gesamtsystems zu testen und zu überprüfen, ob die gestellten Anforderungen eingehalten werden.

Schließlich werden die Ergebnisse zusammengefasst und ein Ausblick über die mögliche Weiterentwicklung des Systems gegeben.

2. Grundlagen / Stand der Technik

In dem Stand der Technik wird zunächst auf den Grad der Automatisierung aktueller Fahrzeuge eingegangen. Es folgt eine exemplarische Darstellung der Fahrzeugsensorik für die Positionsbestimmung unter Verwendung hochgenauer Straßenkarten.

Der nächste Teil befasst sich mit den Grundlagen bildgebender Sensoren. Insbesondere die Funktionsweise von Lidar-Systemen wird detaillierter erklärt.

Zum Schluss findet eine einführende Darstellung der optisch basierten Positionsbestimmung für die Kartenerstellung auf Basis dreidimensionaler Umgebungsinformationen statt. Zusätzlich werden grundlegende Kartentypen für das speichern dieser Informationen erläutert.

2.1. Stand der Technik

Um aktuell auf dem Markt befindliche, aber auch zukünftige Fahrzeuge in ihrer Leistungsfähigkeit bezüglich autonomen Fahrens einschätzen zu können, gibt es eine Kategorisierung. Diese wird in Europa unter anderem durch die Bundesanstalt für Straßenwesen (vgl. Gasser u. a., 2012) und in den USA durch die SAE (Society of Automotive Engineers, vgl. SAE (2014)) definiert. Die Kategorisierung ist in Stufen bzw. Level eingeteilt und reicht von 0-4 (BaSt) und von 0-5 (SAE). Eine Null steht für keine Automatisierung oder Assistenzsysteme und Stufe vier bzw. fünf repräsentiert das fahrerlose Fahren. Die Passagiere sind in Stufe fünf (SAE) lediglich für die Start- und Zieleingabe sowie das Starten des Systems verantwortlich.

Aktuell auf dem Markt verfügbare Fahrzeuge (Stand 3/2018) bieten noch nicht die Stufe fünf (SAE). Der Audi A8 D5, welcher seit Ende 2017 zum Verkauf angeboten wird, ist mit dem Automatisierungslevel 3 der SAE angegeben. Er unterstützt die Verwendung des „ Audi AI Staupiloten“. Eine Funktion, die es dem Fahrzeug erlaubt auf mehrspurigen Kraftfahrstraßen mit baulich abgetrennter Gegenfahrbahn bis zu 60 km/h die Fahraufgabe abzunehmen. Der Fahrer muss dieses System betrachtet nicht mehr ständig überwachen. Das von Audi realisierte System wird als eine Revolution bezeichnet, weil es durch Sensorfusion erstmals ein permanentes Abbild der Umgebung erzeugt. (vgl. Audi, 2017a, S.)

Im Gegensatz zum Audi A8 besitzt der Tesla Model S einen SAE Automatisierungsgrad der Stufe 2. Der Tesla-Autopilot gilt mit seiner Lenkautomatik jedoch nur als ein Assistenzsystem. Dieser Modus führt bereits zu einer nachgewiesenen Reduktion der Unfallquote. (vgl. t3n.de, 2017)

Um die hohen Stufen der Automatisierung realisieren zu können, wird die genaue Position des Fahrzeuges in seiner Umgebung benötigt. Und das zu jedem Zeitpunkt, in Echtzeit und mit einer Toleranz im Zentimeterbereich. Basis der bisherigen Positionsbestimmung ist das GPS-System. Dieses stößt insbesondere in Gegenden mit hohen Gebäuden oder Landschaften mit Bergen und Felswänden in der direkten Nähe aufgrund der Abschattung oder Reflexion der Satellitensignale an seine Grenzen. Und auch ein GNS-System (Global Navigation Satellite System) mit Netz-RTK (Realtime Kinematics) basiertem Empfang von Korrekturdaten und Fusion mit einem Sensor für Trägheitsnavigation wird innerhalb von Tunnelabschnitten oder Tiefgaragen aufgrund der Signalabschattung zu ungenau. (vgl. Rauch u. a., S. 3 f.))

Die Position des Fahrzeugs und Beschaffenheit seiner Umgebung wird deshalb durch die intelligente Auswertung zusätzlicher Sensoren wie Lidar, Radar, Ultraschall und Kameras bestimmt. Nachfolgende Abbildung 2.1 zeigt die Positionen und Abstrahlwinkel der Sensoren eines Audi A7, der für „piloted driving“ ausgerüstet ist. Dieser Modus entspricht der Kategorie drei des automatisierten Fahrens. (vgl. Audi, 2017b) Die Sensorblickwinkel überschneiden sich, dies erhöht die Redundanz der Sensorwerte und somit die Sicherheit bei der Objekt und Umgebungserkennung.

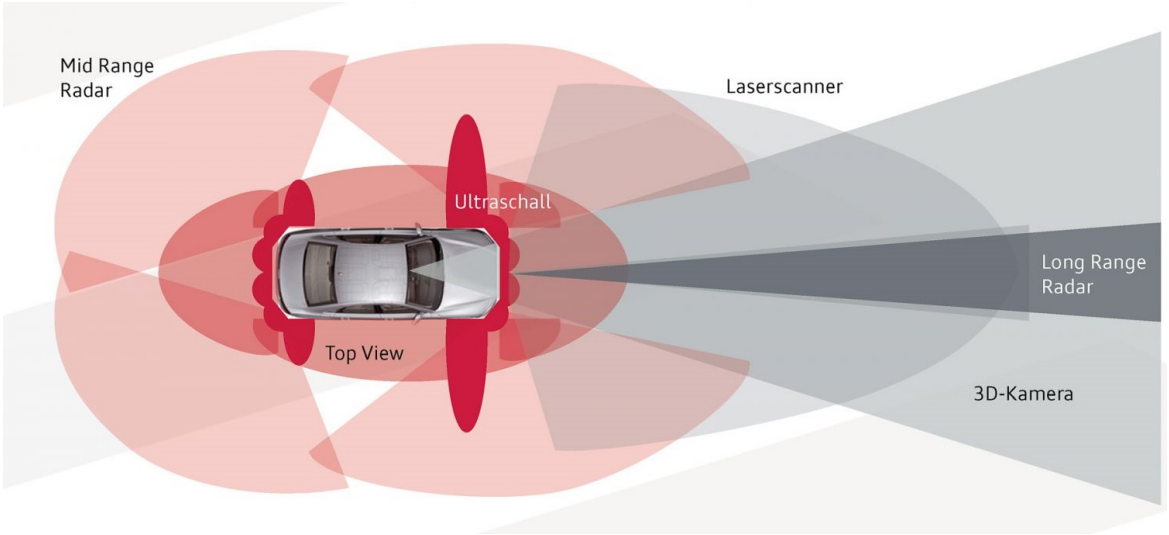

Abbildung 2.1.: Anordnung verschiedener Sensortypen und ihre Reichweite (Quelle: Gleitsmann und Audi-Deutschland)

Zusätzlich zu den Sensoren wird eine digitale Straßenkarte verwendet. Diese ist nicht mit der Karte von handelsüblichen Navigationsgeräten vergleichbar, sondern als ein sehr genaues Abbild der Straßen zu verstehen, in dem verschiedenste Informationen der Umgebung gespeichert werden. Hierzu zählt nicht nur eine dreidimensionale Darstellung, sondern auch Bildinformationen von Kameras, wie die Position von Spurmarkierungen. Mit Hilfe solch einer digitalen Straßenkarte und Algorithmen für die Erkennung von Landmarken wie z.B. der Spurmarkierungen oder auffällige Strukturen und Muster findet dann eine sehr genaue Abschätzung der aktuellen Position statt (vgl. Rauch u. a., S. 5 ff.). Während der Fahrt wird die statische Karte mit dem aktuellen Sensorabbild der Umgebung verglichen und dynamische Objekte, wie andere Fahrzeuge oder neue Hindernisse als dynamische Objekte der Karte hinzugefügt. Durch die Sensorfusion und erzeugen eines vollständigen Abbildes statischer und dynamischer Umgebung ist es bereits 2011 gelungen, ein Forschungsfahrzeug auf einer Autobahn-Teststrecke völlig autonom mit relativ hohen Geschwindigkeiten (130 km/h) fahren zu lassen. Für die ca. 65 Kilometer lange Strecke wurde vorher mittels eines neu entwickelten Verfahrens eine digitale Straßenkarte erzeugt. (vgl. Rauch u. a., S. 1 f.)

Die Erfahrung, die bereits im Bereich des autonomen Fahrens gesammelt wurde zeigt, wie wichtig sehr genaue Straßenkarten für die Navigation sind. Das Erzeugen von digitalen Straßenkarten, welche vom Informationsgehalt auf die Sensorik autonomer Fahrzeuge zugeschnitten ist, spielt eine zentrale Rolle für die Entwicklung.

Verschiedene Kartenanbieter wie Google und Nokia Here spezialisieren sich bereits auf diesem Gebiet und haben mit der Aufnahme detaillierterer Karten, sogenannten „HD maps“, für zukünftige Autogenerationen begonnen. Für die Kartierung werden speziell ausgerüstete Fahrzeuge verwendet, die in der Lage sind, ein dreidimensionales Umgebungsbild zu erzeugen. (vgl. Bergen, 2018)

Als HD maps werden Karten bezeichnet, die speziell für die neuen Autopiloten-Systeme erstellt werden und eine Genauigkeit der Darstellung dreidimensionaler Umgebung im Zentimeterbereich anvisieren.(vgl. Vardhan, 2017)

Zum Einsatz kommen für diesen Zweck ein oder mehrere Lidar-Sensoren für die dreidimensionale Abtastung mit Infrarotlasern und verschiedene Kameras sowie GPS-Systeme für die Lokalisierung. Die Abbildung 2.2 zeigt solch ein Fahrzeug, mit dem Nokia here Messfahrten durchführt. Auf dem Dachträger am oberen Ende der Säule befindet sich ein Lidar-Sensor und darunter ein 360°-Kamerasystem.

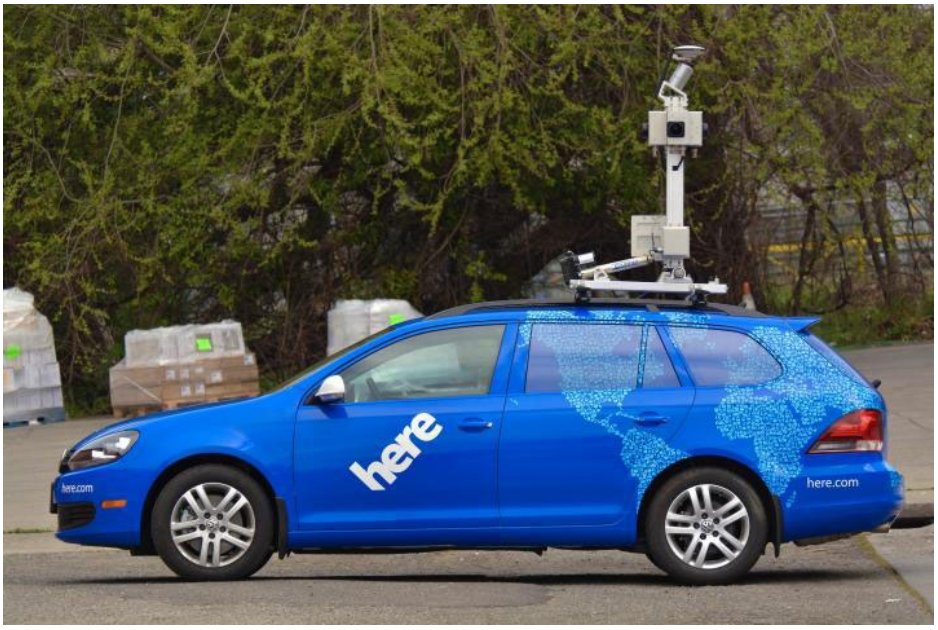

Abbildung 2.2.: Messfahrzeug der Firma Nokia here (Quelle: NokiaHere, 2014)

2.2. Grundlagen bildgebender Sensoren

Die bildgebende Sensortechnologie umfasst ein breites Feld verschiedener Sensoren. Zu den bekanntesten Vertretern gehören einfache Kameras für den Spektralbereich des sichtbaren Lichts. Doch auch komplexere Systeme wie 360-Grad-Kameras, Radar oder Laserscanner fallen in diese Kategorie. Allen gemeinsam ist die Möglichkeit, sich aus aufgenommenen Messdaten ein Abbild der Realität erzeugen zu können.

Nachfolgend werden die Grundlagen und Konzepte von Kamerasystemen und Lidar-Sensoren mit Bezug zur Forschung des autonomen Fahrens erläutert.

2.2.1. Kamerasysteme

Kameras bieten einen sehr hohen Informationsgehalt in den Bild- und Videodaten. Beim autonomen und auch beim assistierten Fahren gibt es verschiedene Einsatzzwecke, die mit je unterschiedlichen Systemen bedient werden. Beispielsweise wird aus vier bis sechs nach von gerichteten Kameras ein dreidimensionales Bild erzeugt. Das Prinzip dahinter entspricht dem von Stereokameras, welche durch die Verschiebung von Objekten relativ zu beiden Kamerabildern die Entfernung bestimmen. Des weiteren werden Kameras für die Objekt- und Mustererkennung in mittlerer Entfernung von 100-200 Metern verwendet. Zu der Mustererkennung gehört unter anderem das Erkennen von Straßenbeschilderung. Ebenso wird im mittleren Bereich der Querverkehr oder Fußgänger erkannt.

Um eine 360°-Überwachung rund um das Fahrzeug gewährleisten zu können werden mehrere Kameras an verschiedenen Positionen angebracht. Die Bilder der einzelnen Kameras überlappen sich und werden meistens für die Verarbeitung in ein zentrales Steuergerät übertragen. (vgl. Rudolph und Voelzke, 2017)

Der Vorteil von Kamerasystemen für die Objekterkennung liegt unter anderem in der Auflösung und Reaktionszeit verglichen mit Ultraschallsensoren. Ein großer Nachteil ist jedoch durch die Abhängigkeit der Lichtverhältnisse gegeben. Starkes Gegenlicht kann die Sicht des Sensors stark einschränken.

Tesla integriert in seine neue Autogeneration (Stand 2018) ein System für 360° Rundumsicht bestehend aus acht Kameras. Damit soll es möglich sein, die Umgebung bis zu einer Entfernung von 250 Metern zu überwachen. (vgl. Tesla, 2018)

Für Forschungszwecke existieren Kamerasysteme, die eine 360°-Rundumsicht unabhängig von einem Fahrzeugtyp bieten. Sie integrieren eine entsprechende Zahl von Kameras in ein Gehäuse und eignen sich durch ihren Aufbau für die Montage auf Sensorträgern.

Bei dem in dieser Arbeit eingesetzten Ladybug 5+ handelt es sich um ein solches System. Es besteht aus einem pentagonalen Grundkörper, in dessen Flächen sechs einzelne Kameras integriert sind. Fünf von sind für die Rundumsicht zuständig und eine zusätzliche Kamera deckt den Bereich über den Sensor ab. In Abbildung 2.3 ist der Grundkörper ohne Säule und Montageplatte zu sehen. Insgesamt werden 90% einer Sphäre abgebildet. Die einzelnen Kameras sind baugleich und besitzen eine Auflösung von je sechs Megapixeln.(vgl. Flir, 2017b)

Abbildung 2.3.: Ansicht des Ladybug 5 Kamerasystems (Quelle: Flir, 2017a)

Die Einzelbilder werden über einen Bildprozessor verarbeitet und anschließend aneinander gereiht und komprimiert. Die Übertragung der Bilddaten auf einen Computer erfolgt über eine USB 3.1 Schnittstelle. Die Nachfolgende Abbildung 2.4 zeigt den Datenfluss einer Ladybug 5+. (Flir, 2017b)

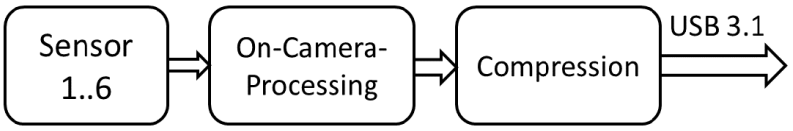

Abbildung 2.4.: Datenfluss des Kamerasystems

2.2.2. Lidar-Sensoren

Die Abkürzung Lidar steht für engl. „light/laser detection and ranging“. Somit ist die Bezeichnung Lidar-Sensor sehr allgemein und beinhaltet alle Sensoren, die mit Hilfe von Licht oder Laserstrahlen eine Entfernung bestimmen können. Die Grundlegenden Funktionsweisen sind bei den meisten Sensoren jedoch die gleichen.

In günstigeren Systemen wird häufig die Triangulationsmethode verwendet. Dabei wird ein Laserstrahl ausgesendet und von einem Objekt oder einer Wand reflektiert. Je nach dem in welcher Entfernung sich das Objekt zu dem Sensor befindet, ändert sich der Winkel des Reflektierten Strahls gegenüber dem Detektorelement des Sensors. Der Strahl trifft somit auf unterschiedliche Stellen des Detektors, wodurch über die Winkelbeziehungen auf die Entfernung des Objekts schließen lassen. (vgl. Micro-Epsilon, 2018)

Ebenfalls vertreten ist das Bestimmen einer Distanz aufgrund der Phasenverschiebung zwischen ausgesendetem und reflektiertem Licht. Es wird ein kontinuierlicher amplitudenmodulierter Laserstrahl ausgesendet, reflektiert und wieder empfangen. Anschließend wird die Phasenverschiebung der Signale bestimmt und mit Hilfe der Lichtgeschwindigkeit die Entfernung berechnet. (vgl. Janos, 2008)

Bei den für das autonome Fahren eingesetzten Systemen wird im Gegensatz dazu ein weiteres Verfahren angewendet. Basis dafür bildet die ToF-Messung (engl. Time of Flight), also die Ermittlung der Zeitdauer vom Aussenden bis zum Empfangen eines Laserpulses. Für eine genaue Zeitbestimmung wird eine hohe temporale Auflösung und Genauigkeit im Picosekundenbereich benötigt. In der Abbildung 2.5 sieht man die Prinzip-Darstellung eines eindimensionalen Lidar-Sensors. Neben dem Laser, Detektor und Timer-Modul können verschiedene Optiken den gewünschten Strahlengang erzeugen. (vgl. picoquant.com)

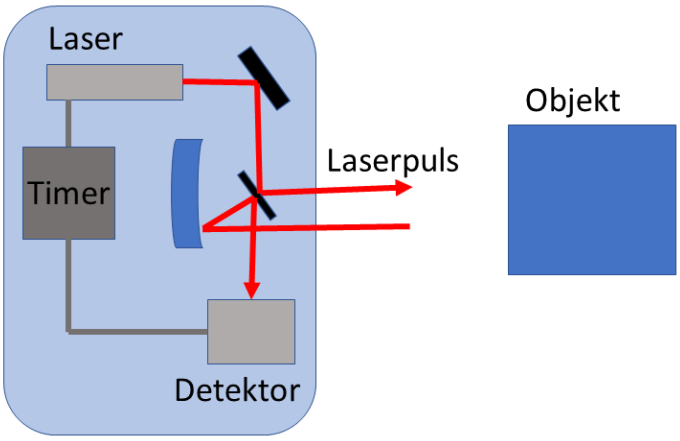

Abbildung 2.5.: Aufbau eines eindimensionalen „time of flight“’-Sensors

Für die Abtastung der Umgebung und Nutzen in autonomen Fahrzeugen ist ein einzelner Sensor in dieser Ursprungskonfiguration noch nicht geeignet. Um die räumliche Auflösung zu erhöhen, also mehr Messpunkte an verschiedenen Positionen im Raum zu generieren, gibt es je nach Anwendung verschiedene Methoden.

Eine davon ist es, mehrere Sensoreinheiten nebeneinander anzuordnen, so dass man ein mehrkanaliges System erhält. Diese sind auf Fahrzeugen meistens nach vorn ausgerichtet und erlauben so die Unterscheidung von zum Beispiel 12 horizontalen Segmenten in einem Blickwinkel von 20°-100°. Die räumliche Auflösung solcher Systeme ist gering, die Bildwiederholfrequenz jedoch hoch (100Hz). (vgl. Proxitron, 2018)

Für die Abtastung der Umgebung und das Erstellen von Karten werden meist rotierende Lidar-Sensoren verwendet. Das Prinzip dahinter ist über eine Mechanik und Spiegelsystem den Strahl des Lasers rotieren zu lassen. Hierzu wird zusätzlich eine Information über den aktuellen Winkel des Lasers benötigt. Die Auflösung in der Scan-Ebene ist hoch. Sie hängt mit der Abtastrate und der Rotationsfrequenz zusammen.

Für die dreidimensionale Erfassung des Raumes werden zusätzliche Sensoreinheiten wieder zu einem mehrkanaligem System kombiniert. Die Anordnung erhöht nun allerdings nicht die horizontale Auflösung, sondern die vertikale. Durch die unterschiedlichen Winkel, mit denen die Laserpulse ausgesendet werden, erhöht man die vertikale Auflösung in Form von zusätzlichen Zeilen, in denen die Umgebung abgetastet wird. Abbildung 2.6 stellt das Prinzip anschaulich dar.

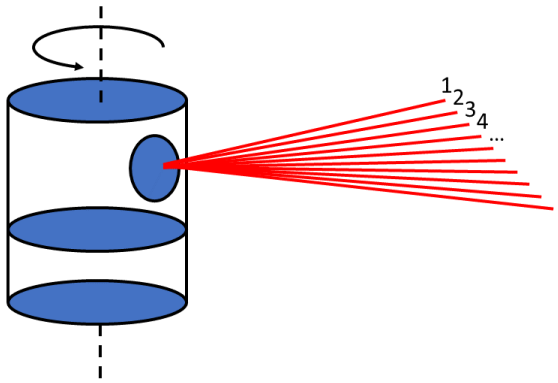

Abbildung 2.6.: Prinzip eines 360° Lidar-Systems mit einer vertikalen Auflösung von X-Zeilen (Quelle: In Anlehnung an xyht.com, 2017)

Im Falle eines Velodyne HDL-32E, an den die obige Abbildung angelehnt ist, wird die Umgebung mit 32 Zeilen abgetastet. Die einzelnen Zeilen spannen einen Winkel von -30,67° bis +10,67° bezogen auf die horizontale Achse auf. Die Rotationsfrequenz kann zwischen 5-20 Hz eingestellt werden. Jedes einzelne Laser/Detektor Paar hat eine Abtastrate von 21,7 kHz, was bei 32 dieser Einheiten zu einer Erzeugung von bis zu 695.000 Raumpunkten pro

Sekunde führt. Die Laserpulse werden auf einer Wellenlänge von 903 Nanometern gesendet und sind somit im nahen Infrarotbereich und für das menschliche Auge nicht sichtbar. Durch die Spezifikation der Laserklasse 1 sind die Pulse ungefährlich und der Velodyne kann bedenkenlos im urbanen Umfeld eingesetzt werden.

Der Velodyne HDL-32E wiegt ca. 1kg und hat einen Durchmesser von 85mm bei einer Höhe von 144mm. Die Abbildung 2.7 zeigt seine zylindrische Form. Das obere Segment ist der rotierende Teil mit einer Öffnung für die Optik.

Abbildung 2.7.: Velodyne HDL-32E (Quelle: velodynelidar.com)

Die Datenübertragung erfolgt über eine Ethernet-Schnittstelle nach dem UDP (engl. User Datagramm Protocol). Der Vorteil von Scanning-Lidar ist, dass die lateralen Informationen und Tiefeninformation jedes einzelnen Scan-Endpunktes gemeinsam vorliegen und vom Sensor übertragen werden. Mit der Software VeloView des Herstellers können die empfangenen Daten gespeichert und angezeigt werden. Eine solche Beispielaufnahme der auch Punktwolke oder engl. pointcloud genannten Daten ist in der nachfolgenden Abb. 2.8 zu sehen. Die einzelnen Zeilen in denen eine Abtastung erfolgt sind deutlich erkennbar. (vgl. Velodyne, 2016)

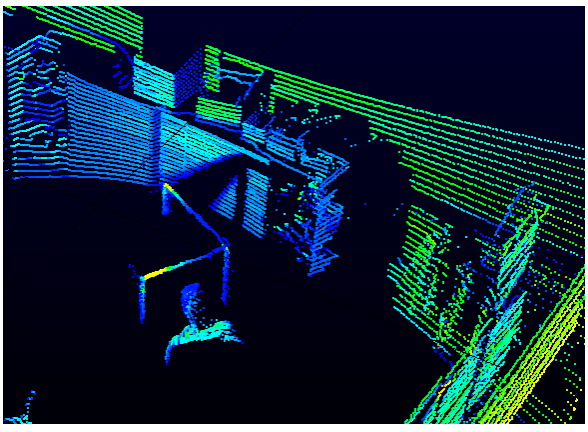

Abbildung 2.8.: Beispielaufnahme eines Velodyne HDL-32E, dargestellt mit VeloView

2.3. Positionsbestimmung und Kartografie

Es wird das Thema des gleichzeitigen Lokalisierens und Kartografieren durch Sensordaten behandelt. Für die Navigation autonomer Fahrzeuge ist es essentiell und in der Robotik unter der Abkürzung SLAM (engl. Simultaneous Localization And Mapping) bekannt. Abhängig vom Anwendungsgebiet werden dafür verschiedene Kartentypen verwendet, von denen die übergeordneten Klassen erläutert werden.

2.3.1. Grundlagen der Kartenrepräsentationen

In der Nachfolgenden Tabelle 2.9 ist eine Übersicht häufig verwendeter Kartendarstellungen zu sehen. Von ihnen gibt es verschiedene Anpassungen und Verbesserungen, die sich in Unterkategorien einordnen lassen.

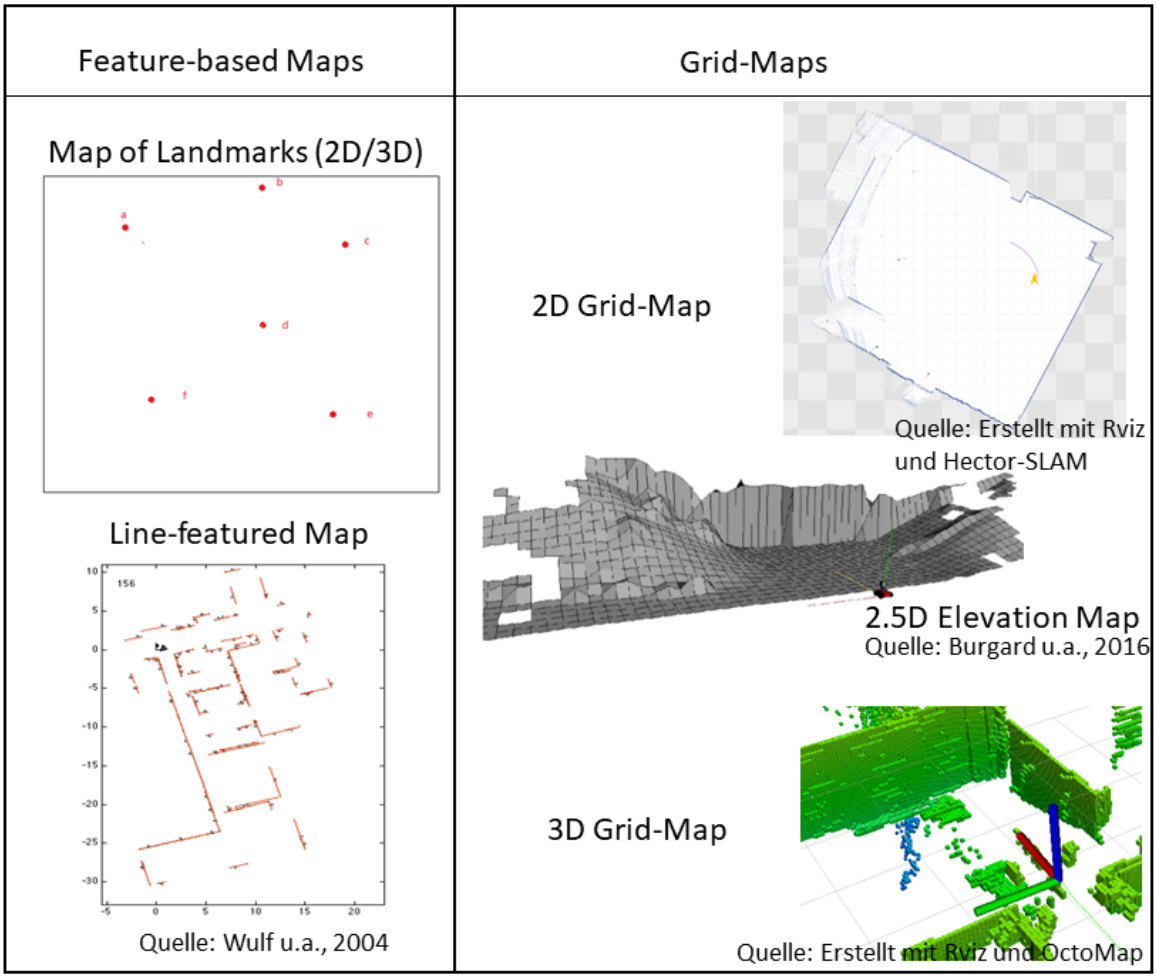

Abbildung 2.9.: Verschiedene Kartentypen für die Lokalisation

Die Hauptunterscheidung der Karten liegt im darin, ein detailliertes Abbild der Umgebung wiederzugeben oder sich auf bestimmte Eigenschaften zu beschränken. Eigenschaftsbasierte Karten (engl. feature-based maps) stellen eine sehr starke Vereinfachung der Umgebung dar und beschränken sich auf Details, die als Landmarken für die Lokalisierung und Navigation genutzt werden können. Die andere Klasse von Karten stellt die Umgebung in zwei- oder dreidimensionalen Rastern unterteilt dar und werden im englischen als „occupancy grid maps“ bezeichnet. Jede einzelne Flächen- oder Volumeneinheit kann durch Variablen

verschiedene Informationen speichern. Wird von den Sensoren ein Objekt erkannt, werden die entsprechenden Zellen als besetzt markiert. Ist der Sensor zum Beispiel ein Laserscanner, so werden die Zellen entlang des Strahls zwischen Sensorposition und Objekt als unbesetzt markiert. Noch nicht abgetasteter Raum hat den Status unbekannt. Meist wird mit diesen Karten versucht, die Umgebung mit allen von den Sensoren erfassten Details in diskretisierter Form wiederzuspiegeln. Die erreichbare Genauigkeit so einer Karte ist durch die gewählte Auflösung der kleinsten Flächen- oder Volumeneinheit limitiert. Im Gegensatz dazu ist die darstellbare Genauigkeit einer eigenschaftsbasierten Karte nur von der Präzision der berechnenden Hardware abhängig. Also mit welcher Genauigkeit Berechnungen durchgeführt und was für einer Bitlänge Zahlen vom System ausgedrückt und gespeichert werden können. (vgl. Lee, 1996, S. 29 f.)

Es ist leicht vorstellbar, dass eine eigenschaftsbasierte Karte wesentlich geringere Anforderungen an die Speicherkapazität hat, als eine hochaufgelöste dreidimensionale Grid-Map, die das selbe Areal darstellt. Aus diesem Grund hängt die Auswahl der Karte stark mit dem Verwendungszweck zusammen. Für die reine Lokalisierung werden eigenschaftsbasierte Karten bevorzugt. Soll allerdings im Zuge einer intelligenten Navigation eine Pfadplanung erfolgen, ist in unregelmäßigem Gelände eine Grid-Map von Vorteil, da für die Pfadplanung alle Begrenzungen und Hindernisse berücksichtigt werden müssen. Dies schließt auch dynamische Objekte wie Fußgänger oder Fahrzeuge ein, dessen Abmessungen für ein Umfahren bekannt sein müssen.

Eine weitere Variante für die Navigation in künstlicher Umgebung mit vielen Wänden und großen flächigen Strukturen ist eine eigenschaftsbasierte Karte mit geraden Linien als speicherbare Eigenschaften, wie in Abb. 2.9 zu sehen ist. Jede Wand muss jedoch durch entsprechende Algorithmen aus den Sensordaten erkannt werden, wofür Rechenkapazität benötigt wird. Der Vorteil liegt darin, dass große Strukturen stark vereinfacht werden und im Falle von geraden Mauern dennoch ein recht genaues Abbild der Umgebung liefern. Durch die Vereinfachung wird, verglichen mit einer Grid-Map viel Speicherplatz gespart, da beispielsweise nur der Anfangs- und Endpunkt des kompletten Mauerstücks gespeichert wird, anstatt jede Zelle entlang der Mauer bei einer Grid-Map. (vgl. oliver Wulf u. a., 2004, S. 4204 ff.)

Von einem Team der Universität Freiburg wurde eine Kartenrepräsentation auf Basis einer 3D-Grid-Karte entwickelt, die durch eine spezielle Datenstruktur den Speicherbedarf gegenüber der ursprünglichen Variante verringert (vgl. Hornung u. a., 2013). Das OctoMap genannte Framework speichert Daten hierfür in einer Baumstruktur, die sich bis zu einer festgelegten Ebene immer weiter verfeinern lässt (vgl. Abbildung 2.10). Das Auflösen der räumlichen Struktur geschieht durch Oktanten, wodurch sich ein übergeordneter Würfel in acht Oktanten der darunter liegenden Ebene aufteilt. Abbildung 2.11 zeigt eine besetzte Zelle in einer unterschiedlich aufgelösten Oktanten-Struktur. Diese Abbildung stellt grafisch die Informationen der Baumstruktur der darüber angeordneten Abb. 2.10 dar. Jeder Oktant (grauer Kreis)

enthält acht Kinder und somit acht Äste. Ein weißes Quadrat bedeutet, dass dieser einzelne Voxel nicht besetzt ist und ein schwarzes Quadrat deutet auf einen besetzten Voxel hin. Der Begriff Voxel ist ein Kunstwort und aus den beiden englischen Wörtern volume und elements zusammengesetzt. Er bezeichnet in diesem Zusammenhang einzelne Zellen der dreidimensionalen Struktur.

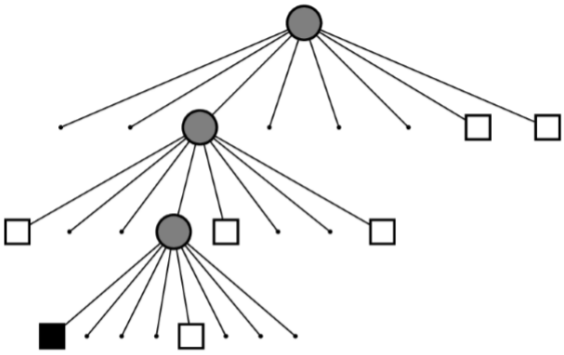

Abbildung 2.10.: Baumstruktur der OctoMap (Quelle: Hornung u. a., 2013, S. 4)

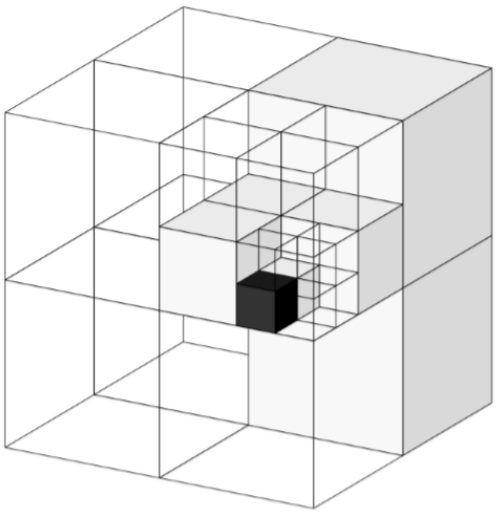

Abbildung 2.11.: Grafische Darstellung der Baumstruktur aus Abb. 2.10 (Quelle: Hornung u. a., 2013, S. 4)

Der Vorteil einer so entstehenden Karte liegt darin, dass nicht jeder Bereich der Karte voll aufgelöst sein muss. Enthalten alle Kinder eines Baumes die gleichen Informationen, können

diese zusammengefasst und allein von der darüber liegenden Ebene repräsentiert werden. Dies macht sich in der Praxis besonders dann bemerkbar, wenn viele große Wandstücke oder Objekte von den Sensoren erfasst werden. Die Kontur der Oberfläche kann weiterhin exakt aufgelöst sein, während die innere Auflösung der Objekte nicht mehr relevant ist und verringert werden kann.

Innerhalb des Speichers erfolgt der Zugriff auf einzelne Oktanten mit Pointern auf die darunter liegende Ebene, wie auf der linken Seite der Abbildung 2.12 zu sehen ist. Enthält ein Datensatz Informationen über Kinder eines Oktanten, so zeigt der Pointer auf das erste Element eines Arrays mit acht Elementen, aus denen dieser Oktant besteht.

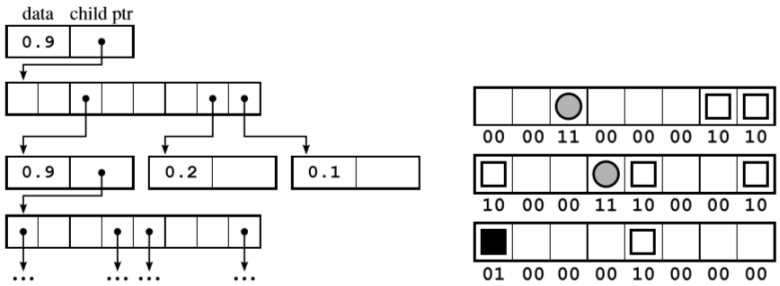

Abbildung 2.12.: Übersicht über die Subsysteme von Hector-SLAM (Quelle: Hornung u. a., 2013, S. 4)

Abgespeichert wird die Karte als eine einzige Bitfolge mit einer zwei Bit Kodierung. Eine 00 bedeutet, dass der Inhalt unbekannt ist, 01 steht für einen besetzten Oktanten, 10 für einen freien Oktanten und zuletzt bedeutet 11, dass als nächstes eine Bitfolge der Kinder des Oktanten folgt. Auf der rechten Seite der Abbildung 2.12 ist die Bitfolge des Beispiels aus Abbildung 2.11 zu sehen. Ein grauer Kreis bedeutet, dass die nächsten acht Elemente die Informationen über die Kinder enthalten. Ein weißer Kasten steht für ein freies Feld und ein schwarzer Kasten für ein besetztes.

In der aktuellen Implementierung der OctoMap ist die Tiefe der Baumstruktur auf 16 Ebenen begrenzt. Somit ist auch die maximale Anzahl der möglichen Voxel auf $2^{48} = 2,1815 * 10^{14}$ begrenzt. Um sich eine Vorstellung davon zu machen, wie diese Zahl zustande kommt, beginnt man mit dem Ursprungs-Voxel. Dieses unterteilt sich in 8 Oktanten. Jeder der Oktanten lässt sich wiederum in acht Oktanten unterteilen. Wiederholt man den Vorgang insgesamt 16 mal, erhält man den Ursprungs-Voxel mit einer Unterteilung in 65536 (2^{16}) Voxel je Kantenlänge in der niedrigsten Ebene. Wählt man für die Kantenlänge der kleinsten Einheit einen Zentimeter, kann man ein Volumen von 655,36m*655,36m*655,36m darstellen. Ein erweitern der Ebenen auf 32 ist generell auch möglich und würde dann bei gleicher Auflösung

ein Gebiet von $2^{32} = 42949672,96m$ abdecken können. Weiterhin ist es möglich die Informationen eines Oktanten zu erweitern und zusätzliche Daten bezüglich Reflexionsgrad oder Farbton hinzuzufügen. (vgl. Hornung u. a., 2013, S.6 ff.)

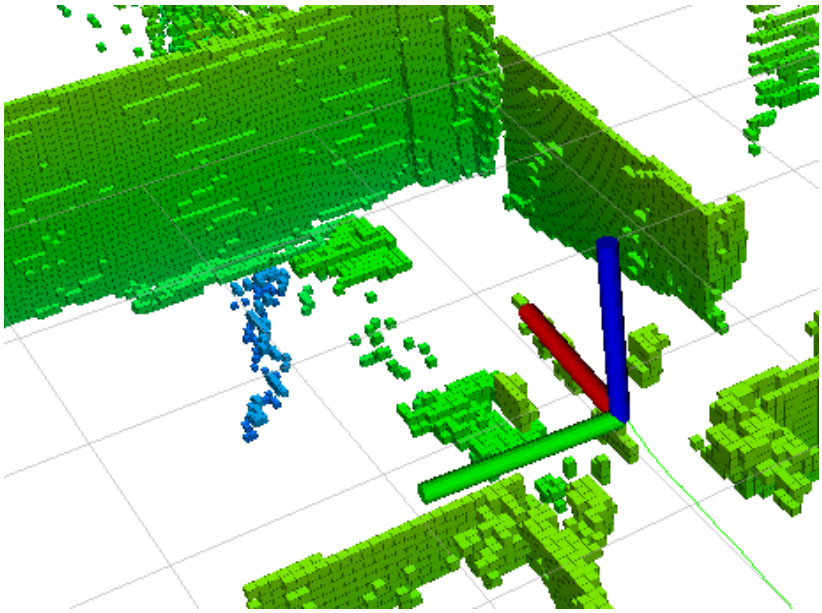

Abbildung 2.13.: Eine mit ROS und Rviz dargestellte OctoMap-Karte

In der obigen Abbildung 2.13 ist der Ausschnitt einer mit OctoMap erstellten Karte zu sehen. Ein einzelner Voxel entspricht einem Würfel mit einer Kantenlänge von drei Zentimetern in der Realität.

2.3.2. Grundlagen des gleichzeitigen Lokalisieren und Kartografierens (SLAM)

SLAM steht für „Simultaneous Localization And Mapping“ und ist ein überordnender Begriff, der die grundlegende Lösung eines einfach darzustellenden Problems beschreibt. Es wird in der Literatur gerne als Henne-Ei-Problem bezeichnet und muss gelöst werden, wenn ein autonomes Fahrzeug oder Roboter in unbekanntem Terrain navigieren muss. Denn dadurch, dass man durch neues unbekanntes Terrain fährt, für das noch keine Karte existiert, kommt es zu einem Paradoxon. Ohne Karte, kann man seine eigene Position nicht durch Wiedererkennen von Mustern oder Landmarken ermitteln. Kennt man aber seine eigene Position nicht, ist man auch nicht in der Lage sich aus den Sensordaten eine neue Karte zu erzeugen.

SLAM Lösungsansätze versuchen beides gleichzeitig. Sie ermitteln aus den laufenden Sensordaten die eigene relative Positionsveränderung und erweitern die Karte mit Hilfe der neuen Position kontinuierlich. Unter die Kategorie SLAM fallen alle Arten von Algorithmen und Methoden, die das geschilderte Problem lösen. Sehr abstrahiert betrachtet besteht SLAM aus zwei Teilen, der Lokalisierung und einer erweiterbaren Karte zum speichern der Umgebungsinformationen. Nach Erhalt der ersten Sensorinformationen wird mit der initiierten Position eine erste Karte erstellt. Im nächsten Schritt wird angenommen, dass sich der Roboter oder das Fahrzeug ein Stück bewegt. An der neuen Position werden wieder Sensorwerte gesammelt. Neue und alte Werte können nun in Relation zueinander gesetzt werden, wodurch die neue Position des Sensors bekannt wird. Danach wird die Karte abgeglichen und neue Eigenschaften oder Details hinzugefügt, die im nächsten Schritt wieder für die Positionsfindung genutzt werden können.

Bedingt durch Rauschen und limitierte Genauigkeit der Sensoren werden SLAM-Ansätze meistens auf Basis von wahrscheinlichkeitstheoretischen Werkzeugen realisiert (vgl. Grisetti u. a.). Die Auswahl wird durch viele Faktoren beeinflusst. Zu ihnen gehören beispielsweise die Art der Sensoren, die Umgebungsbeschaffenheit und Leistungsfähigkeit der berechnenden Hardware.

Unterschieden werden die verschiedenen SLAM-Varianten nach dem gewählten Lösungsansatz. Auf Basis der Ansätze gibt es wiederum verschiedene Entwicklungen, die diese erweitern, verbessern oder besser an bestimmte Einsatzzwecke anpassen. Diese einzelnen Entwicklungen sind dann unter den jeweiligen Projektnamen bekannt.

Zu den bekanntesten Ansätzen zählen die nachfolgenden drei in Abbildung 2.14.

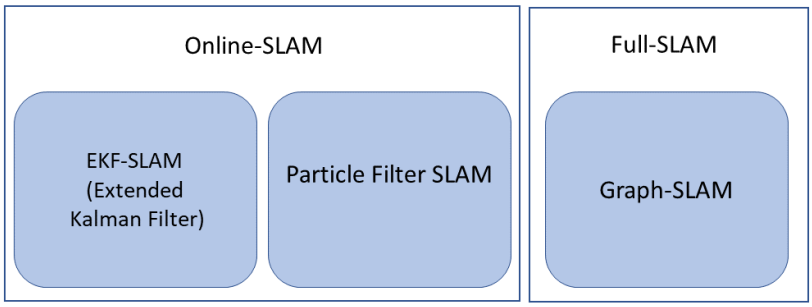

Abbildung 2.14.: Unterteilung der SLAM-Ansätze

Sie werden zunächst in zwei Kategorien eingeteilt. Und zwar in Online- und Full-SLAM. Diese Einteilung bezieht sich auf die Verwendung gesammelter Sensordaten und in der Vergangenheit liegender Positionen des Systems. Beim Online-SLAM wird die nächste Position lediglich aus aktuellen Sensordaten und der letzten Berechneten Position sowie der Karte ermittelt. Dabei wird angenommen, dass die letzte Positionsberechnung korrekt war. Im Gegensatz

dazu werden beim Full-SLAM alle in der Vergangenheit liegenden Positionen des Roboters oder Fahrzeugs, also der komplett zurückgelegte Pfad inklusive Sensordaten verwendet. Bei unterschiedlichen Positionen entdeckte Landmarken existieren unabhängig voneinander und werden nicht wie beim Online-SLAM mit jeder neuen Berechnung miteinander verknüpft und diese Verknüpfung als abgeschlossener Prozess betrachtet. (vgl. Thrun u. a., 2005, S. 309 ff.)

Der Vorteil von Full-SLAM Lösungen liegt darin, dass neue Beobachtungen die Genauigkeit von in der Vergangenheit liegenden Positionsbestimmungen verbessern können und sich somit Rückwirkend die Genauigkeit der prozessierten Karte verbessert. Der Rechenaufwand von Full-SLAM steigt mit der Größe der Karte bzw. mit den vorhandenen Sensorwerten. Aus diesem Grund wird diese Lösung meistens offline Verwendet, um Karten aus vorhandenen, kompletten Datensätzen zu erzeugen. Für die Positionsbestimmung während des Einsatzes werden weniger rechenintensive Online-SLAM Varianten verwendet.

Graph-SLAM ist eine Full-SLAM Lösung, bei der in einem ersten Schritt aus den Sensordaten mögliche Positionen des Sensors (Knoten) mit Kanten verbunden und lose assoziiert werden. In weiteren Schritten findet eine Optimierung des Graphen statt, indem der Algorithmus einzelne Lösungen gewichtet und neue Sensordaten berücksichtigt, bis der Graph mit der besten (wahrscheinlichsten) Bewertung aufgelöst ist. Alle Daten werden in einer Informationsmatrix gespeichert. Graph-SLAM ist eine Anwendung, die auf Informationstheorie basiert. (Vgl. Thrun u. a., 2005, S. 340 ff.)

Im Gegensatz zum eben beschriebenen Ansatz sind EKF- und Particle Filter SLAM Lösungen für das Online-SLAM problem. Unter Verwendung des erweiterten Kalman Filters (EKF) können nur eigenschaftsbasierte Karten für die Positionsbestimmung verwendet werden. Um Position zu ermittel, integriert EKF-SLAM zwei Modelle, um die Unsicherheiten zu modellieren. Ein Bewegungsmodell, dass Ungenauigkeiten der Odometrie oder der Ausführung von Bewegungen berücksichtigt, und ein Beobachtungsmodell, welches die Ungenauigkeiten der Sensorik nachempfindet. Verändert der Roboter oder das Fahrzeug seine Position, wird mit dem Bewegungsmodell die neue Position mit einem Winkel- und Distanzfehler behaftet abgeschätzt. Als nächstes wird aufgrund dessen und der Ungenauigkeit in der Detektion von Landmarken eine Wahrscheinlichkeitsverteilung für die neue erwartete Position der bekannten Landmarken erzeugt. Im letzten Schritt werden die tatsächlichen Beobachtungen mit den abgeschätzten Positionen der Landmarken assoziiert und in einem Korrekturschritt die ungenaue Abschätzung der eigenen Position verbessert. In Abbildung 2.15 sind diese Schritte dargestellt. EKF-SLAM benötigt eine möglichst genaue Abschätzung der zukünftigen Position durch die Eingabe von Fahrbefehlen bei Robotern oder Winkelencoder bzw. Odometrie. (Vgl. Thrun u. a., 2005, S. 312 ff.)

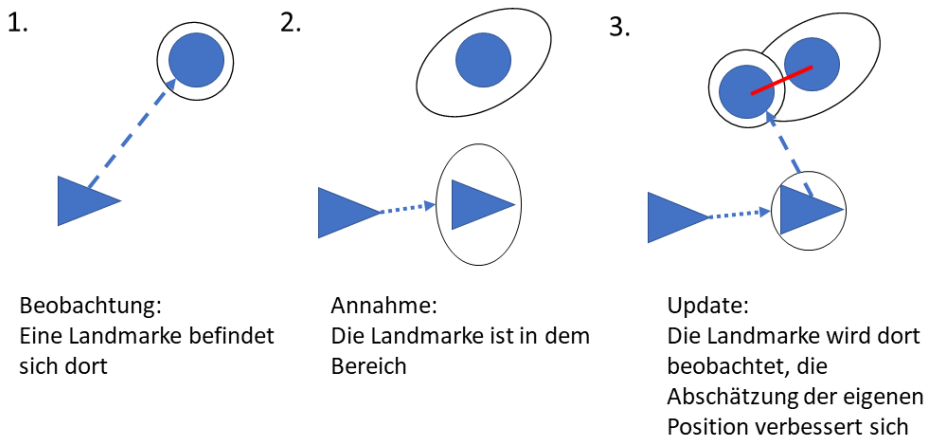

Abbildung 2.15.: Abstrahierter Ablauf der Positionsbestimmung beim EKF-SLAM

Partikel Filter basiertes SLAM kann sowohl mit eigenschaftsbasierten Karten, als auch mit Grid-Maps realisiert werden. Für beide Varianten gibt es Entwicklungen. Um Grid-Maps als Grundlage der Lokalisierung zu verwenden, werden Scan-Matching-Verfahren angewandt. Dies ist die Überlagerung und Verschiebung von Datensätzen, bis eine möglichst große Übereinstimmung erreicht wird.

Bei Partikel Filter SLAM werden in einem ersten Schritt Partikel erzeugt. Jeder Partikel stellt eine mögliche Roboter/Fahrzeug Pose dar. Sie werden mit einem Algorithmus erstellt, der eine Abschätzung über die mögliche neue Position macht und streut die Partikel in einer bestimmten Weise um diese Abschätzung in die Karte. In nachfolgenden Schritten werden neue Sensorwerte berücksichtigt, und jeder Partikel gewichtet, also die durch die Partikel-Position repräsentierte Welt mit aktuellen Sensorwerten verglichen. Zu unwahrscheinliche Posen werden entfernt und um die wahrscheinlicheren Posen werden neue Partikel erzeugt. Dieser Prozess wird auch Resampling genannt. Falsche Annahmen sterben dadurch nach und nach aus und es bildet sich eine Konzentration von Partikeln in immer besserer Annäherung an die tatsächliche Position, wie in Abb. 2.16 dargestellt. Die Karte wird dann anhand der wahrscheinlichsten Position aktualisiert bzw. erweitert (Vgl. Stachniss).

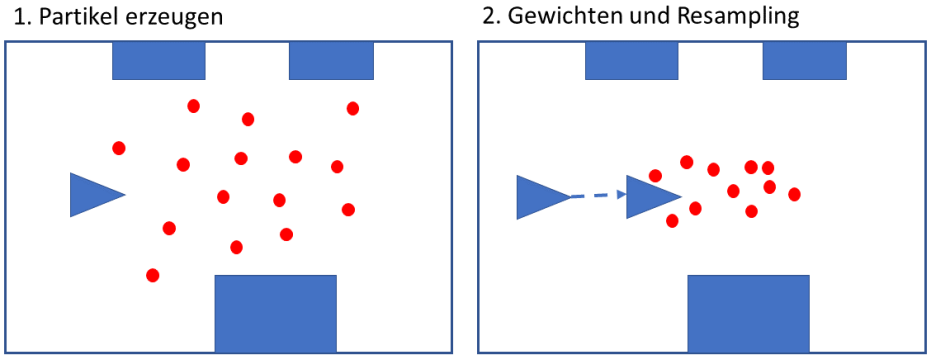

Abbildung 2.16.: Erzeugen von Partikeln und verbessern der Positionsannahmen durch Resampling (Quelle: In Anlehnung an Stachniss, 2013)

Jeder einzelne Partikel speichert nicht nur eine mögliche Position, sondern auch die komplette Karte aus seiner Sicht. Aus diesem Grund steigt der Rechenaufwand mit ihrer Anzahl stark. Um die Zahl geringer und die Streuung klein halten zu können, wird wie beim EKF-SLAM mit weiteren Informationsquellen versucht, einen möglichst geringen Fehler in der Abschätzung über die neue Position zu machen. Diese Quellen können wieder Befehle der Aktoren, Odometrie-Sensorik oder zum Beispiel ein Magnetometer zum verbessern des Winkelfehlers sein. (Vgl. Wolfram Burgard)

Die bisher genannten online-SLAM fähigen Ansätze benötigen über das Bewegungsmodell eine möglichst genaue Abschätzung der zukünftigen Position. Einen etwas anderen Weg geht „Hector-SLAM“, das von der Technischen Universität Darmstadt entwickelt wurde. Die Namensgebung rührt von dem „Team Hector“ genannten Entwicklungsteam her. „Hector“ steht für „Heterogeneous Cooperating Team Of Robots“ (vgl. teamhector, 2016).

Hector-SLAM wurde speziell für zweidimensionale Laserscanner mit hoher Umdrehungszahl entwickelt. Die Besonderheit ist, dass keine odometrischen Informationen von zusätzlichen Sensoren benötigt werden. Das Abspeichern der abgetasteten Umgebung geschieht ausschließlich in einer Grid-Map. Es handelt sich um eine Lösung, die für die zweidimensionale Positionsbestimmung nur einen Scan-Matcher-Algorithmus auf Basis des Gauß-Newton-Verfahrens verwendet. Die Funktionsweise des Algorithmus ist so zu verstehen, dass mit jedem Scan ein Minimierungsproblem gelöst wird. Die nachfolgende Gleichung 2.1 verdeutlicht das Minimierungsproblem. (vgl. Kohlbrecher u. a., 2011, S. 3)

$$\xi^* = argmin \sum_{n=1}^{n} [1 - M(S_i(\xi))]^2 \tag{2.1}$$

Die gesuchte Koordinatentransformation ξ^* ist für den zweidimensionalen Fall $\xi = (p_x, p_y, \Psi)$ und besteht aus der X- und Y-Position sowie die Ausrichtung auf der Karte in

Form eines Winkels. Mit dieser Transformation können aus den Daten des Sensors und der $S_i(\xi)$-Funktion die Weltkoordinaten der einzelnen Laserscan-Endpunkte bestimmt werden. Die Funktion $M(S_i(\xi))$ wiederum vergleicht den Endpunkt mit der vorhandenen Karte bzw. der Karte die beim letzten Scan entstanden ist. Sie liefert eine 1, wenn die Karte an der angegebenen Koordinate eine besetzte Zelle hat bzw. an dieser Stelle besetzt ist. Die optimale Lösung des Problems ist folglich die Transformation ξ, bei welcher jeder einzelne Punkt des Scans mit einem Punkt auf der Karte übereinstimmt, weil so die Summe über alle Punkte Null wäre.

Der Algorithmus versucht, mit jedem neuen Scan die Fehlerquadrate zu minimieren und die Transformation mit dem geringsten Fehler und somit der größten Übereinstimmung zu finden. Praktisch veranschaulicht zeigt Abbildung 2.17 die Funktion des Algorithmus.

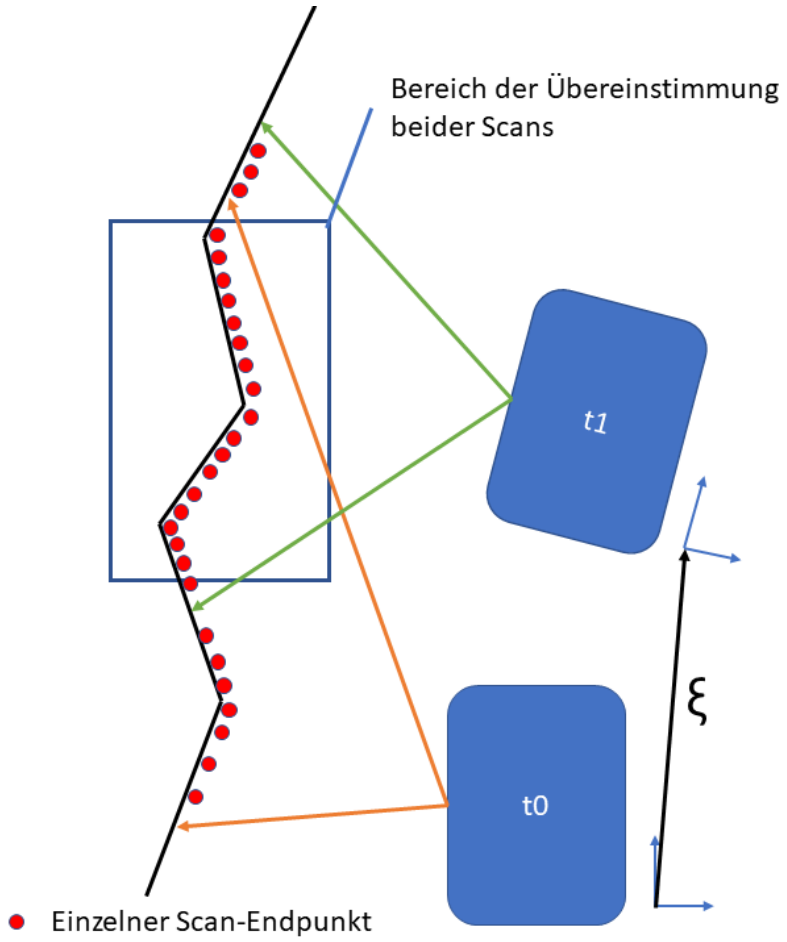

Abbildung 2.17.: Berechnen der Transformation durch Vergleich zweier Scans

Bewegt sich der Sensor durch bisher unbekanntes Gebiet, so wird mit jedem Scan-Vorgang

ein anderer Bereich abgetastet. Einzelne Ausschnitte wiederholen sich von einem Scan zum nächsten, da der gleiche Bereich abgetastet wurde. Dies setzt jedoch voraus, dass sich das Fahrzeug nicht zu schnell vorwärts bewegt und die Abtastfrequenz ausreichend hoch ist (z.B. 10-40 Hz). Der Algorithmus verschiebt nun die beiden Scans so zueinander, dass die Möglichkeit mit den meisten übereinstimmenden Punkten die Lösung ist. Die gefundene Verschiebung bzw. Transformation entspricht dann der Positionsveränderung des Sensors zwischen den beiden Scans.

Hector-SLAM hat die Möglichkeit, die Genauigkeit der Lokalisation zu erhöhen, in dem die Informationen weiterer Sensoren intelligent fusioniert werden. Dies wird über die Zusammenarbeit zweier Subsysteme geregelt (vgl. Abb. 2.18). Das erste Subsystem enthält die reine SLAM-Funktionalität und ermittelt die Position und Ausrichtung auf einer Fläche. Das zweite Subsystem basiert auf einer IMU (Inertial Mearurement Unit) und optional auf weiteren Sensoren wie GPS, Compass oder einem Altimeter. Dieses Subsystem verarbeitet die Sensordaten und die Daten des SLAM-Subsystems über einen erweiterten Kalman-Filter. Das Ergebnis der Filterung ist eine Einschätzung der aktuellen Position auf Basis des letzten SLAM-Ergebnisses und der Sensordaten. Diese Einschätzung kann wiederum den Scan-Matching-Prozess verbessern, in dem sie als Startposition für den Optimierungsprozess des Scan-Matchers dient. (vgl. Kohlbrecher u. a., 2011, S. 4)

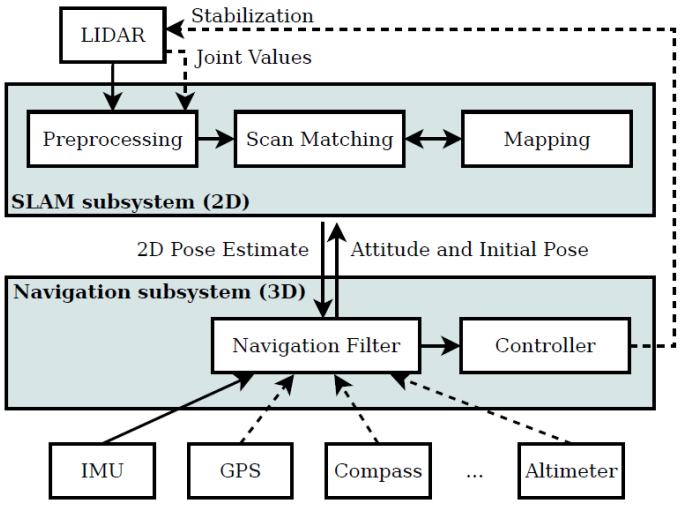

Abbildung 2.18.: Übersicht über die Subsysteme von Hector-SLAM (Quelle: Kohlbrecher u. a., 2011, S. 1)

3. Analyse und Design der Hardware

Dieses Kapitel behandelt die Analyse der vorhandenen Komponenten und Hardware. Nach einer Vorstellung des Aufbaus werden allgemeine Anforderungen an das Gesamtsystem formuliert und anschließend das Design des Gesamtsystems festgelegt. Danach werden die benötigten Eigenschaften eines Computersystems analysiert. Im Design-Teil folgt die Auswahl des Computers sowie die Planung der funktionalen Integration aller Systembestandteile in das Versuchsfahrzeug.

3.1. Vorhandene Komponenten

Die zur Verfügung stehende Hardware besteht aus einem Versuchsfahrzeug vom Typ Tesla Model S (vgl. tesla.com). Auf diesem befindet sich ein Dachgepäckträger der als Träger für die Sensorsysteme dient. Die Sensoren bestehen aus dem Velodyne HDL-32E Lidar (vgl. Kapitel 2.2.2) sowie dem Kamerasystem Ladybug 5+ (vgl. Kapitel 2). In der Abbdilung 3.1 ist das gesamte Fahrzeug und in Abbildung 3.2 eine Detailansicht des Dachträgers mit den Sensoren zu sehen.

Abbildung 3.1.: Vorhandener Versuchsaufbau mit Tesla Model S und den installierten Sensorsystemen auf dem Dachträger

Abbildung 3.2.: Ansicht des montierten Dachträgers mit dem Velodyne HDL-32 (vorne) und dem Ladybug 5+ dahinter

Die genaue Position des Lidar-Sensors ist mittig zur Querachse des Fahrzeugs und befindet sich in 1,88m Höhe sowie 2,64 Meter nach vorne versetzt bezogen auf das Ende des Hecks.

Das Kamerasystem wiederum befindet sich ebenfalls mittig bezogen auf die Querachse. die Objektive des Systems befinden sich in einer Höhe von 1,94m und das gesamte System 2,23m vor Fahrzeugende. Die Gesamtlänge des Model S beläuft sich auf 4,98 Meter.

Diese Werte basieren auf den vorhanden CAD-Daten des Zusammenbaus und sind mit der folgenden Abbildung 3.3 zu vergleichen.

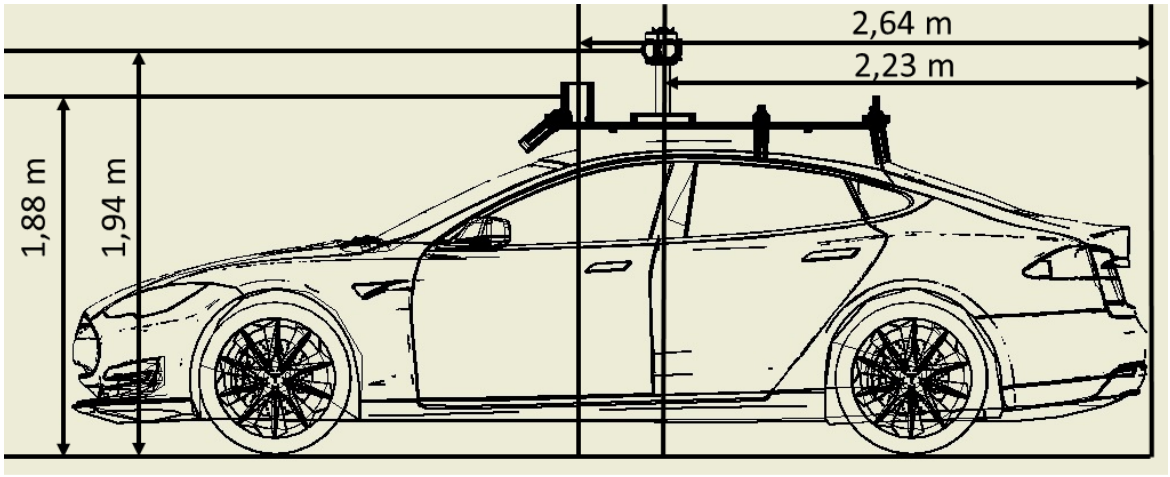

Abbildung 3.3.: Vorhandenes CAD-Modell des Versuchsaufbaus inklusive Bemaßung der Sensorposition

Weiterhin steht ein LiFe-PO4 Akku mit der Bezeichnung „Power Peak Pack 40“ des Herstellers Victron Energy zur Verfügung (Vgl. VictronEnergy, 2017). Dieser hat eine angegebene Kapazität von 40 Ah bei einer Nennspannung von 12,8 Volt. Er besitzt verschiedene Aus-

gänge für hohe Ströme von 150A und geringere Ströme mit einer Belastbarkeit von 30A Dauerstrom.

Abbildung 3.4 zeigt den Peak Power Pack und den Drucktaster, welcher über ein Kabel mit dem Akku verbunden ist und über den der Ausgang geschaltet wird.

Abbildung 3.4.: Peak Power Pack 40 Ah mit Drucktaster

3.2. Anforderungsanalyse des Gesamtsystems

Das System soll entsprechend der Art der Nutzung einigen Kriterien entsprechen. Dazu gehören Anforderungen, die durch den Umgang und die Bedienbarkeit, wie z.B. in einem Forschungsprojekt entstehen. Auch implizite Anforderungen durch die Umgebungsbedingungen werden berücksichtigt. In diesem Teil werden allgemein geltende Anforderungen an das System abgeleitet und formuliert. Spezielle Anforderungen an die Hardware des auswertenden Computers sind in Kapitel 3.4 zu finden. Anforderungen an die auswertende Software und Algorithmen werden in Kapitel 5 behandelt.

Das System soll Basis für die Auswertung und Aufzeichnung von Sensordaten sein. Dementsprechend ist eine Kompatibilität mit den vorhanden Sensoren zu gewährleisten. Da es sich bei dem System um ein Forschungsprojekt handelt, ist davon auszugehen, dass die Sensorkonfiguration im Verlaufe des Projektes verändert oder erweitert wird. Diese Veränderungen sollen durch allgemeine Reserven bezüglich Schnittstellen und Rechenleistung vom System getragen werden können.

Für die Inline-Auswertung und Aufzeichnung, also der Betrieb und die Auswertung der Sensoren während der Fahrt, ist als Spannungsversorgung der Akku vorgesehen. Als sinnvolle

Mindestdauer für die Einsatzzeit des Systems mit Hilfe des Akkus ist eine Stunde festgelegt.

Die Nutzung während der Fahrt bedingt auch, dass der Einbau oder die Integration von Komponenten gegen auftretende Erschütterungen oder Beschleunigungskräfte gesichert ist. Eine ausreichende Ladungssicherung für den Transport im oder am Auto ist vorzusehen.

Eine Bedienung des Systems während der Fahrt oder des Einsatzes ist ebenfalls eine Anforderung. Dabei ist aufgrund allgemeiner Sicherheitsbestimmungen das Bedienen des Systems vom Beifahrersitz oder von der Rückbank des Fahrzeuges zu ermöglichen.

Damit das System nicht nur im Fahrzeug genutzt werden, sondern auch im Labor konfiguriert und getestet werden kann, soll ein möglichst leichter Ausbau und Transport möglich sein. Zwei Personen sollen in der Lage sein, den Ausbau ohne Sensoren in maximal einer halben Stunde vollziehen zu können.

Das Forschungsfahrzeug befindet sich in einer Garage in Hamburg, Deutschland. Dementsprechend herrschen klimatische Bedingungen vor, die die Auswahl der Komponenten beeinflussen. Hierzu zählt beispielsweise die mögliche Bildung von Kondenswasser im Innenraum an kalten Tagen mit Temperaturen von beispielsweise 0 - 5°C.. Auch höhere Temperaturen von ca. 50°C können auftreten, wenn das Fahrzeug inklusive System im Sommer über längerer Zeit starker Sonneneinstrahlung ausgesetzt ist.

Die folgenden Aufzählungen fassen die genannten Anforderungen zusammen.

Funktionale Anforderungen:

- Auswertung und Aufzeichnung der vorhandenen Sensoren (Analyse in Kapitel 5)

Nicht funktionale Anforderungen:

- Einfache Bedienbarkeit von einem Passagierplatz aus
- Erweiterbarkeit (Schnittstellen, Leistungsreserven)
- Wartbarkeit und Handhabung der Hardware (Ausbau von zwei Personen in <30 Min.)
- Ladungssicherung während der Fahrt
- Das System soll in einem Temperaturbereich von 0-50°C arbeiten
- Einsatzdauer des Systems >1 Std. mit dem 40Ah Akku

3.3. Design des Gesamtsystems

Das System beseht aus den vorhandenen Komponenten und wird um einen Computer für das Erfassen und Auswerten der Daten erweitert. Sowohl der Computer als auch der Akku werden im Kofferraum untergebracht und mit der Sensorik auf dem Dachträger verbunden. Für die Bedienung vom Passagierbereich aus ist für das Ein- und Ausschalten des Computers und Akkus ein eigenes Bedienteil vorgesehen. Der Computer selbst wiederum wird über eine Fernwartungs-Funktion (Remote Desktop) und einer Netzwerkverbindung von einem anderen Computer bzw. Laptop ferngesteuert. In der Abbildung 3.5 ist eine schematische Darstellung des Systems inklusive Schnittstellen zu sehen.

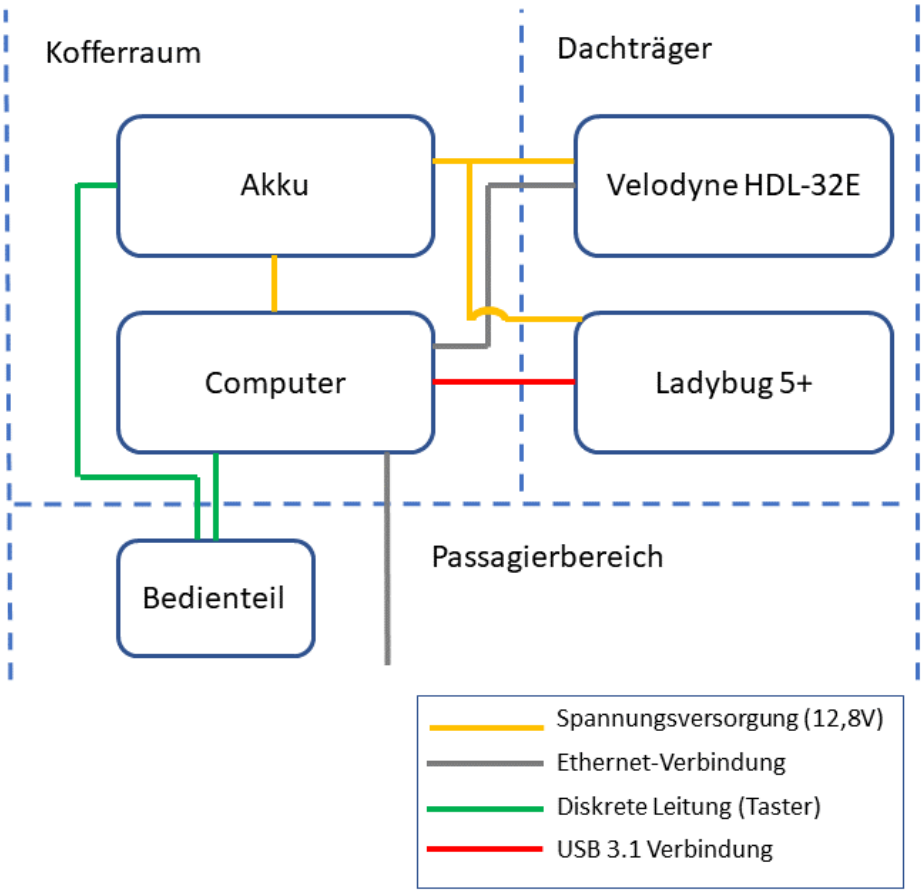

Abbildung 3.5.: Diagramm der Aufteilung und Schnittstellen des Gesamtsystems

3.4. Anforderungsanalyse und Auswahl des Computersystems

In diesem Abschnitt wird untersucht, welche Anforderungen das Computersystem erfüllen muss, um im Zusammenspiel mit den vorgestellten Sensoren in dem Forschungsfahrzeug funktional eingesetzt werden zu können. Anschließend werden einzelne Komponenten spezifiziert und der ausgewählte Computer vorgestellt.

Das System soll in der Lage sein, die Daten bildgebender Sensoren möglichst schnell auszuwerten, um beispielsweise die Positionsbestimmung mit SLAM-Algorithmen nutzen zu können. Ebenso ist eine gewisse Erweiterbarkeit vorgesehen. Erweiterungen beziehen sich sowohl auf die Schnittstellen, als auch auf die Leistungsfähigkeit, für die dadurch zusätzliche Reserven eingeplant werden müssen. Von diesem Gesichtspunkt aus betrachtet, ist eine aktuelle Computergeneration mit den besten Leistungswerten auszuwählen.

Eingeschränkt wird diese Auswahl jedoch durch die nicht funktionalen Anforderungen an das Gesamtsystem, die in den nächsten Absätzen quantisiert werden.

Zunächst ist die Anforderung an die Einsatzdauer des Systems zu erfüllen. Hierzu wird die maximale Leistungsaufnahme aller Geräte berechnet, um den Betrieb über den Akku für mindestens eine Stunde zu gewährleisten. Es wird davon ausgegangen, dass mindestens 10 % der Kapazität im Akku verbleiben, um eine Tiefentladung zu vermeiden. Das Peak Power Pack besitzt einen Schutz vor Tiefentladung, dieser ist jedoch in der Anleitung nicht genauer spezifiziert (vgl. VictronEnergy, 2017). In Gleichung 3.1 wird mit der zugrunde liegenden Nennspannung die entnehmbare Energie berechnet. Um das System eine Stunde zu betreiben darf die durchschnittliche Leistungsaufnahme nicht über ca. 460 Watt liegen.

$$40Ah * 0.9 * 12.8V = 460,8Wh \tag{3.1}$$

Die dadurch erlaubte Leistungsaufnahme des Computers liegt abzüglich der Versorgung für die Sensoren bei 435W. Der Velodyne HDL-32E ist mit einem Verbrauch von durchschnittlich 12 Watt angegeben (vgl. Velodyne, 2017) und der Ladybug 5+ mit 13 Watt (vgl. Flir, 2017b). Damit der Computer versorgt werden kann, ist ein Computernetzteil mit einer Eingangspannung im Bereich der Akkuspannung nötig. Der Spannungsbereich des Akkus ist im Datenblatt nicht angegeben. Messungen am geladenen und entladenem Akku haben 11-13,2 Volt ergeben. Bei der obigen Berechnung sind noch keine Wandlerverluste durch das Netzteil bedacht.

Als nächstes wird der Einbauort des Computers berücksichtigt. Der genannte Temperaturbereich von 0-50°C sowie mögliche Beschleunigungskräfte, Erschütterungen und Vibrationen

während der Fahrt beschränken die Auswahl der Systeme auf einen Industrie-PC ein. Diese erlauben je nach Hersteller einen Betriebsbereich von 0-55°C und zeichnen sich durch robuste und getestete Komponenten aus. Weiterhin wird bei der Montage von I-PCs darauf geachtet, dass keine Kabel scheuern und alle Komponenten in einer raueren (industriellen) Umfeld ihren Dienst zuverlässig verrichten. Da Flash-Speicher basierte Festplatten keine beweglichen Teile beherbergen sind sie grundsätzlich unempfindlicher gegen Erschütterungen als Festplatten mit beweglichem Lesekopf und rotierenden Platten. Flash-Speicher wird daher bevorzugt.

Für die Ladungssicherung soll der Computer über ein Montagesystem verfügen, so dass er fest angeschraubt werden kann.

Die Erweiterbarkeit des Systems lässt sich über die Flexibilität von Schnittstellen ausdrücken. Die meisten Computer haben bereits mehrere externe USB-Schnittstellen um Peripherie anzuschließen. In einem Forschungsprojekt sind unter Umständen jedoch auch andere Schnittstelle/Bussysteme gefragt um Daten zu übertragen (Can-Bus, UART etc.). Deshalb ist es sinnvoll wenn auf dem Mainboard mindestens zwei PCI Express Steckplätze für verschiedene Erweiterungskarten zur Verfügung stehen. Über die Erweiterungskarten lassen sich bei Bedarf diverse Schnittstellen oder Bussystem-Anbindungen realisieren. Um auch eine Grafikkarte nachrüsten zu können sollte ein PCI Express Steckplatz eine hohe Datenübertragungsrate bei 16 Lanes (PCI-E 16fach) unterstützen.

Der Velodyne-Lidar benötigt für die Datenübertragung eine Ethernet-Schnittstelle. Ebenso wie die Funktion des ferngesteuerten Desktops.

Die Ladybug-Kamera braucht für die Unterstützung der vollen Übertragungsrate eine USB 3.1 Schnittstelle. Im Lieferumfang befindet sich eine PCI-E Steckkarte, die zwei USB 3.1 bereitstellt. Somit wird ein zusätzlicher PCI-E Steckplatz benötigt.

Die folgende Auflistung fast die bisherigen Anforderungen zusammen:

- Leistungsaufnahme des Computers durchschnittlich <435 Watt
- Eingangsspannung des Computernetzteils im Bereich 11-13,2 Volt
- Computer nach industriellem Standard (I-PC)
- Gehäuse des Computers über Montagesystem fest anschraubbar
- Gehäuse möglichst kompakt, Abmessungen müssen in den Kofferaum passen (975mm x 1135mm x 520mm)
- Mindestens 1x PCI-E 16fach, 2x PCI-E
- 2 Ethernet-Anschlüsse

- System soll möglichst leistungsfähig/rechenstark sein
- unempfindliche Flash-Speicher Festplatten (SSDs) als Datenspeicher (durch Vibration und Stöße)

Im nächsten Schritt werden nun die leistungsbestimmenden Systemkomponenten gewählt, um die letzte Anforderungen zu quantisieren. Zum Zeitpunkt der Auswahl des Computers (Stand 01/2018) ist die 7. Generation der Intel Core-Prozessorreihe die bei den gefundenen I-PC Anbietern neuste und leistungsfähigste Prozessorreihe. Aus diesem Grund ist als Prozessor ein Intel Core 7700K vorgesehen. Er bleibt nach den Herstellerangaben mit einer TDP (Thermal Design Power) von 95 Watt weit unterhalb des gesetzten Limits.

Da mit dem Ladybug 5+ durch die insgesamt 30 Megapixel auflösenden Bilder binnen kurzer Zeit große Datenmengen übertragen werden können, sind zwei SSD-Festplatten mit je 1 Terabyte als Speichervolumen gewählt. SSD-Festplatten haben außer der Unempfindlichkeit noch einen Leistungsvorteil gegenüber herkömmlichen Festplatten in den Punkten Zugriffszeiten und Datenübertragungsraten.

Der Arbeitsspeicher wird mit 64 Gigabyte angefragt, da ein großer Arbeitsspeicher dem Computer den Umgang mit großen Dateien verbessert. Ein zu kleiner Arbeitsspeicher bremst durch häufige Ladeprozesse das System aus. Insbesondere bei der Ansammlung großer Datenmengen und Arbeiten mit verschiedenen Programmen, die Bildverarbeitung beinhalten, ist ausreichend dimensionierter Arbeitsspeicher von Vorteil.

Das Computernetzteil mit einer möglichen Eingangsspannung von 11-13,2 Volt wird mit 500 Watt dimensioniert. Zum einen damit Reserven für Erweiterungskarten zur Verfügung stehen und zum anderen, damit das Netzteil bei einer Leistungsspitze nicht in der Nähe des Limits betrieben wird.

Wieder zusammengefasst ergibt sich insgesamt diese Computerkonfiguration:

- Computernetzteil mit 500 Watt Ausgangsleistung bei 12,8 V Nenneingangsspannung
- Computer nach industriellem Standard (I-PC)
- Gehäuse des Computers über Montagesystem fest anschraubbar
- Mindestens 1x PCI-E 16fach, 2x PCI-E
- 2x Ethernet-Anschluss
- Intel Core i7 7700K (Prozessor)
- 2x SSD-Festplatte mit je 1 Terabyte Speichervolumen
- 64 Gigabyte Arbeitsspeicher

Das ausgewählte System hat folgende Spezifikationen:

- Schnittstellen [RJ45]: 2x Ethernet Anschluss
- Schnittstellen [PCI] : 1x PCI-E 16fach, 3x PCI-E 4fach, 3 PCI
- Arbeitsspeicher [DDR4]: 64 GB
- Festplatte [SATA]: 2x 1 TB SSD
- Prozessor: Intel Core i7 7700k, 4x4,2 Ghz
- Netzteil : 500 Watt, 12V Eingang

Das Gehäuse hat die Möglichkeit über die zum Lieferumfang gehörenden Metallwinkel fest mit einer Struktur verschraubt zu werden. Abbildung 3.6 zeigt das Gehäuse mit den Montagewinkeln. Es hat die Außenmaße von 200x320x480 mm.

Abbildung 3.6.: Das Gehäuse des Models Endevour MB, Quelle: janztec.com

3.5. Design des Einbaus

In diesem Teil wird das Konzept der Systemintegration in das Fahrzeug vorgestellt. Die elektrische Verkabelung wird dimensioniert und benötigte Halterungen sowie das Bedienteil konstruiert.

3.5.1. Konzept des Einbaus

Die Komponenten für den Einbau bestehen aus dem Akku und dem ausgewählten Computersystem. Außerdem wird ein Verteiler integriert, der alle Geräte über den Akku mit Strom versorgt. Der Einbauort, der hierfür den nötigen Platz bietet, ist der Kofferaum des Tesla Model S. Dieser ist in Abbildung 3.7 mit Bemaßung zu sehen. Die niedrigere Höhenangabe geht bis zur Abdeckung des Kofferraums.

Um der Anforderung der sicheren Verstauung gerecht zu werden, müssen die Komponenten im Kofferraum fixiert werden. Gleichzeitig spielt aber auch die Handhabbarkeit des Systems eine Rolle. Unter Berücksichtigung, dass zwei Personen möglichst einfach und schnell das Auto für den Einsatz mit dem System rüsten können, ist die Montage aller Komponenten auf einer Siebdruckplatte als Lösung gewählt. Siebdruckplatten sind beschichtete Multiplex-Holzplatten, die eine unempfindliche und wasserdichte Oberfläche haben und sich aus diesem Grund gut als Grundstruktur eignen.

Abbildung 3.7.: Kofferraum des Tesla Model S

Die Breite der Platte ist mit 900mm so gewählt, dass sie exakt in das schmaler werdende Stück Kofferraum passt. Dies fixiert die Platte gegen seitliches verschieben. Hebt man die Platte am Kofferraum Ende an, um sie herausheben zu können, darf sie nicht verkannten. Aus diesem Grund ist ihre Länge zu 1070mm gewählt, was an den kürzesten Stellen des Kofferraums zu ca. zwei Zentimetern Untermaß führt. Damit die Platte eine ausreichende Steifigkeit besitzt und als Grundlage für das Fixieren mit schrauben dienen kann, ist eine Dicke von 15mm vorgesehen.

Für eine gute Handhabung sind vier Griffe an den langen Seiten vorgesehen. Es ist angedacht, dass zwei Personen die sich seitlich am Kofferraum befinden die Platte an einem Griff zunächst anheben und etwas hervorziehen. Danach kann mit der anderen Hand der zweite Griff erreicht und die Platte sicher aus dem Auto herausgehoben werden.

Die Anordnung von Akku, Computer und Verteiler werden von den elektrischen Eigenschaften der Leitungen beeinflusst. Durch die relativ hohen Ströme (siehe Kapitel 3.5.2) sind kurze Leitungen vom Akku über den Verteiler zum Computer vorgesehen, da so der Spannungsabfall gering gehalten wird. Die Komponenten sollen entsprechend einen geringen Abstand von ca. 30 cm voneinander haben. In nachfolgender Abbildung 3.8 wird die geplante Anordnung der Teile ohne die Verkabelung auf der Holzplatte maßstabsgetreu gezeigt. Blau dargestellt ist der Akku. Links daneben ist der ausgewählte Computer und vor dem Akku ein Verteiler auf Basis von Stromschienen.

Die Velodyne-Interfacebox wird nach dem Einbau mit Klettband auf der Platte befestigt, um sie beim Ausbau schnell wieder Entfernen zu können, da die Box fest mit dem Velodyne-Lidar verbunden ist. Sie ist nicht Teil des CAD-Modells.

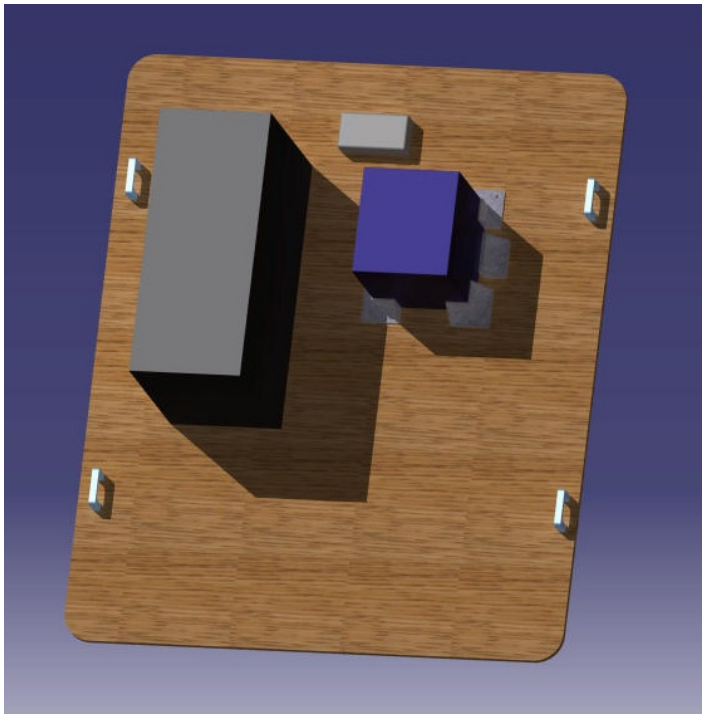

Abbildung 3.8.: Anordnung der Systemkomponenten auf der Holzplatte

3.5.2. Auslegung der elektrischen Verkabelung

Die Planung der elektrischen Verkabelung beinhaltet das Berechnen des Leitungsquerschnitts unter Berücksichtigung eines maximalen Spannungsabfalls durch die Leitungen unter Last. Ebenso wird ein Verteilertyp ausgewählt und die maximalen Leitungslängen, die den Berechnungen zu Grunde liegen festgelegt.

Zunächst wird eine Verbindung vom Hochstrom-Ausgang des Akkus zum Verteiler hergestellt. Danach erfolgt die Verbindung des Verteilers mit dem Computer und zusätzlich mit den Peripheriegeräten. Die nachfolgenden Berechnungen beziehen sich nur auf die Berechnung des Leitungsquerschnitts zum Verteiler und von diesem zum Computer, da hier die höchsten Ströme fließen. Die Sensoren sind mit ausreichend dimensionierten Leitungen geliefert.

Für die Berechnung des Mindestquerschnitts wird insgesamt eine Länge von 0,8 Meter angenommen. Diese besteht aus den beiden genannten Teilstücken. Die Verbindung von Verteiler und Computer wird aus praktischen Gründen mit dem gleichen Leitungsquerschnitt ausgeführt wie die Verbindung von Akku und Verteiler, da die abgehende Leistung durch die Peripherie am Verteiler gering ist.

Es wird der maximale Strom durch die Leistungsaufnahme des Computers berechnet und die der Sensoren hinzu addiert. Bei der Leistungsaufnahme des Computers wird bereits der Einbau einer Grafikkarte berücksichtigt. Diese setzt sich aus der TDP (Thermal Design Power) des Prozessors von 91 Watt (vgl. Intel) sowie dem geschätzten Verbrauch einer Grafikkarte von 200 Watt zusammen. Der Verbrauch des Chipsatzes des Mainboards wird vernachlässigt, da dieser viel kleiner relativ zu dem des Prozessors ausfällt. Mit dem Wirkungsgrad des Computernetzteils von 65 % (vgl. synoceantech) ergibt sich eine Leistungsaufnahme des Computers von: $291W/0.65 = 447,7W$ unter Volllast. Zu diesem Wert wird noch der Verbrauch der Sensoren addiert. Der Velodyne HDL-32e ist mit 24 Watt angegeben (vgl. Velodyne, 2017) und die Ladybug 5+ mit 13 Watt (vgl. Flir, 2017a). Weitere zukünftige Sensoren werden mit weiteren 20 Watt geschätzt. Somit ergibt sich die maximale Gesamtleistungsaufnahme zu: $448 + 24 + 13 + 20 = 505Watt$.

Aufgrund der Leistung sowie der Nennspannung des Akkus von 12,8 Volt und der Leitungslänge von 0,8 Metern ergibt sich für einen gewünschten Spannungsabfall von <2 % mit Hilfe des spezifischen Widerstands von Kupfer folgende Rechnung:

$$\rho = 0.0178\frac{\Omega * mm^2}{m} \rightarrow \rho = \frac{R * q}{l} \tag{3.2}$$

mit $R = \frac{U}{I}$ und $I = \frac{P}{U}$ ergibt sich der den Leiter durchfließende Strom zu $I = \frac{505W}{12,8V} \rightarrow I = 39,45A$.

Setzt man weiterhin in die Formel 3.3 alle Werte ein und rechnet mit dem maximalen Spannungsverlust von 2 % erhält man das Ergebnis aus Formel 3.4.

$$q = \frac{\rho * l * 2 * I}{\Delta U} \tag{3.3}$$

einsetzen der Werte liefert:

$$q = \frac{0,01786 * 0,8 * 2 * 39,45}{0,02 * 12,8} \rightarrow q = 4,4mm^2 \tag{3.4}$$

Der Leitungsquerschnitt sollte folglich unter der angegebenen Maximallast einen Querschnitt von $4,4mm^2$ nicht unterschreiten, wenn der Spannungsabfall über die Leitungen 2 % nicht überschritten werden soll. Da dieser Querschnitt nicht in der Normung vorkommt wird der nächst größere Querschnitt von $6mm^2$ gewählt. Hierdurch entstehen zusätzliche Leistungsreserven ohne einen höheren Spannungsabfall zu verursachen. Der Verlust über den Verteiler und der Übergangsstellen durch die Kabelschuhe wird aufgrund des geringen Widerstands bei fachgemäßer Montage vernachlässigt.

Damit der Verteiler einen geringen Übergangswiderstand hat, soll er auf ausreichend dimensionierten Stromschienen basieren und einen Strom von mindestens 460 Watt : 12,8 Volt = 36 Ampere dauerhaft verteilen können. Die Anschlüsse können dann durch Bolzen und Kabelschuhe realisiert werden.

3.5.3. Konstruktion der Halterungen und des Bedienteils

Der Computer besitzt bereits eine ausreichend Dimensionierte Halterung, um ihn mit der Holzplatte zu verbinden. Auch die Velodyne-Interfacebox und der 12 V Verteiler (siehe Kapitel 3.5.2) sind bereits für eine Schraubverbindung vorgesehen und haben entsprechende Bohrungen im Gehäuse. Für den Akku gibt es keine ab Werk vorgesehene Befestigungsmöglichkeit. Dennoch ist es wichtig ihn ausreichend zu fixieren, da er mit einem Gewicht von 8,6 Kg bei auftretenden Beschleunigungen kinetische Energie entwickeln und andere Komponenten Beschädigen kann.

Eine Kombination aus vier Eckhaltern und zwei Haltern mit einem Gurt soll die Verbindung realisieren. Die Eckhalter verhindern ein Verrutschen in X- und Y-Richtung. Die beiden Halter mit Gurt halten den Akku auf der Platte. Sollte man den Akku von der Platte lösen wollen, muss lediglich der Gurt gelöst werden und der Akku kann nach oben hin entfernt werden. Alle Halter werden mittels 3D-Druck hergestellt.

Die Abbildungen 3.9 und 3.10 zeigen die in Catia V5 konstruierten Eck- und Gurthalter und deren geplante Anordnung am Akku. Sie werden mit 10mm Holzschrauben mit der Grundplatte verbunden.

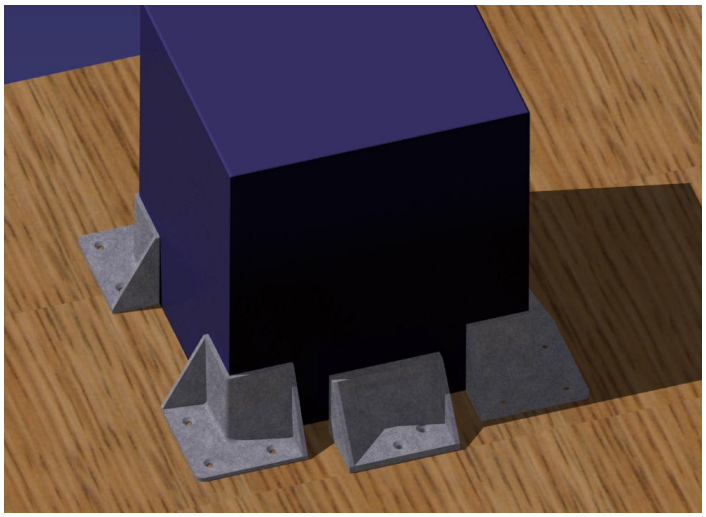

Abbildung 3.9.: Gesamtansicht der Halter für den Victron Energy Akku

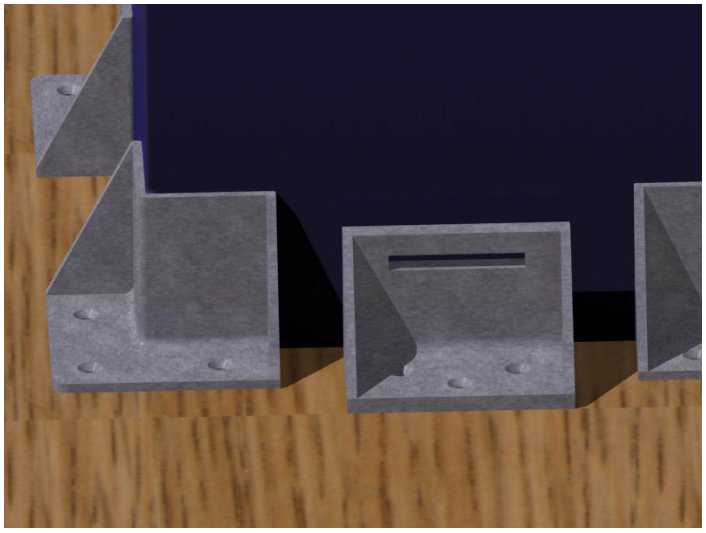

Abbildung 3.10.: Detailansicht des Gurthalters inklusive Aussparung für den Gurt

Das Bedienteil besteht aus einem Gehäuse, in das zwei Taster eingebaut sind. Ein Taster gehört zum Lieferumfang des Peak Power Pack Akkus und ist zum Schalten der verschiedenen Betriebsmodi gedacht. Zusätzlich verfügt der Taster über eine Led, die durch Blinksignale eine Rückmeldung bezüglich Status und Akkuladestand gibt. Der zweite Taster ist ein einfacher Schließer, der für das Ein- und Ausschalten des Computers gedacht ist. Das Kabel des Tasters vom Akku hat eine Länge von zwei Metern. Das Kabel des Computer-Tasters wird von der Länge entsprechend angepasst. Beide Taster werden auf einem kleinen Teil mit Beschriftung montiert. Es besteht aus einer Front mit den Aussparungen für die Taster sowie einer eindeutigen Beschriftung und verfügt über eine Zugentlastung mittels Kabelbinder. Das Gehäuse wird wieder als 3D-Druck realisiert. Die CAD-Konstruktion ist in Abbildung 3.11 zu sehen.

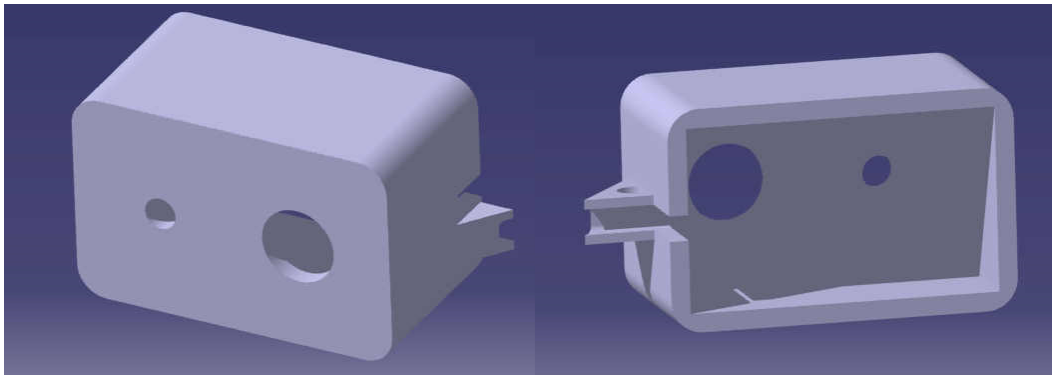

Abbildung 3.11.: Ansicht des Bedienelements für den Computer und den Akku

4. Realisierung der Hardware

Es wird die realisierte Lösung des geplanten Aufbaus für die Integration in das Versuchsfahrzeug vorgestellt.

4.1. Beschreibung des fertigen Aufbaus und der Anordnung der Komponenten

Die Siebdruckplatte ist mit den vorher berechneten Maßen realisiert. Die Ecken sind mit einem Radius von fünf Zentimetern versehen, um scharfe Kanten zu vermeiden. Dadurch wird der Ein- und Ausbau der Platte angenehmer und Schäden am Fahrzeug leichter vermieden. Für das Kabelmanagement sind Verdrahtungskanäle installiert. Diese Zeichnen sich im Gegensatz zu Kabelkanälen dadurch aus, dass die Wände nicht durchgängig sind, sondern aus einzelnen Lamellen bestehen. Durch herausbrechen einzelner Lamellen an gewünschten Stellen können die Kabel in bzw. aus dem Kanal geführt werden. Eine Draufsicht auf die Platte mit allen installierten Komponenten ist in Abb. 4.2 zu sehen und eine Ansicht nach installation im Kofferraum in Abbildung 4.1.

Abbildung 4.1.: Ansicht auf den Kofferraum mit der integrierten Platte

Der Kabelkanal hat eine Breite von 64mm und eine Höhe von 45 mm. Diese Größe bietet ausreichend Platz, um weitere Kabel und Leitungen hinzufügen zu können. Die Rückseite des Computers ist zur Kofferraum-Öffnung hin ausgerichtet, was das Stecken und Verlegen von Kabeln im Eingebauten Zustand erleichtert. Auf dieser Seite ist auch der Verteiler installiert. Der Akku und Computer sind über Kabelschuhe mit den M6-Bolzen des Verteilers verbunden. Die Peripherie, also der Velodyne-Lidar und die Ladybug-Kamera sind mit Kabelschuhen auf kleineren M3-Bolzen befestigt. Am anderen Ende sind die zweiadrigen Leitungen der Sensoren als Hohlstecker ausgeführt, die direkt mit den Hohlbuchsen der Sensoren kompatibel sind.

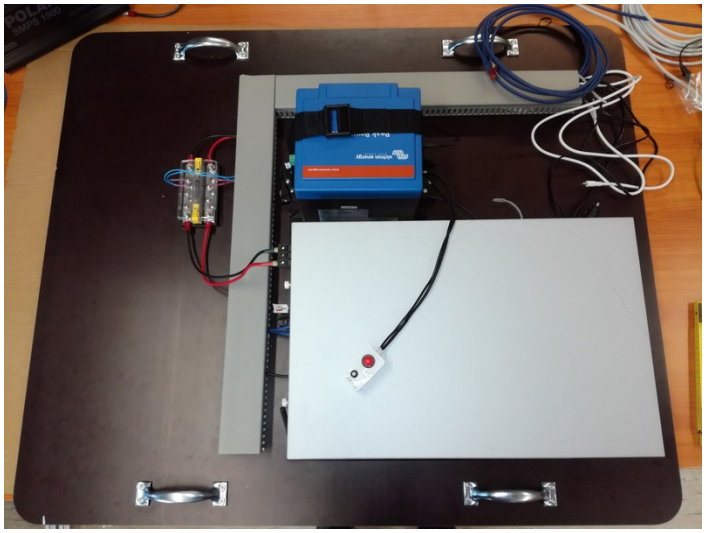

Abbildung 4.2.: Ansicht von oben auf die fertige Konfiguration der Platte

Auf dem Computer ist das Bedienteil zu sehen. Seine Leitungen werden vom offenen Ende des Kabelkanals über die Rückbank in den Passagierraum geführt, Ebenso wie das Ethernet Kabel für die Verbindung des Computers mit einem Laptop für den Fernzugriff. An dieser Position werden alle übrigen Verbindungen mit den Sensoren des Dachträgers hergestellt.

5. Analyse und Design der Software Implementierung

Die Software für die Auswertung und Aufzeichnung der Sensordaten besteht aus mehreren Ebenen. Die Oberste und abstrakteste Ebene stellt das Betriebssystem (Windows, Linux) dar. Eine Ebene darunter befindet sich als Softwareplattform ein Framework, dass als Umgebung für die Algorithmen sowie ihre Bedienung und dient. Schließlich die unterste Ebene, in der die Algorithmen in einer Hochsprache als Quellcode realisiert sind.

Das Nutzen einer vorhandenen Softwareplattform bzw. eines Frameworks für die Realisierung der Funktionen ist aus verschiedenen Gründen sinnvoll. Es bietet eine feste Grundstruktur mit zum Beispiel Werkzeugen für das Kompilieren von Quellcode als auch Kommunikationsstandards für erzeugte Programme. Visuelle Hilfsmittel für die Anzeige von Sensorwerten oder sogar ganze Robotersimulatoren sind in verschiedenen Frameworks enthalten. Der große Vorteil liegt aber darin, dass man eine nutzbare Arbeitsumgebung hat und die eigentliche Problemstellung fokussieren kann, ohne sich zunächst eigene Standards definieren zu müssen.

In Abbildung 5.1 ist die Struktur der Gesamtsoftware dargestellt. Das Framework nutzt das Betriebssystem wie andere Programme als Umgebung. Die gewünschten Funktionen werden als Teil des Frameworks realisiert und von diesem organisiert.

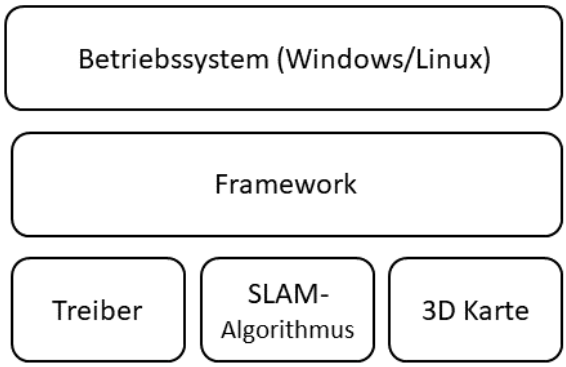

Abbildung 5.1.: Aufbau der Software aus Betriebssystem, Framework und Programmbeispielen innerhalb des Frameworks

In diesem Kapitel werden die Anforderungen an die Softwareplattform bzw. das Framework beschrieben. Danach erfolgt anhand der Anforderungen die Festlegung auf eine Softwareplattform, da diese im weiteren Verlauf als Grundlage der weiteren Implementierung genutzt wird.

Anschließend wird eine Anforderungsanalyse der zu realisierenden Funktionen und Auswahl benötigter Software-Module für die Nutzung mit dem Framework durchgeführt. Auf Basis dieser wird das Konzept der Sensorauswertung erstellt.

5.1. Anforderungsanalyse der Softwareplattform

Die Hardware-Schnittstellen sind durch den Computer realisiert. Die genutzte Software-Plattform soll eine Kommunikation mit den angeschlossenen Sensoren ermöglichen. Somit ist es von Vorteil, wenn die Plattform möglichst viele Gerätetreiber unterstützt und sowohl aktuelle als auch zukünftige Sensoren softwareseitig integriert werden können.

Dadurch, dass im Verlauf des Gesamtprojekts vorgesehen ist weitere Sensoren hinzuzufügen, soll die Plattform flexibel und erweiterbar sein. Im Anwendungsfall der universitären Forschung arbeiten mehrere Personen an verschiedenen Projekten mit diesem System. Zum Teil mit verschiedener Sensorik, aber auch die selben Ressourcen nutzend. Für diesen Fall ist ein modularer Aufbau der Software von Vorteil. Gerätetreiber und aufeinander aufbauende Programme sollen gemeinsam genutzt werden können. Außerdem ist es der Projektarbeit förderlich, wenn eigene Module seperat entwickelt und unabhängig voneinander getestet werden können.

Die Plattform soll nach Möglichkeit auf einem den Projektmitarbeitern bekanntem Betriebssystem aufbauen, um die Einarbeitungszeit zu verringern. Ebenso erleichtert das Verwenden bekannter Programmiersprachen wie Python/C++/C die Entwicklungsarbeit und eventuell bereits entwickelte Programme vorhergehender Arbeiten können integriert werden. Auch das Integrieren bestehenden Codes, wie zum Beispiel spezieller Bibliotheken als Werkzeuge oder ganzer Software-Module für bestimmte Aufgaben soll ebenfalls möglich sein.

Eine weitere Anforderung ist die freie Nutzung der Software unter einer Open-Source-Lizenz. Dies erleichtert den rechtlichen Umgang sowohl bei Veröffentlichungen als auch bei der Nutzung der Software. Insbesondere ist auch der Quellcode von Open-Source-Programmen einseh- und veränderbar und somit flexibler an die eigenen Bedürfnisse anpassbar. Ein weiterer Grund für die Nutzung von Open-Source-Software ist die kostenlose Verwendung.

Die Anforderungen an die Plattform lauten zusammengefasst:

- Unterstützen vorhandener Sensoren durch Gerätetreiber

- Nachträgliches Einbinden weiterer Sensoren möglich
- Modularer Aufbau, unabhängiges Entwickeln und Testen einzelner Programme
- Bekanntes Betriebssystem als Basis
- Bekannte Programmiersprachen wie Python/C/C++
- Viele Software-Module für die eigene Nutzung verfügbar
- Open-Source-Lizenz

5.2. Auswahl der Softwareplattform

Es gibt verschiedene Softwareplattformen bzw. Software-Frameworks, die sich mit dem Thema der Verarbeitung von Sensordaten oder Ansteuern von Aktorik befassen. Sie bedienen häufig das komplexe Themengebiet der Robotik und verfügen über verschiedene Softwarepakete, um unterschiedliche Arten von Robotern zu simulieren oder realisieren. Auch können Softwarepaket mit nützlichen Algorithmen für die Positionsbestimmung oder Kartierung für die Frameworks existieren, was sie für das autonome Fahren besonders interessant macht. Die folgend genannten Plattformen sind im Bereich der Sensorauswertung die im Internet subjektiv am häufigsten diskutierten und wurden für die Eignung als Plattform für dieses Projekt genauer betrachtet.

- Mobile Robot Programming Toolkit (MRPT)
- Microsoft Robotics Developer Studio
- Carnegie Mellon Robot Navigation Toolkit (Carmen)
- Robot Operating System (ROS)

Das Mobile Robot Programming Toolkit ist ein C++ basierter Zusammenschluss verschiedener getesteter Bibliotheken, um die Navigation und Lokalisation von Robotern zu ermöglichen. Die Bibliotheken sind in einzelne funktionale Module unterteilt. (vgl. mrpt.org)

Microsoft's Developer Studio kann unterschiedliche Roboter mit verschiedenen Sensoren simulieren. Hauptsächlich kann jedoch nur LEGO Mindstorms NXT und Fischertechnik Hardware verwendet werden. Somit scheidet die Software für die Anwendung aus. (vgl. Microsoft, 2007)

Bei Carmen handelt es sich um eine Open Source Softwaresammlung um wichtige Funktionen wie die Lokalisation, Kartenerstellung und Routenplanung von Robotern umzusetzen. Es gibt eine eigene festgelegte Kommunikation für Carmen-Programme. Die Hardware-Unterstützung ist jedoch sehr begrenzt und das verwendete Lidar-System nicht teil der Unterstützung. (vgl. CARMEN)

Als Framework mit den meisten Softwarepaketen und besten Hardwareunterstützung hat sich das Robot Operating System (ROS) herausgestellt. Es erfüllt nahezu alle an das System gestellte Anforderungen. Es ist sehr flexible einsetzbar und bietet relativ viel Unterstützung durch aktive Diskussionsforen sowie verschiedene Artikel und Bücher. Auch, dass ROS bereits bei Industrie-Projekten eingesetzt wird, ist ein Argument für die Verwendung (vgl. rosindustrial.org). Es gibt Software-Pakete welche die vorhandenen Sensoren auslesen und die Daten weiterverarbeiten können. Der einzige Nachteile ist die relativ hohe Komplexität, die mit dem Funktionsumfang einhergeht. Ebenfalls nachteilig ist, das lediglich Ubuntu Linux als Plattform komplett unterstützt wird, dies wird in Forschungsprojekten jedoch häufig verwendet und ist dadurch entsprechend bekannt. (vgl. wikipedia.de, 2018)

Aufgrund der gesammelten Informationen wird das Robot Operating System als Basis der Software-Implementierung genutzt.

5.3. Anforderungen an die Funktionalität der ROS Software-Pakete

Die Auswertungssoftware verarbeitet die Sensordaten, die mit Hardware-Treibern in ROS eingelesen werden, weiter. Die Software soll die Funktion des Systems herstellen, durch geeignete SLAM-Algorithmen im Zusammenspiel mit der Sensorik die aktuelle Position mit einer Genauigkeit von 0.02 Metern zu bestimmen und gleichzeitig eine entsprechend genaue dreidimensionale Umgebungskarte zu erzeugen. Der Wert für die Genauigkeit ist aufgrund der Anforderungen an das autonome Fahren gewählt. Diese tolerieren nur sehr geringe Werte für die Abweichung des Fahrzeug von der eigenen Fahrspur. Insbesondere schmalere Fahrstrecken und das Berücksichtigen anderer Verkehrsteilnehmer führen dazu, dass die eigene Position im Zentimeterbereich genau bekannt sein muss.(Vgl. Rauch u. a., S. 3)

Da eine dreidimensionale Karte mit Hilfe der Positionen als Ergebnis des SLAM-Algorithmus erstellt werden soll, muss die Position und Ausrichtung entsprechend sechs Freiheitsgraden bestimmt werden. Drei Freiheitsgrade geben die XYZ-Position in einem festgelegten Koordinatensystem an und die restlichen Freiheitsgerade die Neigung bzw. Ausrichtung des Sensors über die Winkellage der drei Koordinatenachsen. Die Bestimmung der Position soll

zunächst nur durch die Daten aus dem Velodyne-Lidar erfolgen. Generell soll aber die Möglichekit bestehen, die Daten weiterer Sensoren zu fusionieren. Durch die Fusion der Lidar-Daten mit Inertialsensoren und Informationen eines GPS-Empfängers könnte im Verlauf des Gesamtprojekts die Genauigkeit und Zuverlässigkeit erhöht werden. Im Nahbereich ist auch denkbar, durch Objekt bzw. Mustererkennung mit dem Ladybug-Kamerasystem die Positionsbestimmung zu verbessern.

Nach Möglichkeit arbeitet die Kartenerzeugung unabhängig von der Lokalisierung des SLAM-Ansatzes. Ein modularer Aufbau einzelner Funktionen wird aufgrund einer besseren Flexibilität angestrebt. So kann sowohl der Lokalisierungs-Algorithmus als auch das Programm für die Kartenerzeugung oder Darstellung bei Bedarf gegen eine Bessere alternative ausgetauscht werden. Die erzeugte Karte muss entsprechend der angestrebten Genauigkeit der Positionsbestimmung eine Mindestauflösung von 1 cm bei entsprechend genau bestimmten Position des Sensors und somit Fahrzeugs erreichen. Zu bedenken ist hierbei, dass eine künstlich verringerte Auflösung der Umgebungskarte mit dem Fehler der berechneten Position einen noch größeren Fehler erzeugt. Bei der späteren Navigation durch die Umgebung und auch bei der Positionsbestimmung anhand einer bereits erstellten Karte können so Schwierigkeiten entstehen. Abhängig von der Leistung des Kartenprogramms auf dem Computersystem wird zur Minimierung des Fehlers eine möglichst feine Auflösung einzelner Bildpunkte innerhalb der Karte angestrebt. Im Idealfall ist die Darstellung im Bereich von 1-2 cm möglich. In der Abbildung 5.2 ist die entstehende Ungenauigkeit der Karte durch Auflösungsvermögen und Positionierungsfehler für den zweidimensionalen Fall Dargestellt. Die Ansicht zeigt den Blick von oben auf den Sensor, welcher in der Ebene den Raum abtastet. Der Positionsfehler setzt sich aus der Ungenauigkeit des Velodyne-Lidar und ungenauen Ergebnissen des SLAM-Algorithmus zusammen. Der Velodyne HDL-32 hat eine angegebene Genauigkeit von +/- 2cm (vgl. Velodyne, 2017). Idealerweise sind die Ergebnisse nur dadurch limitiert.

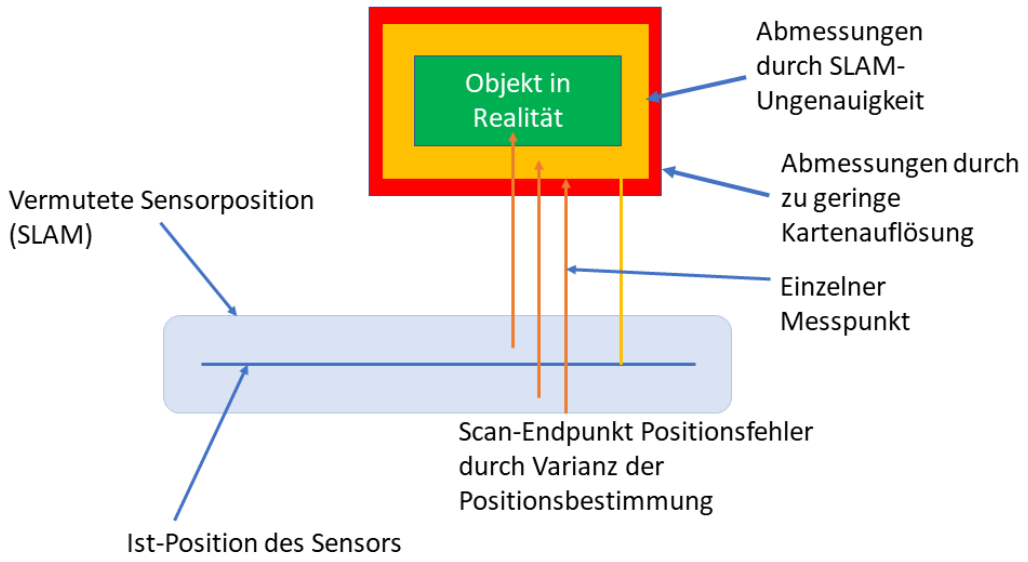

Abbildung 5.2.: Einfluss der Positionsbestimmung und Auflösung der Karte

Eine weitere Anforderungen an das Programm für das Erstellen und Speichern der Umgebungskarte sind die Nutzbarkeit und Zugriff auf die Daten. Es muss eine Schnittstelle bzw. Bibliothek existieren, mit dessen Hilfe andere Programme auf Teile der Karte zugreifen und verändern können. Grund hierfür kann das aktualisieren der Kartendaten bei baulicher Veränderung als auch das hinzufügen dynamischer Objekte in die statische Umgebungskarte sein. Der Speicherbedarf einer hochauflösenden Digitalen Karte spielt ebenfalls eine wichtige Rolle. Werden größere Areale vermessen, soll die Karte vom System weiterhin gut handhabbar sein und sich vom Speicherbedarf im Gigabyte-Bereich befinden. Als größeres Areal wird zum Beispiel eine große Tiefgarage mit einer Fläche von 500x500 Metern Fläche angesehen. Die Nutzung des Arbeitsspeichers soll ebenfalls effizient sein und die 64 Gigabyte des verwendeten Computersystems nur zu einem möglichst geringen Anteil nutzen, um das Betriebssystem und andere Programme nicht zu verlangsamen bzw. die Anforderungen an die Genauigkeit und Latenz nicht nur in Karten mit geringem Volumen zu erfüllen.

Die ROS-Programme sollen mit dem ausgewählten Computersystem die Position möglichst in Echtzeit bestimmen. Dies ist für die autonome Navigation und Routeplanung wichtig und insbesondere wenn auf sich schnell verändernde Umgebungsbedingungen durch Straßenverkehr oder ähnliches reagiert werden soll.

Eine durch SLAM erstellte Karte soll nicht nur binär verfügbar sein, sondern über ein Nutzerinterface visuell dargestellt werden können. Dies erleichtert das schnelle Beurteilen der Qualität und Plausibilität der Karte durch die SLAM-Algorithmen.

Nachfolgend sind die Anforderungen an die Auswertungssoftware zusammengefasst:

- Positionsbestimmung auf zwei Zentimeter genau
- Latenz in der Positionsbestimmung möglichst gering (Echtzeit)
- Positionsbestimmung in 6 Freiheitsgraden
- Sensorfusion der Lidar-Daten mit anderen Sensoren möglich
- Auflösungsvermögen der dreidimensionalen Karte 1 cm
- SLAM-Algorithmus und Kartenprogramm modular/austauschbar
- einfacher Zugriff auf die Daten der Karte (Bibliotheksfunktionen)
- Karte ist während eines SLAM-Einsatzes visualisierbar
- Programme sind möglichst effizient und ressourcenschonend
- Erzeugen größerer Karten möglich (Bereich von 500x500x500 Metern)

5.4. Auswahl der ROS-Pakete für die Datenauswertung

Die Auswahl der ROS-Pakete basiert auf Nutzererfahrungen verschiedener ROS-Diskussionsforen und vorliegenden Informationen aus verschiedenen Berichten und Artikeln.

Als Programm, dass den SLAM-Algorithmus beinhaltet, wurde Hector-SLAM ausgewählt. Es ist als Open-Source Programm verfügbar und verarbeitet Sensordaten eines Laserscanners. Es ist speziell für die hohen Abtastraten neuerer Laserscanner geeignet (bis 40 Hz) und arbeitet mit diesen gut zusammen. Hector-SLAM soll nur relativ wenig Rechenleistung benötigen. Der Nachteil ist jedoch, dass nur zweidimensionale Lidar-Daten verarbeitet werden können und die Positionsbestimmung nur 3 Freiheitsgraden in einer Ebene entspricht (vgl. Grundlagen Kapitel 2.3.2). Durch einige Annahmen und Erweiterungen die in dem nächsten Kapitel 5.5.1 der Konzeptvorstellung und Realisierung (Kapitel 6) erläutert werden, eignet sich Hector-SLAM dennoch sehr gut.

Für die dreidimensionale Kartenerzeugung eignet sich das ROS-Paket OctoMap. Es ist ebenfalls ein Open-Source Programm und auf mobile robotische Anwendungen zugeschnitten. Wie in dem Grundlagenkapitel 5.5.1 genauer beschrieben, ist OctoMap im Vergleich mit anderen Kartenerstellungs-Programmen sowohl ressourcenschonend als auch sehr flexibel. Die Flexibilität ist durch die Erweiterung und Ergänzung der erstellten Karte mittels

entsprechender ROS-Parameter und C++ Bibliotheken gegeben. OctoMap kann die Positionsinformationen eines SLAM-Programms und dreidimensionale Lidar-Sensordaten verarbeiten, was in Zusammenhang mit Hector-SLAM eine Grundvoraussetzung ist. Die Auflösung der Kartendarstellung eines Voxels kann bis zu $1cm^3$ betragen, was ebenfalls die Voraussetzung erfüllt. Durch ein Plug-In lässt sich die erstellte Karte in dem ROS-Integrierten Visualisierungs-Programm Rviz mit verschiedenen Einstellmöglichkeiten darstellen.

Anschließend sind noch die Hardware-Treiber zu erwähnen. Für den Velodyne-Lidar sind diese für ROS unter der Kategorie „ros-drivers/velodyne“ über GitHub als download verfügbar. Die Treiber sind in der Lage, die über die Ethernet-Schnittstelle empfangenen Daten einzulesen und in verschiedene ROS eigene Datenformate für die Kommunikation der ROS-Pakete umzuwandeln. Es gibt nur ein ROS-Paket für Velodyne-Treiber, wodurch keine Auswahlmöglichkeit besteht (Stand 03/2018). Das Paket enthält verschiedene Unterprogramme („Nodes“) für das Erzeugen der einzelnen Nachrichten-Typen. (vgl. wiki.ros.org, 2016b)

Für die Ladybug-Kamera existiert kein eigener Treiber für ROS. Da die Kamera aber ein Protokoll auf Basis des IEEE 1394 Digital Camera (IIDC) protocol verwendet, können mit einem allgemeinen ROS-Kameratreiber Bilder Empfangen werden. Das Paket, dass den Treiber enthält heißt “camera1394“. Da es sich um einen allgmeinen Treiber handelt, sind keine speziellen Funktionen wie eine Bildfeldentzerrung oder das Erstellen von Panoramaaufnahmen verfügbar. (Vgl. wiki.ros.org, 2016a)

5.5. Design der Softwareauswertung

Nach einer Übersicht des Konzepts für die Umsetzung der SLAM-Funktion mit den ausgewählten ROS-Paketen folgt eine detaillierter Darstellung. Sie enthält die benötigten Verknüpfungen zwischen den Paketen und beschreibt die Zusammenhänge der Kommunikation der einzelnen Programme sowie eine Darstellung der benötigten räumlichen Transformationen der Sensordaten des Velodyne-Lidars.

Nach dem Konzept der SLAM-Funktionalität wird die Vorgehensweise der Bilderfassung des Ladybug 5+ in ROS am Ende des Kapitels erläutert. Um eine spätere Erweiterung der Positionsbestimmung durch Fusion der Velodyne-Daten mit den Bilddaten der Kamera zu ermöglichen, soll die Bilderfassung- und Anzeige gleichzeitig mit dem SLAM-Algorithmus funktionieren.

Um die Konzepte verständlich zu machen, wird an dieser Stelle kurz auf die Terminologie der Kommunikation von ROS eingegangen. ROS ist ein System, dass viele kleine Unterprogramme organisiert um eine Gesamtfunktion zu erreichen. Diese kleinen Programme werden Nodes und Nodelets genannt. Nodes bezeichnen einen einzelnen Prozess, der von ROS gestartet wird und der meistens genau eine Funktion übernimmt. Um die Gesamtfunktion herzustellen, müssen die einzelnen Nodes miteinander Kommunizieren und sich zum Beispiel Ergebnisse von Berechnung etc. weiterleiten. Dieses Weiterleiten ist in ROS sehr anschaulich dargestellt und Funktioniert über „Messages“ und „Topics“ (Nachrichten und Themas). Berechnet eine Node zum Beispiel eine Positionsangabe, die von einer anderen Node benötigt wird, dann sendet die Node ihre Angaben an ein spezifisches Positions-Topic. Die andere Node kann die Nachrichten über den Topic-Namen abonnieren und erhält fortan die Daten, ohne die genaue Herkunft kennen zu müssen. Ein ROS-Paket kann aus vielen verschiedenen Nodes bestehen und verschiedene Topics enthalten. (Vgl. O’Kane, 2013, S. 24 f.)

5.5.1. Zusammenfassung des SLAM-Konzepts

Die Softwareauswertung besteht Modular aus verschiedenen ROS-Paketen, die miteinander zusammenarbeiten und sich über den in ROS Integrierten Kommunikationsstandard mit den geforderten Informationen versorgen. Die angedachte Strategie, die verfolgt wird, sieht einen Informationsfluss und Datenverarbeitung von den Rohdaten des Velodyne HDL-32E hin zur fertiggestellten dreidimensionalen Karte vor.

Das erste Paket besteht aus den Hardware-Treibern, um die Lidar-Daten zu erfassen und in das richtige ROS-Datenformat zu übertragen. Diese Daten werden dann für die Weiterverarbeitung mit dem nächsten Paket in ein entsprechendes Topic gesendet.

Als nächstes wird aus den nun in ROS verfügbaren Sensor-Daten eine Information über die relative Positionsveränderung des Sensors gewonnen.

Die Daten bestehen bisher nur aus einer Information über relative Position und Intensität eines reflektierten Signals des Velodyne Lidars in Bezug auf den Sensormittelpunkt. Aus diesen Informationen kann mittels eines ROS-Pakets, dass SLAM-Algorithmen beinhaltet, eine Veränderung der Position des Sensors in seiner Umgebung registriert werden. Die ermittelten Positionsdaten werden ebenfalls in einem entsprechenden Topic veröffentlicht. (vgl. Kapitel 2.3.2)

Mit Hilfe der ROS-Messages über die relativen Positionsänderungen und den zeitlich dazu passenden Rohdaten des Sensors kann dann ein Programm eine dreidimensionale Karte mit allen als besetzt markierten Punkten erzeugen und den gefahrenen Weg anzeigen. (vgl. Kapitel)

Die Abbildung 5.3 zeigt eine starke Abstraktion des anvisierten Datenflusses, um aus den Sensordaten eine Umgebungskarte zu erzeugen.

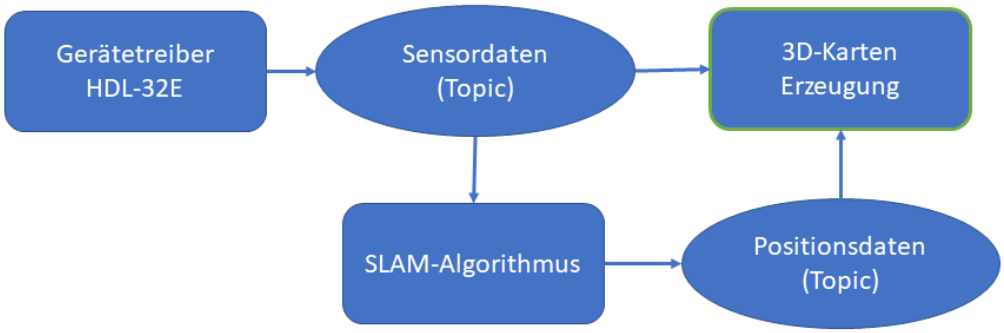

Abbildung 5.3.: Abstraktion des Datenflusses einzelner ROS-Pakete

5.5.2. Detaillierte Darstellung des SLAM-Konzepts

Die Detail-Darstellung des Systementwurfs geht genauer auf die Schnittstellen zwischen den einzelnen ROS-Paketen und deren Nodes ein. Zu den Schnittstellen zählt die Kommunikation über die Topics, als auch das Nutzen der Transformations-Funktionen von ROS, welche in diesem Abschnitt vorgestellt und in dem Nachfolgenden Kapitel 6 genauer erklärt werden.

Durch die Einschränkungen von Hector-SLAM, die Position nur zweidimensional ermitteln zu können, muss der Informationsfluss und die genutzten Funktionen erweitert werden, um aus den Daten dennoch eine dreidimensionale Karte erstellen zu können.

Die Annahme, die für eine dreidimensionale Positionsbestimmung gemacht wird, beruht auf der festen Montage des Velodyne-Sensors auf dem Dachträger des Fahrzeugs. Dadurch wird vorausgesetzt, dass sich der Abstand des Sensors zum Boden sich beim langsamen Fahren nicht oder nur wenig verändert. Kleine Unebenheiten des Bodens werden zwar somit vernachlässigt, die Auswirkungen auf die erstellte Karten sollten sich jedoch innerhalb des geforderten Toleranzbereichs befinden. Das erste Konzept vernachlässigt ebenfalls die Verkippung des Sensors gegenüber der Horizontalen. Zusammenfassend wird also angenommen, dass die Befestigung des Sensors auf dem Fahrzeug eine Bewegung entlang der Hochachse und die Rotation um die Längs- und Querachse des Fahrzeugs verhindert. Somit erhält man trotzdem der SLAM-Algorithmus nur die Position auf einer zweidimensionalen Ebene und die Ausrichtung um die Z-Achse berechnen kann eine Abschätzung über die Position und Ausrichtung im dreidimensionalen Raum. Um mit OctoMap eine dreidimensionale Karte erzeugen zu können ist es wichtig, dass die Lage des Sensors in sechs Freiheitsgeraden bekannt ist und diese Informationen für die OctoMap-Nodes in den Topics verfügbar sind. Nur mit Transformations-Nachrichten, in denen alle Freiheitsgerade mit definierten Zahlenwerten gefüllt sind, kann OctoMap eine Karte aufbauen.

Datenfluss der ROS-Nachrichten über die Topics

Die Rohdaten des Velodyne-Lidar werden von den Treibern verarbeitet und ein ROS-Nachrichtenformat umgewandelt. In der ersten Stufe der Verarbeitung erzeugt die „velodyne_node“ Nachrichten vom Typ „velodyne_msgs/VelodyneScan„ und sendet sie an das Topic „velodyne_packets“. Die Nachrichten enthalten die aufgenommenen Rohdaten des Sensors für eine Umdrehung des Sensorkopfes.

Die nächste Ebene konvertiert die Nachrichten mit den Rohdaten in ein Format, mit dem viele ROS-Programme arbeiten und 3D-Informationen der Umgebung erhalten. Durch die „velodyne_pointcloud“-Node, welche das Topic „velodyne_packets“ abonniert (in ROS „subscribed“ genannt), werden „sensor_msgs/pointcloud2“ Nachrichten in das „velodyne_points“-Topic gesendet. Die Nachrichten sind im englischen als „point cloud“, zu deutsch Punktwolke, bekannt. Sie zeichnen sich dadurch aus, dass die Position jedes einzelnen Punktes eindeutig durch Raumkoordinaten beschrieben ist, wohingegen die Rohdaten des Velodyne_Lidars Informationen über Winkel und Entfernung eines Endpunktes vom Sensor enthalten. (vgl. docs.ros.org) Die erzeugten „pointcloud2“-Nachrichten sind bereits im richtigen Format, um von den OctoMap-Nodes verarbeitet zu werden.

Um mit Hector-SLAM Positionsdaten zu erzeugen, werden zusätzlich Nachrichten vom Typ „LaserScan“ benötigt. Diese bestehen ähnlich den Rohdaten in den "velodyne_msgs/VelodyneScan“-Nachrichten aus Winkel- und Distanzinformation einzelner

Endpunkte, jedoch nur für den zweidimensionalen Fall. Folglich muss aus den dreidimensionalen Rohdaten des Velodyne-Lidar eine zweidimensionale Bildinformation extrahiert werden. Die in dem Velodyne Treiberpaket enthaltene „velodyne_laserscan“-Nodelet ist in der Lage aus den Pointcloud-Nachrichten einzelne Ringe des 32 Zeilen auflösenden Velodyne-Lidar in zweidimensionale „LaserScan“-Nachrichten umzuwandeln und in das „/scan“-Topic zu veröffentlichen.

Der angedachte benötigte Informationsfluss einzelner Nodes über die Nachrichten und Topics wird in der folgenden Abbildung 5.4 gezeigt. Die Rechtecke stellen wieder die verwendeten Nodes und die Ellipsen die Topics dar. Die beiden Topics „/tf“ und „/tf_static“ die von der „octomap_server“-Node benötigt werden sind Nachrichten, welche räumliche Transformationen enthalten und im nächsten Kapitel 5.5.2 genauer erläutert werden.

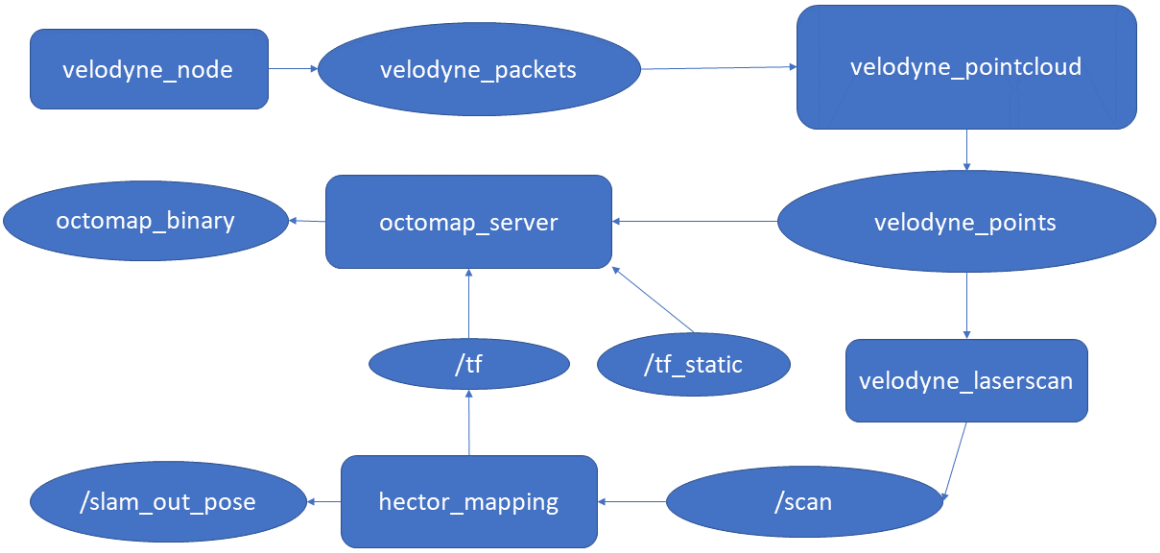

Abbildung 5.4.: Verknüpfung der ROS-Nodes über die Topics

Die Hector-SLAM-Node heißt „hector_mapping“ und erzeugt sowohl „geometry_msgs/PoseStamped“-Nachrichten mit Positionsinformationen und sendet sie an das „/slam_out_pose“-Topic als auch Nachrichten eines speziellen Typs, die Transformationen enthalten. Über die beiden „/tf“-Topics erhählt die „octomap_server“-Node Zugriff auf diese Daten und nutzt sie, um die „velodyne_points“ auf die aktuelle Position des Sensors im Raum zu transformieren. Die

Erzeugen benötigter Transformationen

In ROS gibt es für die Koordinatentransformation ein eigenes Software-Paket, dass Funktionen zur Unterstützung für das Managen von Transformationen bietet. Transformationen werden in einer leicht überschaubaren Baumstruktur organisiert, dem sogenannten „transform tree“. Mit diesem lässt sich die Verbindung zwischen den verschiedenen Koordinatensystemen verfolgen und nachvollziehen. Eine Transformation muss dabei immer von einem Eltern-System zu einem Kind-System verlaufen. (Vgl. http://wiki.ros.org, 2015)

Bezogen auf die Benutzung von OctoMap und Hector-SLAM bedeutet dies, dass ein Transformations-Baum mit teils festgelegten Koordinatensystemen in richtiger Reihenfolge verknüpft werden muss, um dem System eine korrekte Darstellung der einzelnen Messpunkte in einer Karte zu ermöglichen. Die Transformationen verlaufen im konkreten Fall von einem globalen Koordinatensystem über die Basis des Versuchsfahrzeugs hin zu der Position des Sensormittelpunkts auf dem Dachträger.

In der Übersicht, die man mit dem ROS-Werkzeug „rqt_graph“ erzeugen kann und welche die Nodes und Topics ähnlich Abbildung 5.4 zeigt, werden die Bezeichnungen der verwendeten Koordinatensysteme nicht angezeigt. Lediglich der Nachrichten- und Transformations-Typ wird dort angezeigt. Es gibt in dem Konzept zwei verschiedene Transformations-Topics mit den Bezeichnungen „/tf“ und „/tf_static“. Die Bezeichnungen geben Aufschluss darüber, dass das System über einen Transformations-Baum verfügt und dass die Transformationen sowohl dynamisch, also zeitlich veränderlich (/tf), als auch statisch und somit zeitlich unveränderlich (/tf_static) sind.

Die dynamischen Transformationsnachrichten werden von der hector_mapping-Node auf Basis des SLAM-Ergebnisses generiert und beinhalten dementsprechend die Informationen über die Verschiebung und Verdrehung des Koordinatensystems in einer Ebene, auf welcher sich der Sensormittelpunkt vom Velodyne-Lidar befindet. Für die OctoMap wären die Transformationen von Hector-SLAM unvollständig, da sie keine Informationen der Höhe und Verdrehung der Ebene in einem dreidimensionalem Raum, dem sogenannten „/map-frame“ enthält. Folglich müssen diese Daten durch weitere Transformationen hinzugefügt werden, bis OctoMap einen zu jedem Zeitpunkt vollständig definierten Aufenthaltsort des Velodyne-Lidars innerhalb des /map-Koordinatensystems hat.

An dieser Stelle kommen die vorher erwähnten statischen Transformationen ins Spiel. Durch die Annahme, dass die Montage des Sensors auf dem Versuchsfahrzeug drei Freiheitsgrade sperrt, kann eine fixe Transformation die fehlenden Daten bereitstellen.

Die Abbildung 5.5 zeigt die benötigte Baumstruktur der Transformationen und Bezeichnungen der Koordinatensysteme. Später kann die Ansicht der Struktur mit dem ROS-Werkzeug „view_frames“ vom implementierten System abgeleitet und für Debugging-Zwecke grafisch

dargestellt werden. Dies wird in dem Kapitel 6.3 der Realisierung benötigter Transformationen genauer betrachtet.

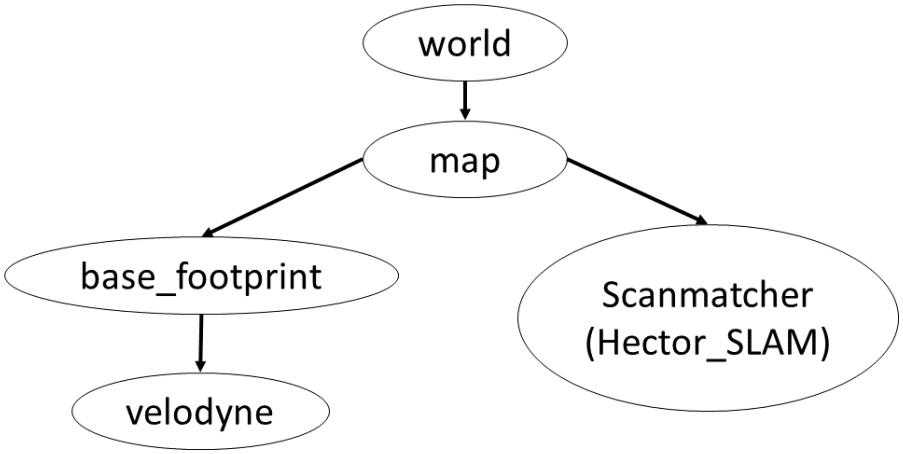

Abbildung 5.5.: Benötigte Baumstruktur der Transformationen

Das „world“-Koordinatensystem ist bei ROS immer als das Basis-Koordinatensystem vorhanden. Es hat kein Eltern-Koordinatensystem und alle weiteren Systeme sind Kinder-Systeme von diesem und befinden sich in dem Basis-System. Dies ist vor allem in einem Multi-Roboter-System sinnvoll, wenn mehrere Roboter jeweils in eigenen unabhängigen map-System unterwegs sind. Das world-System wird mit einer statischen Transformation, die keine Veränderung der Position enthält, in das map-System überführt, da OctoMap dieses Koordinatensystem für die Darstellung der Umgebungskarte nutzt. Die Daten, die OctoMap in Form der pointcloud2-Nachrichten bekommt, befinden sich zunächst im velodyne-Koordinatensystem. Mittels der statischen Rücktransformationen über das base_footprint-System und der dynamischen Rücktransformation vom scanmatcher- zum map-System werden die Transformationen für OctoMap vollständig. In der nachfolgenden Abbildung ist eine räumliche Darstellung des Transformationsbaumes zu sehen. Das base_footpring-System entspricht dem Mittelpunkt des Fahrzeugs mit Bodenkontakt. Von diesem aus führt eine statische Transformation zum Sensormittelpunkt auf dem Dachträger. Die Überführung des map-Frames in das base_footprint-System legt nur fest, dass sich dieses auf einer Ebene mit der Höhe Null befindet. Die Information, wo sich das base_footprint-System auf der Ebene befindet, wird durch die Rücktransformation vom scanmatcher-Frame ergänzt.

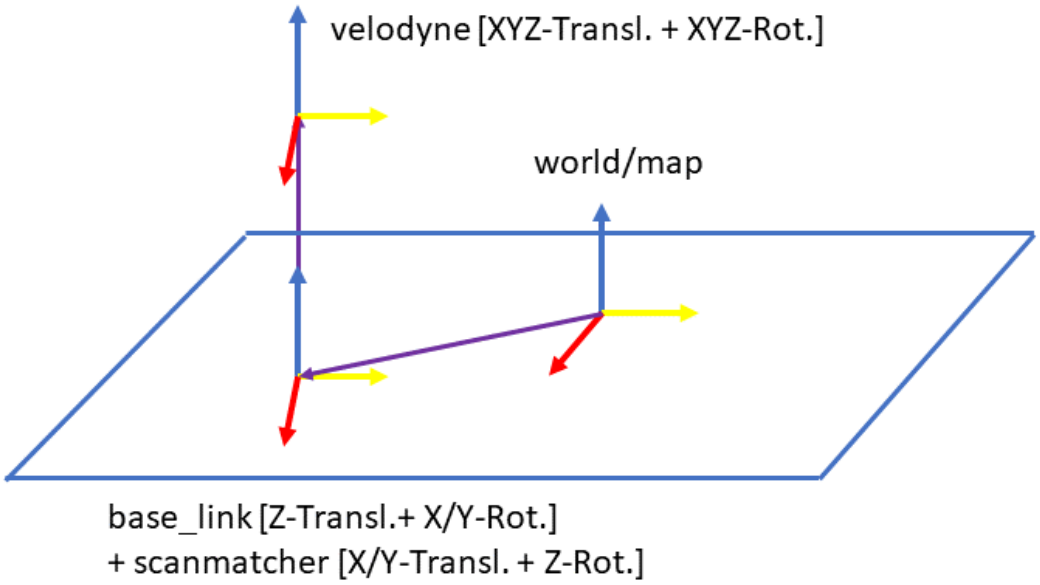

Abbildung 5.6.: Räumliches Ergebnis der Transformationen

5.5.3. Konzept der Datenerfassung mit der Ladybug 5+ Kamera

Die Ladybug 5 360°-Kamera wird über die zum Lieferumfang gehörende USB 3.1 PCI-Express Steckkarte, die in den Computer eingebaut wird, angebunden.

Die Kamera benötigt Linux-Hardwaretreiber und ein entsprechendes ROS-Paket, um die Daten in ROS anzuzeigen und verarbeiten zu können. Die Nodes innerhalb des ROS-Pakets "camera1394“, dass für die Integration der Kameradaten in ROS genutzt werden soll, erzeugen Nachrichten des Typs „sensor_msgs/Image Message“ und senden sie an ein Topic mit einer entsprechenden Kamerakennung. (vgl. http://wiki.ros.org/, 2015)

Es ist zu beachten, dass die Kamera einen Stream aus einzelnen Bildern sendet, die jeweils die Größe aller sechs Einzelbilder des Systems zusammengefügt haben. Ein übermitteltes Rohbild hat somit eine Auflösung von 2464*12292 Pixeln, was selbst mit JPEG8 Komprimierung zu einer Speichergröße von 30,3 MB (vgl. Flir, 2017b, S.60 ff.) führt. Unter Linux ist in den Einstellungen der Speicherpuffer auf 2 MB begrenzt und muss deshalb erweitert werden.

Das Speichern von Bilder-Streams soll parallel zu der Aufnahme von Lidar-Daten und anderen Sensordaten erfolgen können. Für das Erfassen von verschiedenen Sensornachrichten und abspeichern mit einem Zeitstempel gibt es in ROS das rosbag-Paket. Dieses besteht aus

einem Kommandozeilen-Werkzeug, mit dem man Nachrichten einzeln ausgewählter oder aller verfügbaren Topics als bag-Dateien abspeichern kann. Abgespeicherte bag-Dateien lassen sich für eine nachträgliche Auswertung mit diesem Tool abspielen, was dazu führt, dass die gespeicherten Nachrichten wieder an die Topics gesendet werden, die für die Aufnahme ausgewählt wurden. (vgl. wiki.ros.org, 2018)

6. Realisierung der Software Implementierung

In diesem Kapitel wird die Vorgehensweise der Implementierung der gewünschten Funktionen beschrieben. Es wird die Installation einzelner Software-Komponenten durchgeführt und Schritt für Schritt ein lauffähiges System implementiert und parametrisiert. Für eine im weiteren Verlauf der Forschung möglichst einfache Bedienung wird ein eigenes ROS-Paket angelegt, über welches die verschiedenen Unterprogramme und Werkzeuge im Verbund aufgerufen werden können.

6.1. Installation und Einrichtung von ROS

6.1.1. Download und Installation

Zu Beginn erfolgt die Installation von ROS Kinetic. Die Versionsbezeichnungen sind mit dem Alphabet aufsteigend. Die aktuellste Version von ROS ist die Version „Lunar Loggerhead“ (vgl. wiki.ors.org, 2017), erschienen am 23. Mai 2017. Die empfohlene Version ist jedoch Kinetic und am 23. Mai 2016 erschienen. Diese Version wird für die Arbeit mit ROS verwendet, da sie die verwendeten Software-Pakete unterstützt und von vielen Anwendern verwendet wird, wodurch online ein besserer Support gegeben ist.

ROS Kinetic benötigt die Ubuntu Linux Version 16.04 (Xeanial Xerus) oder 15.10 (Wily). Auf dem verwendeten Computersystem wird Version 16.04 verwendet, da dies die aktuellere Linux Version ist.(vgl. wiki.ros.org, 2017b)

Nach der Linux Installation wird die Installation von ROS vorbereitet. Download und Installation kann wie die von anderer Linux-Software über Shell-Befehle erfolgen. Hierzu muss in den Einstellungen „http://packages.ros.org/ros/ubuntu“ als erlaubte Quelle für Software hinzugefügt werden.

Danach wird mit dem Befehl „sudo apt-key adv –keyserver hkp://ha.pool.sks-keyservers.net:80 –recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116“ auf den

Keyserver zugegriffen und ein Authentifizierungsschlüssel angefordert. Dieser Schlüssel steht danach dem Paket-Management-System zur Verfügung.

Mit Hilfe des Befehls „sudo apt-get update“ kann nach dem Hinzufügen der ROS Paket Quelle der Index aller verfügbaren Pakete aktualisiert werden. Zu beachten ist, dass alle Quellen auf neue Pakete hin durchsucht werden, und nicht nur die für ROS verfügbaren.

Da das System nun über die Pakete informiert ist und der Authentifizierungsschlüssel bereitliegt, kann mittels Eingabe der Zeile „sudo apt-get install ros-kinetic-desktop-full“ in die Shell ROS heruntergeladen und installiert werden. Es gibt verschiedene Paket Varianten, welche unterschiedlich viel vorinstallierte Software, wie zum Beispiel für einfache Simulation, beeinhalten. Je nach verfügbarem Hardware-System (zum Beispiel Raspberry Pi) können schlankere Versionen von ROS, die einen kleineren Speicherbedarf haben, Sinn machen. (vgl. wiki.ors.org, 2017, S. 13)

6.1.2. Einrichten einer Arbeitsumgebung in ROS

Bevor ROS verwendet werden kann, muss „rosdep“ initialisiert werden. Dies ist ein Werkzeug, dass für die Kontrolle und Installation von Paket-Abhängigkeiten zuständig ist. Es arbeitet unabhängig vom Ubuntu System und eine Instanz über dem, was sich hinter „apt-get“ verbirgt. Somit regelt es beispielsweise bei der Installation neuer ROS-Pakete die internen Abhängigkeiten und die korrekte Integration der Pakete.

Die Initialisierung erfolgt durch den Shell-Befehl „sudo rosdep init“ und wird nur einmalig nach Installtion von ROS durchgeführt. Danach wird mit „rosdep update“ das rosdep-System in dem aktuellen Benutzer-Account durch hinzufügen einiger Dateien im „home“-Verzeichnis eingerichtet. Es ist ebenfalls nötig Umgebungsvariablen zu initialisieren, damit ROS alle benötigten Dateien innerhalb des Dateisystems lokalisieren kann. Hierzu wird der Befehl „source /opt/ros/kinetic/setup.bash“ verwendet, wodurch das setup.bash-Skript ausgeführt.

Eigene Projekte und ROS-Pakete, die nicht mittels „apt-get install“ automatisch heruntergeladen und installiert werden können, werden in einem separaten „workspace“gespeichert. Der workspace besteht aus einem eigenen Verzeichnis, dass man mit dem Linux Shell-Befehl „mkdir“ (für „make directory“) erstellen kann. Dieses Verzeichnis kann an einer beliebigen Stelle auf dem System liegen. Innerhalb des erstellten Verzeichnisses muss ein Unterordner mit der Bezeichnung „src“ angelegt werden. Dieser Ordner enthält später den Quellcode der angelegten Pakete. Das catkin-build-System von ROS kompiliert nach eingabe des Befehls „catkin_make“ in dem Verzeichnis alle sich darin befindlichen Pakete und Quellcode.(vgl. O’Kane, 2013, S. 13 ff.)

Der für diese Arbeit erstellte workspace befindet sich im Linux Home-Verzeichnis und wurde für eine einfache Identifikation „catkin_ws“ genannt.

6.2. Installation der Hardware-Treiber, OctoMap und Hector-SLAM

Der erste Schritt um die Hardware-Treiber für den Velodyne-Lidar und Ladybug 5+ zu installieren erfolgt durch den Download der Quellcode enthaltenden ROS-Pakete. Verfügbar sind die Pakete auf GitHub. Für Gerätetreiber bietet GitHub einen eigenen Bereich mit einer Liste der auf der Seite enthaltenen Pakete und Downloads. ((vgl. github.com/ros drivers))

Die Pakete werden als Zip-Dateien heruntergeladen und in den src-Ordner des catkin_ws-Verzeichnisses entpackt. Um ausführbare Programme aus dem Quellcode zu generieren wird als nächstes das catkin-build-System verwendet. Hierzu wird ein neues Terminal-Fenster geöffnet und in das Verzeichnis des workspace gewechselt, wo der Befehl „catkin_make“ eingegeben wird. Dieser lässt das build-System den enthaltenen Quellcode der Pakete kompilieren und erstellt einige Unterverzeichnisse. (vgl. O'Kane, 2013, S. 45)

Bevor die nun neu erstellten Programme der Treiber-Pakete von einem Terminal aus gestartet werden können, fehlt noch ein letzter Schritt. Es muss nach jedem build-Vorgang ein Skript mit dem Namen „setup.bash“ ausgeführt werden. Dieses Skript wird vom build-System automatisch erstellt und legt Umgebungsvariablen an, damit ROS die Pakete und die darin enthaltenen Nodes und Launch-Dateien finden kann. Der Befehl, mit dem das Skript ausgeführt wird lautet „source devel/setup.bash“. Um zu überprüfen, ob die Installation der Pakete funktioniert, können einzelne Nodes der Pakete gestartet werden. Dies geschieht wieder über die Eingabe in ein Terminal und durch die Befehle „rosrun“ für Nodes und dementsprechend „roslaunch“ für Launch-Dateien. Auf die Launch-Dateien wird in dem Kapitel 6.4 näher eingegangen. Soll eine Node ausgeführt werden, benötigt ROS zusätzlich zum Namen der Node noch den Paketnamen.

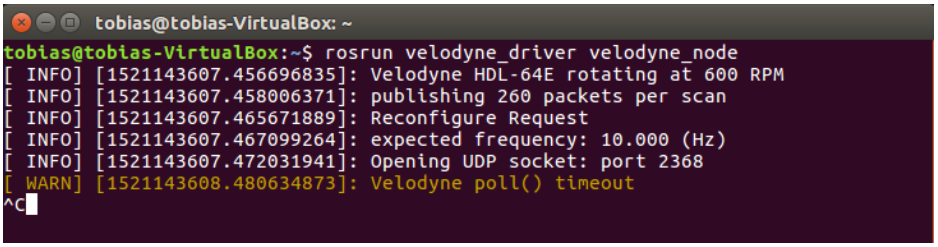

Abbildung 6.1.: Antwort der Velodyne-Node nach dem Aufrufen

In der obigen Abbildung sind die im ROS-Logging-System mit „INFO“ deklarierten Nachrichten der Velodyne-Node zu sehen. Viele Nodes stellen dem Benutzer auf diese Weise Informationen über den Zustand und wichtige Parameter des Programms zur Verfügung.

Die Installation von Hector-SLAM wird auch über einen Download von GitHub und anschließendem Kompilieren im workspace realisiert. (vgl. github.com/tu-darmstadt-ros pkg/)

OctoMap wird als stand-alone Bibliothek und folglich ROS unabhängig installiert. Hierzu wird der Befehl „sudo apt-get install ros-kinetic-octomap“ verwendet. Die Installation beinhaltet eine Konfiguration für ROS, so dass OctoMap als ROS-Paket verwendet werden kann (vgl. wiki.ros.org/octomap). OctoMap wird in dem Verzeichnis „ /opt/ros/kinetic/share/“ gespeichert.

6.3. Herstellen der SLAM-Funktionalität

Sind alle für die Realisierung benötigten ROS-Pakete installiert und durch einzelnes Starten auf ihre Funktion hin überprüft, kann mit der Umsetzung der SLAM-Funkion begonnen werden.

Wie im Konzept (Kapitel 5.4) erläutert, besteht die Realisierung der SLAM-Funktion und Kartenerzeugung aus zwei Teilen. Erstens die Verbindung der benötigten Nodes mit den richtigen Topics und zweitens das Sicherstellen einer korrekten Transformation der Koordinatensysteme.

6.3.1. Verknüpfung der Nodes/Nodelets über Topics

In der Dokumentation der einzelnen Nodes und Nodelets ist angegeben, in welche Topics sie ihre Nachrichten senden und aus welchen sie Daten beziehen. Werden sie über ein Terminal gestartet, versuchen sie automatisch diese Topics zu finden und von ihnen Nachrichten zu beziehen.

Wird der Nodelet-Manager „velodyne_nodelet_manager„ und die Nodelets mit den Endungen „_laserscan“, „_driver“ und „_cloud“ gestartet, erfüllt jede Nodelet ihre Aufgabe sobald Daten eines angeschlossenen Velodyne-Lidars zur Verfügung stehen. Eine gespeicherte Aufnahme von Velodyne-Daten kann mit den Treibern ebenfalls abgespielt werden und simuliert dann einen angeschlossenen Sensor.

Nodelets unterscheiden sich in der Art der Kommunikation von Nodes. Nodelets untereinander verwenden für die Informationsweitergabe „shared pointer“ also geteilte Zeiger auf Addressen im Datenspeicher. Mit anderen Nodes können sie nur durch einen Nodelet-Manager kommunizieren, da dieser den gleichen Standard wie Nodes verwendet. Der Vorteil den Nodelets dadurch haben liegt in der schnellen und kopierfreien („zero copy") Datenweitergabe, was bei großen Datenmengen die Systemleistung erhöht. (Vgl. ClearpathRobotics, 2015)

Die Nodelet-Manager-Node erzeugt über die Treiber durch starten der drei benötigten Nodelets die Topics „velodyne_points“ und „/scan“.

Als nächtes werden durch das Starten der „/hector_mapping“-Node die Daten aus dem erzeugten „/scan“-Topic der Velodyne-Treiber bezogen und weiterverarbeitet. Da das erzeugte und benötigte Topic übereinstimmt, also sowohl die Bezeichnung, als auch der Nachrichtentyp, sind keine Modifikationen notwendig und der Datenfluss erfolgt automatisch nach dem Starten.

Um die von den Treibern erzeugten „pointcloud2“-Nachrichten über das „/velodyne_points“-Topic mit OctoMap vearbeiten zu können, muss in der Start-Datei eine Anpassung vorgenommen werden. OctoMap wird über eine Launch-Datei gestartet, in welcher angegeben wird, von welchem Topic die Daten stammen. In dieser Datei muss unter „remap“ eine neu Zuordnung des „cloud_in“-Topics stattfinden, wie in Abb. 6.2 geschehen.

```
<node pkg="octomap_server" type="octomap_server_node" name="octomap_server">
        <param name="resolution" value="0.03" />

        <!-- fixed map frame (set to 'map' if SLAM or localization running!) -->
        <param name="frame_id" type="string" value="map" />

        <!-- maximum range to integrate (speedup!) -->
        <param name="sensor_model/max_range" value="6.0" />

        <!-- data source to integrate (PointCloud2) -->
        <remap from="cloud_in" to="/velodyne_points" />
</node>
```

Abbildung 6.2.: Ausschnitt der OctoMap Launch-Datei

Sind alle Nodes und Nodelets gestartet und wird mit mit dem Rosbag-Paket eine Datei abgespielt, die Velodyne-Rohdaten an das „velodyne_packets“-Topic sendet, zeigt sich nachfolgende Ansicht der Abbildung 6.3. Diese wird mit dem Werkzeug „rqt_graph“ erzeugt und zeigt alle aktiven Nodes/Nodelets und Topics. Die dargestellten Nodes, die mit dem Transformatoions-Topic „/tf“ verbunden sind werden in dem nächsten Kapitel 6.3.2 genauer erklärt, da sie für das Erzeugen der Baumstruktur der Transformationen zuständig sind.

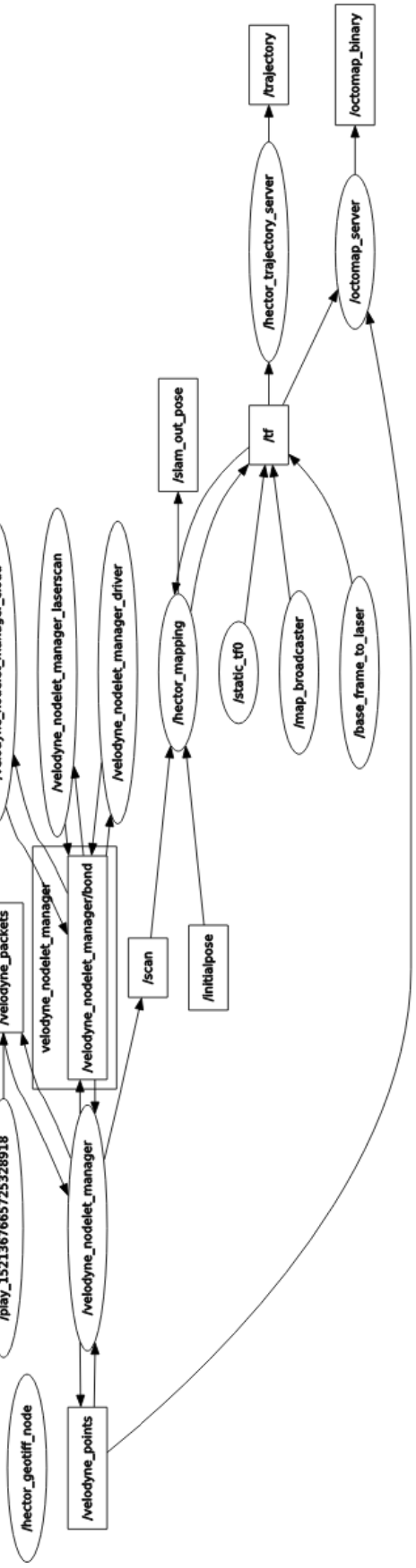

Abbildung 6.3.: Ausgabe der aktiven Nodes/Nodelets und Topics mit rqt_graph

6.3.2. Erzeugen der benötigten Transformations-Struktur

Wie in dem Kapitel 5.4 über das Konzept beschrieben, wird insgesamt eine vollständige Transformation zwischen den Sensordaten im Velodyne-Koordinatensystem und dem Map-Koordinatensystem, in dem die Sensorposition variiert, benötigt.

Hector-SLAM erzeugt nach erfolgreichem Durchlaufen des SLAM-Algorithmus bereits eine Transformation für 3 Freiheitsgrade und sendet diese an das „/tfTopic. Die dorthin gesendeten Transformationen werden von der „/octomap_server„- und der „/hector_trajectory_server“-Node abonniert. Letztere zeichnet alle erhaltenen Transformationen auf und erstellt eine in Rviz anzeigbare Trajektorie über die abgefahrene Route.

Um die zusätzlich benötigten statischen Transformationen zu erzeugen, wird auf das tf-Paket von ROS zurückgegriffen. Mit diesem lassen sich mit Hilfe von Bibliotheksfunktionen Transformationen entweder in C++ oder Python programmieren. Für den vereinfachten Fall der statischen Transformationen bietet das Paket außerdem ein Kommandozeilen-Werkzeug, was in diesem Fall genutzt wird.

Nachfolgende Transformationen zwischen Koordinatensystemen werden benötigt :

- /world $\rightarrow$ /map
- /map $\rightarrow$ /base_footprint
- /base_footprint $\rightarrow$ /velodyne

Das base-footprint-System wird von Hector-SLAM erzeugt und ist das zweidimensionale Ergebnis der Positionsbestimmung. Die Position des map-Systems relativ zum world-System und die relative Position vom base_footprint- und velodyne-System müssen ergänzt werden.

Das Kommandozeilen-Werkzeug „static_transform_publisher“ kann eine statische Transformation sowohl durch direkte Eingabe in ein Terminal als auch durch schreiben in eine Launch-Datei erzeugen. Die äußere Form der Funktion hat dabei folgendes Format

static_transform_publisher x y z rx ry rz frame_id child_frame_id period_in_ms

Bei der Umsetzung ist darauf zu achten, dass die korrekte Höhe der Empfangs- und Sendeeinheit des Sensors gegenüber der Bezugsebene des base_footprint-Systems gewählt wird. Das map-System ist zunächst kongruent zum world-System. Entstehen im Feldversuch Umgebungskarten an verschiedenen Orten, kann die Position der Karten durch entsprechendes anpassen der Transformation verändert werden, so dass die relativen Positionen der Karten zueinander der Realität entsprechen.

```
<node name="world_to_map" pkg="tf"
      type="static_transform_publisher"
      args="0 0 0 0 0 0 /world /map 100" />

<node name="base_frame_to_laser" pkg="tf"
      type="static_transform_publisher"
      args="0.0 0.0 1.65 0.0 0.0 0.0 /base_footprint /velodyne 10" />
```

Abbildung 6.4.: Ausgabe der aktiven Nodes/Nodelets und Topics mit rqt_graph

Abbildung 6.4 zeigt den fertigen Skript-Code, der in eine der verwendeten Launch-Dateien eingefügt werden muss. Die dadurch erzeugte Node für die Transformation von der Basis des Fahrzeugs zum Velodyne-Sensor verschiebt das System des Velodyne um 1,65 m nach oben, an die reale Position des Sensors auf dem Dachträger.

Die Argumente enthalten neben den Werten der translatorischen und rotatorischen Transformation in allen drei Achsen noch das Start- bzw. Eltern-System, das Kind-System und die Periodendauer, mit der die Nachrichten erzeugt werden sollen. Für die sich nicht verändernde Transformation vom word- zum map-System ist eine Frequenz von 10 Hz ausreichend, da selbst nicht aktuelle Nachrichten für die Transformation verwendet werden können. Die Nachrichten der „base_frame_to_laser“ Transformation werden von OctoMap jedoch zusammen mit denen des SLAM-Algorithmus benötigt. Aus diesem Grund sollten die Nachrichten möglichst zum gleichen Zeitpunkt verfügbar sein, da das System sonst auf Nachrichten warten muss und zusätzliche Verzögerungen entstehen, auch wenn die Transformation statisch ist. Um die Verzögerung gering zu halten wird eine Frequenz von 100 Hz eingestellt. (vgl. wiki.ros.org/tf/Troubleshooting, 2009)

Überprüfen kann man die Funktionsweise der Struktur durch ausführen der Node „view_frames“ des tf-Pakets. Diese sammelt Informationen aller Transformationen über einen Zeitraum von fünf Sekunden und speichert eine Übersicht darüber als PDF-Datei. Mit dem Befehl „evince view_frames.pdf“ kann die Datei anschließend geöffnet werden. Die Abbildung 6.5 zeigt diese Übersicht, die die gewünschte Baumstruktur enthält. Es ist zu beachten, dass alle Nodes, die mit den Transformationen in Kontakt stehen, gestartet sein müssen. Für das erzeugen der Übersicht wurden aufgezeichnete Daten in das System eingespeist, damit Hector-SLAM in der Lage ist die dynamische Transformation zu erzeugen.

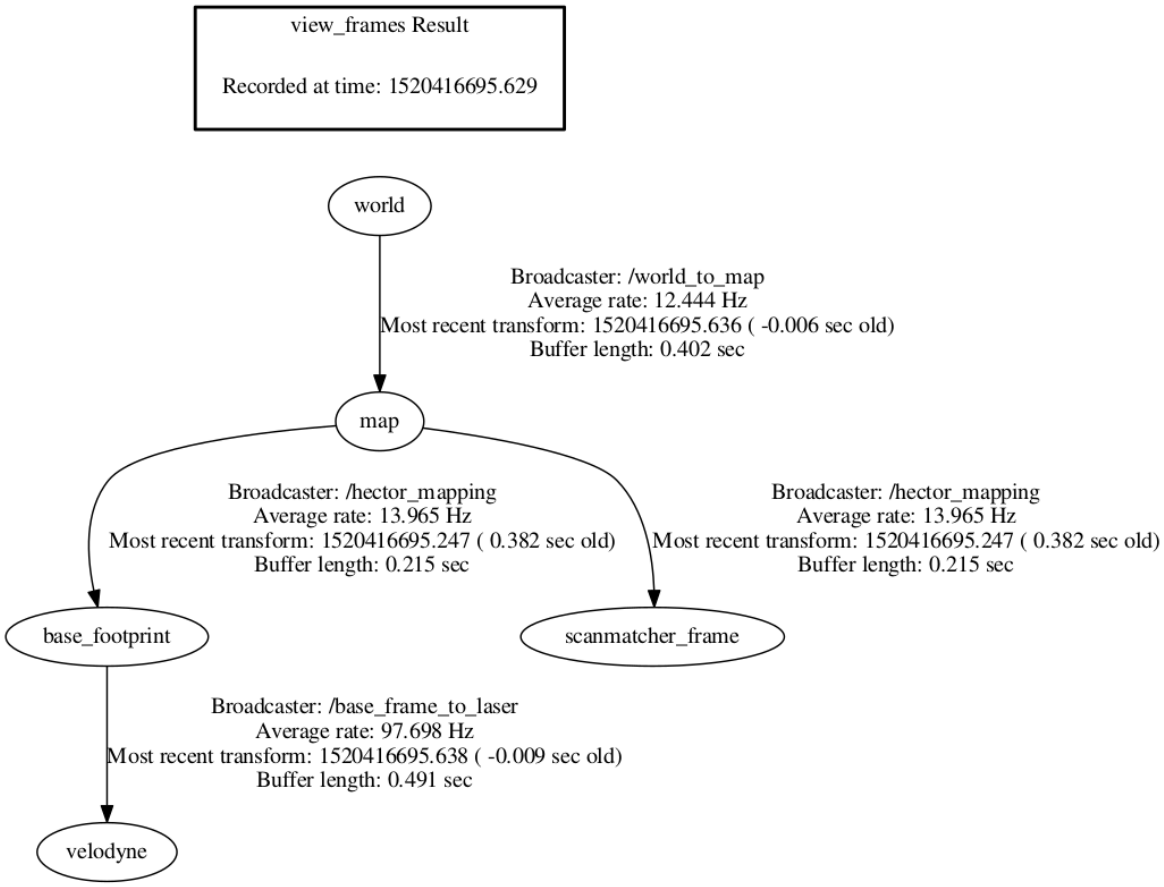

Abbildung 6.5.: Mit dem tf-Paket erzeugte Übersicht der Transformationen

6.4. Erstellen einer Datei für das vorkonfigurierte Starten der SLAM-Lösung

Es kann sehr mühselig sein, alle Nodes und Nodelets einzeln durch Eingabe der entsprechenden Befehle in ein Terminal zu starten. Insbesondere bei komplexeren Programmstrukturen mit vielen Paketen und Funktionen. In ROS gibt es deshalb eine Vereinfachung, mit der man die nötigen Nodes und Nodelets mit nur einer Datei starten kann. Diese Dateien nennen sich Launch-Dateien und haben die Endung ".launch". Sie lassen sich mit einem einfachen Text-Editor öffnen und bearbeiten. Der verwendete Syntax ist ein spezielles XML-Format. (Vgl. O'Kane, 2013, S. 83 ff.)

Die Launch-Dateien befinden sich innerhalb eines ROS-Pakets. Sie werden mit dem Befehl „roslaunch“ ausgeführt. Viele Pakete enthalten bereits solche Dateien, mit denen nicht nur Nodes und Nodelets gestartet, sondern auch konfiguriert werden können.

Da das Projekt noch erweitert werden kann, macht es Sinn zunächst ein neues ROS-Paket zu erstellen. In dieses Paket kann dann sowohl der Quellcode selbst erstellter Nodes als auch eine neu erstellte Launch-Datei. Mit Hilfe dieser werden dann weitere Launch-Dateien verwendeter Pakete aufgerufen, die die entsprechenden Nodes konfigurieren und starten.

Das neue Paket wird mit dem Befehl „catkin_create_pkg“, den man von dem „src“-Verzeichnis des ROS-Workspaces aus ausführt, erstellt. ROS generiert dann automatisch Unterverzeichnisse für Quellcode sowie zwei Dateien im Hauptverzeichnis des Pakets. Eine Datei, die „package.xml“, wird Manifest genannt. In diesem sind Informationen über das Paket enthalten, ebenso wie Abhängigkeiten mit anderen Paketen angegeben. Das Verzeichnis, an dem das Manifest gespeichert ist, ist per Definition gleichzeitig das Hauptverzeichnis des Pakets.(vgl. O'Kane, 2013, S. 17 f.) In diesem Verzeichnis befindet sich zusätzlich die benötigte „CMakeLists.txt“-Datei. Diese dient als Input für das CMake-Programmierwerkzeug. Es kompiliert und fügt den geschriebenen Quellcode mit anderen Quellcodes aus verwendeten Bibliotheken zusammen und erstellt die ausführbaren Dateien. (vgl. wiki.ros.org, 2017a)

Der Inhalt des neu erstellten Pakets ist in Abb. 6.6 zu sehen.

Abbildung 6.6.: Grundlegender Inhalt eines ROS-Pakets mit „package.xml“- und „CMakelists.txt“-Dateien

Nun kann innerhalb des Pakets eine Launch-Datei erstellt werden. Hierzu wird im Hauptverzeichnis ein neuer Ordner mit der Bezeichnung „launch“ erstellt. Als nächstes wird in diesen Ordner navigiert und mit einem Text-Editor eine Launch-Datei erzeugt. Die erstellte Datei trägt den intuitiven Namen „velo_mapping.launch“ und ist nun Teil des neuen „mobile_mapping“-Pakets.

Innerhalb der Datei werden nun drei weitere Launch-Dateien aufgerufen, die sich zu den drei verwendeten Paketen Hector-SLAM, OctoMap und Velodyne zuordnen lassen (vgl. Abb. 6.7.

```
<launch>

<include file="$(find hector_slam_launch)/launch/tutorial.launch"/>
<include file="$(find octomap_server)/launch/octomap_velodyne.launch"/>
<include file="$(find velodyne_pointcloud)/launch/32e_points.launch"/>

</launch>
```

Abbildung 6.7.: Inhalt der Launch-Datei „velo_mapping.launch“

Jede dieser drei Dateien enthält sowohl eine Liste von Nodes oder Nodelets, die mit dem Aufruf gestartet werde, als auch verschiedene Parameter für die Konfiguration oder weitere Launch-Dateien.

Die nachfolgende Abbildung 6.8 zeigt als Beispiel den Inhalt der „tutorial.launch“-Datei von Hector-SLAM.

```
<launch>

  <arg name="geotiff_map_file_path" default="$(find hector_geotiff)/maps"/>

  <param name="/use_sim_time" value="true"/>

  <node name="world_to_map"
        pkg="tf"
        type="static_transform_publisher"
        args="0 0 0 0 0 0 /world /map 100" />

  <node name="base_to_velodyne"
        pkg="tf" type="static_transform_publisher"
        args="0.0 0.0 1.65 0.0 0.0 0.0 /base_footprint /velodyne 10" />

  <node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>

  <include file="$(find hector_mapping)/launch/mapping_default.launch"/>

  <include file="$(find hector_geotiff)/launch/geotiff_mapper.launch">
    <arg name="trajectory_source_frame_name" value="scanmatcher_frame"/>
    <arg name="map_file_path" value="$(arg geotiff_map_file_path)"/>
  </include>

</launch>
```

Abbildung 6.8.: Inhalt der Launch-Datei „tutoroal.launch“ des Hector-SLAM Pakets

In dieser wird der Parameter „use_sim_time“ auf wahr gesetzt, um beim abspielen von gespeicherten Velodyne-Daten einen neuen Zeitstempel zu erzeugen und eine Live-Aufnahme zu simulieren. Es werden die Nodes für die statischen Transformationen gestartet sowie das Visualisierungsprogramm Rviz, dass bei der Installation von ROS automatisch mit installiert wird. Weiterhin werden die Launch-Dateien „mapping_default“ und „geotiff_mapper“ aufgerufen. Jeder Node können verschiedene Argumente übergeben werden, um sie mit den gewünschten Eigenschaften zu starten. Die beiden erwähnten Dateien sind für den Start der eigentlichen SLAM-Funktion (mapping_default) sowie für das Speichern einer zweidimensionalen Karte (geotiff_mapper) zuständig. Die anderen beiden Dateien starten und konfigurieren die Velodyne-Treiber für die Verwendung mit dem HDL-32E sowie das OctoMap Kartenprogramm. Die genauen Inhalte aller verwendeten Launch-Dateien sind im Anhang zu finden.

Wird nun der Befehl „roslaunch mobile_mapping velo_mapping.launch“ ausgeführt, öffnen sich verschiedene Terminal-Fenster, in dem der Status der Nodes sichtbar wird. Zusätzlich öffnet sich ein Benutzerinterface mit einem Fenster für die grafische Darstellung der Sensordaten sowie der berechneten Position der SLAM-Lösung und Anzeige der auf den Daten basierenden dreidimensionalen Karten durch OctoMap. In Abbildung 6.9 ist der geladene

Nutzerbildschirm zu sehen und in Abb. 6.10 eine Detailansicht der Bedienelemente des linken Frames. Um diese Ansicht in Rviz automatisch mit jedem Start zu erhalten, müssen die Einstellungen der Anzeige sowie Topic- und Frame-Auswahl beim ersten Start gespeichert werden. Um direkt Sensordaten angezeigt zu bekommen muss entweder der Velodyne mit dem System verbunden und aktiv sein, oder eine gespeicherte Aufnahme abgespielt werden.

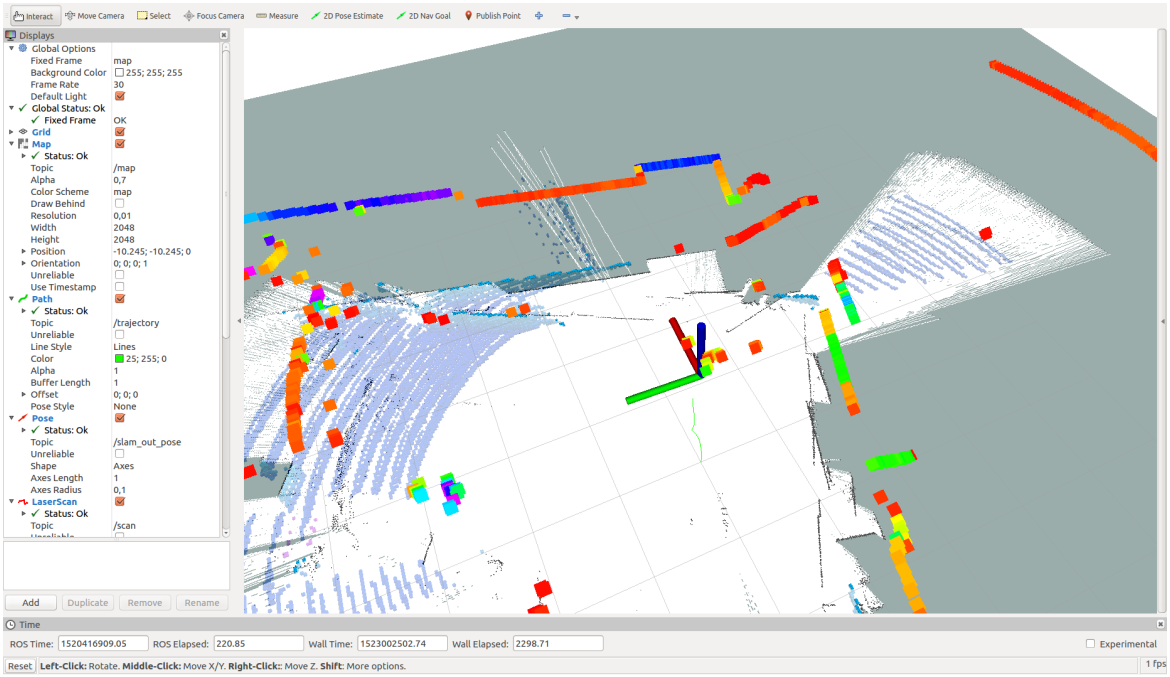

Abbildung 6.9.: Ansicht von Rviz nach dem Start durch die erstellte Launch-Datei

In der Ansicht werden alle Informationen eingeblendet, die Verfügbar sind. Die von Hector-SLAM generierte zweidimensionale Karte, die aktuelle Position des Velodyne sowie die zurückgelegte Strecke als Trajektorie. Die aktuelle Position wird als Koordinatensystem angezeigt und befindet sich in der Mitte des Fensters. Die Trajektorie wird als grüne Linie dargestellt. Auch die Daten des extrahierten Rings sind eingeblendet, mit denen der Scan-Matcher arbeitet. In der Anzeige erscheinen sie als recht große, flächige Quadrate. Die mit OctoMap generierte Karte baut sich über der zweidimensionalen Karte auf. In diesem Beispiel sind nur einige Voxel in der Karte registiert (blaue ringförmige Struktur auf der zweidimensionalen Karte).

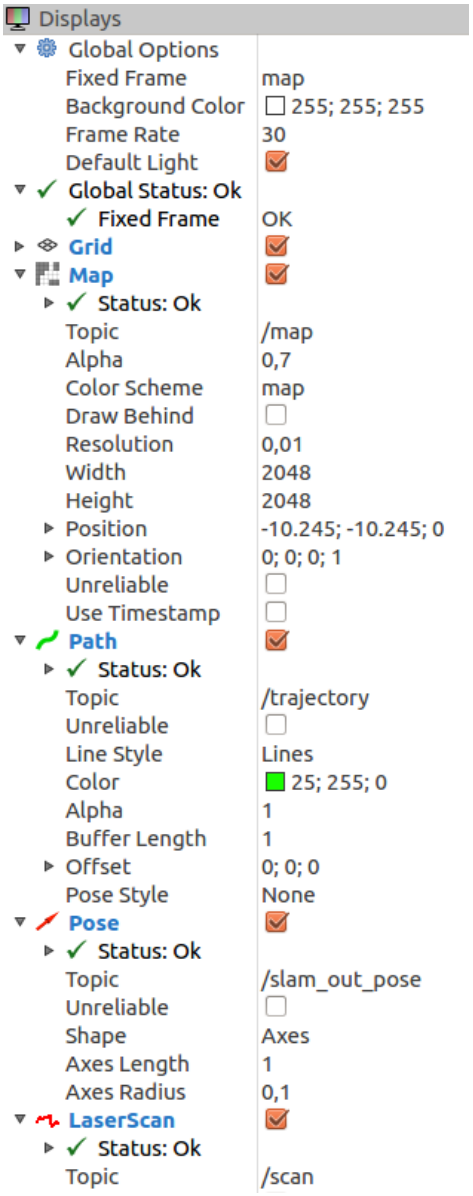

Abbildung 6.10.: Ausschnitt des Display-Frames von Rviz

In dem Display-Frame werden alle Einstellungen für die Anzeige vorgenommen. Es wird das Koordinatensystem der Karte gewählt, sowie alle Topics ausgewählt, die die entsprechenden Daten liefern. Beispielsweise wird die berechnete Position aus dem /slam_out_pose-Topic von Hector-SLAM ausgelesen und dann in der dreidimensionalen Ansicht angezeigt. Einzelne Ansichten lassen sich über einen Haken ein- oder ausblenden.

7. Testen des Systems und der SLAM-Funktion

In diesem Kapitel werden Tests beschrieben und durchgeführt, um die Funktion und Leistungsfähigkeit des Systems zu überprüfen. Ebenfalls findet eine jeweilige Auswertung der Tests statt.

Aufgrund des komplexen Umfangs möglicher Tests für die SLAM-Funktion findet eine Beschränkung auf drei Testfälle statt, die nachfolgend beschrieben werden.

Der erste Test befasst sich mit der Gesamtfunktion des Systems unter Laborbedingungen. Es wird qualitativ getestet, ob die Zusammenarbeit der Komponenten und der Software die gewünschten Funktionen grundsätzlich ermöglichen kann. Weiterhin wird die Genauigkeit der Lösung überprüft.

Als nächstes wird getestet, ob die Funktion auch unter den Bedingungen in einer öffentlichen Straße funktionieren kann. Dieser Test ist qualitativ und soll auch die Grenzen des Systems ausloten.

Der letzte Test sieht die Benutzung in einer Tiefgarage vor. Bei diesem Test handelt es sich um einen realistischen Nutzungsfall für SLAM-Anwendungen ohne GPS-Unterstützung. Auch dieser Test ist qualitativ und beurteilt die Plausibilität der Ergebnisse.

7.1. Test unter Laborbedingungen

Es wird ein Versuchsaufbau beschrieben, mit dem das System mobil im Labor in Betrieb genommen wird. Es werden zwei unterschiedliche Tests durchgeführt, um die Leistung des Systems abzuschätzen. In der Auswertung werden die Ergebnisse reflektiert und die Lösung eines auftretendes Problems beschrieben.

7.1.1. Versuchsaufbau

Der Aufbau stellt den Funktional den Betrieb im Versuchsfahrzeug nach, in dem die Systemkomponenten auf ein fahrbares Gerüst befestigt werden. Abbildung 7.1 zeigt das fertig ausgerüstete Versuchsgefährt. Es besteht aus einem Aluminiumgerüst mit Rollen und einer Siebdruckplatte auf mittlerer Höhe. Der Velodyne befindet sich an einer oberen Ecke und ist mit dem Profil in einer Höhe von 1,92 Meter verschraubt. Die Ladybug Kamera ist der Plattform in der Mitte Gerüsts befestigt. Für die Kommunikation mit dem Systemcomputer wird ein Laptop verwendet, welcher über eine Ethernet Schnittstelle verbunden ist.

Abbildung 7.1.: Der für den Laborversuch genutzte fahrbare Aufbau

7.1.2. Versuchsbeschreibung

Mit diesem Test soll die grundsätzliche Funktionalität überprüft werden. Zu dieser gehört das ordnungsgemäße Zusammenarbeiten der verwendeten Komponenten entsprechend der Planung, also die Stromversorgung der Geräte über den Akku und die Möglichkeit der Datenauswertung mit dem Velodyne-Lidar und Ladybug zeitgleich. Die Stromversorgung wird auf

die Einsatzdauer hin getestet, in dem Messungen der Akkuspannung während des Betriebs gemacht werden.

Weiterhin wird die SLAM-Funktion sowohl qualitativ als auch quantitativ beurteilt. Hierfür wird nach dem Start der SLAM-Funktion der Wagen durch das Labor und durch eine offene Tür in den Flur des Gebäudes bewegt. Werden plausible Positionswerte anhand der Abtastung der Umgebung erzeugt, ist der qualitative Test erfolgreich. Quantitativ wird die erstellte Karte hinsichtlich der Maßhaltigkeit überprüft. Dies geschieht durch vermessen des Labors mit einem Laserentfernungsmessgerät des Typs Bosch PLR 25, der Abstände mit einer angegebenen Genauigkeit von +/- 2mm messen kann (vgl. Bosch). Es wird nur die Genauigkeit von Wandabständen durch Messung überprüft und die erzeugte Karte visuell auf Winkelabweichungen und Fehler jeglicher Art kontrolliert.

Weiterhin wird durch eine definierte Start- und Zielposition der Messfahrt die Wiederholgenauigkeit der vom SLAM-Algorithmus ermittelten Position betrachtet. Zu diesem Zweck wird die aufgezeichnete Trajektorie an Start und Ziel verglichen. Im Idealfall sind Start- und Endpunkt der Trajektorie kongruent.

Der Velodyne wird für den Test mit einer Rotationsfrequenz von 15Hz betrieben.

7.1.3. Versuchsdurchführung

Der Versuchsaufbau wird auf einer Bodenmarkierung platziert und der Ausgang des Akkus eingeschaltet. Danach wird über das Bedienteil der Computer eingeschaltet und sich über den Laptop und einem VNC-Programm mit dem virtuellen Desktop verbunden.

Als nächstes werden die nachfolgenden Befehle nacheinander in die angegebenen Terminalfenster des Linux-Betriebssystems eingegeben:

- Terminal 1: roscore
- Terminal 2: cd catkin_ws
- Terminal 2: source devel/setup.bash
- Terminal 2: roslaunch mobile_mapping velo_mapping.launch
- Terminal 3: rosrun camera1394 camera1394_driver

Mit dem ersten Befehl wird ROS gestartet. Mit den Befehlen in dem zweiten Terminal wird in den workspace navigiert und die Umgebungsvariablen initialisiert, um danach mit „roslaunch“ die Launch-Datei zu starten.

Nach dem alle Nodes/Nodelets gestartet sind und sich Rviz geöffnet hat, wird zusätzlich zu den angezeigten Sensordaten noch die Anzeige der empfangenen Kamerabilder hinzugefügt.

Als nächstes wird mit einer langsamen Schrittgeschwindigkeit der Aufbau vom Labor aus durch die Tür in den Flur der Etage und wieder zurück bewegt. Am Ende wird versucht, den Wagen exakt so auf der Markierung zu platzieren, dass Endposition der Startposition entspricht.

Ist die Position wieder erreicht, wird die erzeugte zweidimensionale Karte inklusive Trajektorie und die dreidimensionale Karte von OctoMap gespeichert. Bis die Karten gespeichert sind, darf das Versuchsfahrzeug seine Endposition nicht verlassen.

Das Speichern erfolgt mit den Befehleen:

- Terminal 4: rosrun octomap_server octomap_saver mapfile.bt
- Terminal 5: rostopic pub syscommand std_msgs/String “savegeotiff“

Um die Mindestanforderungen an die Akkulaufzeit zu testen, wird das System anschließend mit aktiven Sensoren und der SLAM-Funktion bis zu einer Zeitdauer von ca. 2,5 Stunden stehengelassen. Die Akkuspannung wird nach einer Stunden und nach 2,5 Stunden gemessen.

7.1.4. Ergebnis

Nach dem Starten aller Nodes und Nodelets für den Betrieb des Systems mit der SLAM-Funktion zeigt sich nachfolgender Bildschirm in Abbildung 7.2.

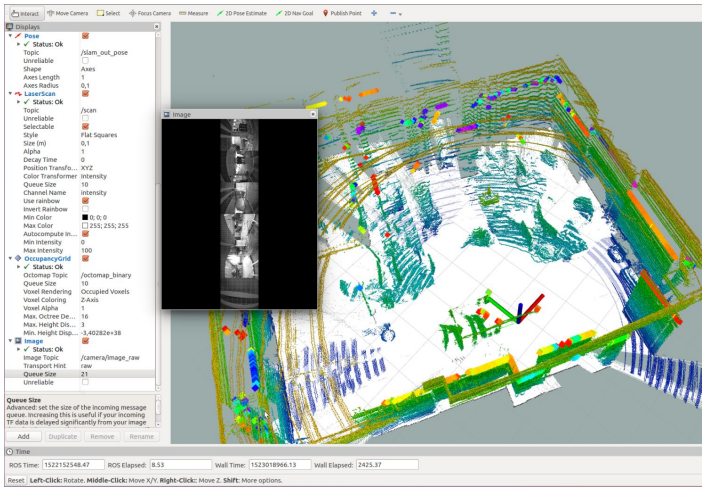

Abbildung 7.2.: Bildschirmansicht nach der Inbetriebnahme aller Funktionen

Zu sehen ist das Rviz Fenster mit den verarbeiteten Daten des Velodyne-Lidars im Hintergrund. In dem kleinen Fenster im Vordergrund ist das Kamerabild zu sehen. Es ist in den Grundeinstellungen schwarz/weiß und wird nicht flüssig wiedergegeben. Die Bildwiederholrate wird auf ca. 5-10 Bilder pro Sekunde geschätzt.

Die als TIFF-Bilddatei gespeicherte Karte der Messfahrt von Hector-SLAM ist ausschnittsweise in Abbildung 7.3 dargestellt.

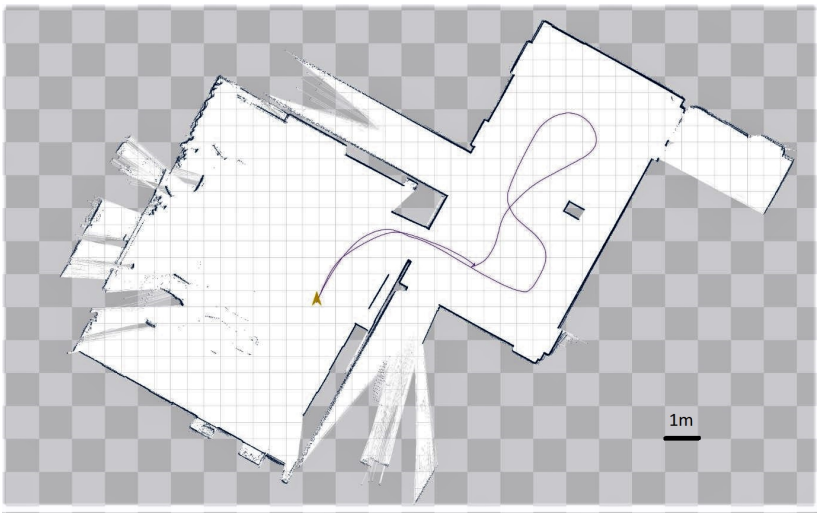

Abbildung 7.3.: Auschnitt der vom Paket Hector-SLAM gespeicherten Karte

Der Maßstab ist in der Bilddatei mit einem Meter pro Quadrat des Schachbrettmusters angegeben. Auf dem Bild deutlich zu erkennen sind die Umrisse der Räumlichkeiten sowie die berechnete Trajektorie. Am Zielpunkt der Messfahrt ist ein kleiner gelber Pfeil sichtbar.

Ein Ausschnitt der dreidimensionalen Karte ist in der folgenden Abbildung 7.4 zu sehen.

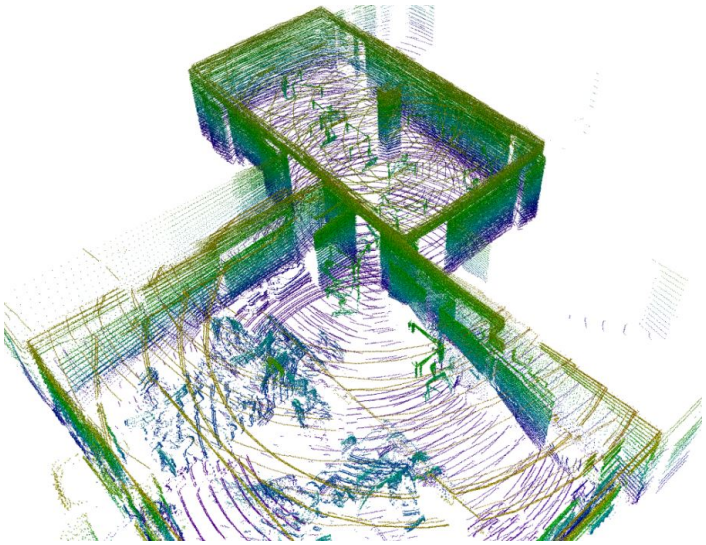

Abbildung 7.4.: Auschnitt der von OctoMap erzeugten dreidimensionalen Karte

Sie zeigt den Raum mit der Start- und Zielposition sowie den Flur der Ebene mit einer Säule in der Mitte. Die Farben der einzelnen Voxel spiegeln die Höhe in der Karte wieder.

7.1.5. Auswertung

Das erste Ziel, eine Softwareauswertung der Sensoren mobil während der Fahrt vollziehen zu können ist, wie in Abb. 7.2 zu sehen, erreicht. Die komplette Software kann über einen Laptop oder ein ähnliches Gerät gestartet und Bedient werden. Es ist möglich, mindestens die vorhanden Sensoren gleichzeitig auszuwerten und ihre Daten aufzunehmen. Nach einer Stunde Betrieb ist die gemessene Akkuspannung von 13,37V auf 12,96V gesunken und nach 2,5 Stunden auf 12,3 Volt. Somit ist die Mindesteinsatzdauer von einer Stunde weit übertroffen und bietet sogar noch Reserven, bis der Tiefentladeschutz des Akkus bei ca. 11 Volt den Ausgang abschaltet.

Die Genauigkeit der SLAM-Lösung wird anhand der Bemaßung der zweidimensionalen Karte von Hector-SLAM beurteilt, da Messungen des dreidimensionalen Raums der OctoMap-Karte schwer zu realisieren sind und die erzeugte Karte eine partiell geringe Dichte von Messwerten aufweist.

Die nachfolgenden Abbildung 7.5 hat die mit dem Bosch PLR 25 nachgemessenen Strecken zwischen einigen Wänden eingetragen. Diese Distanzen werden in der Tabelle 7.6 mit den Distanzen aus Hector-SLAM verglichen. Die Entfernungen in der Bilddatei werden über den angegebenen Maßstab und die Pixel-Anzahl bestimmt. 300 Pixel des Bildes ergeben somit einen Meter.

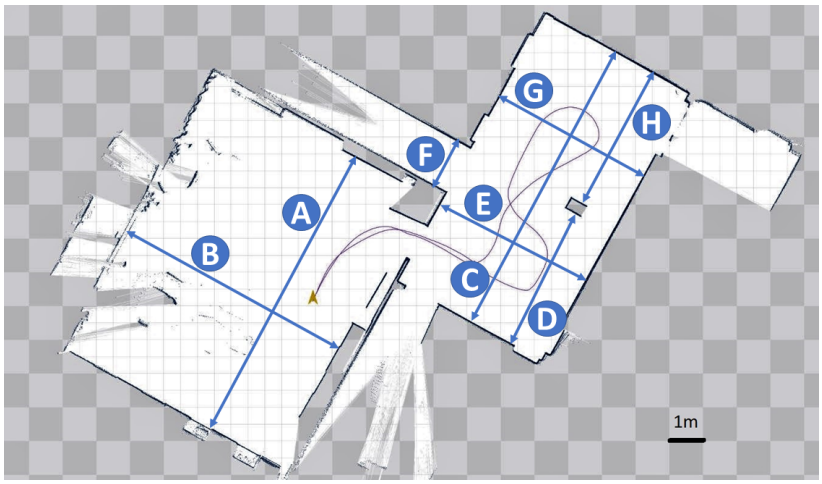

Abbildung 7.5.: Position der Messpunkte für den Vergleich

Messstrecke	SLAM [m]	Bosch PLR 25 [m]	Differenz [m]
A	9,10	9,045	0,055
B	7,24	7,245	0,005
C	9,05	9,003	0,047
D	4,29	4,261	0,029
E	4,85	4,827	0,023
F	1,44	1,441	0,001
G	4,86	4,83	0,030
H	4,37	4,363	0,070
		Mittelwert:	0,033

Abbildung 7.6.: Tabelle mit dem Vergleich der Messwerte über die Differenz

Die Messungen der Einzelstrecken ergeben eine Durchschnittliche Abweichung der beiden Messmethoden von 0,033 Metern. Die niedrigste Differenz beträgt 5mm und die größte 70mm. Somit befinden sich die Werte außerhalb des gesteckten Ziels von 2 cm. Für die verwendete Methodik der Entfernungsmessung ist das Ergebnis jedoch gut, bedenkt man, dass der Velodyne HDL-32E die tolerierte Ungenauigkeit bereits voll ausnutzt. Weitere Toleranzen entstehen durch den Bosch Entfernungsmesser (+/- 2 mm) sowie durch Abweichungen von

der idealen Messstrecke durch Winkelfehler. Ebenso ist die Auflösung der Strukturen in der Karte begrenzt (1 cm pro Zelle) und teilweise eine ungleichmäßige Oberfläche durch Rauschen oder fehl gedeuteten Messungen, wie in Abb. 7.7 zu sehen ist. Diese ist der Ausschnitt einer starken Vergrößerung der Karte.

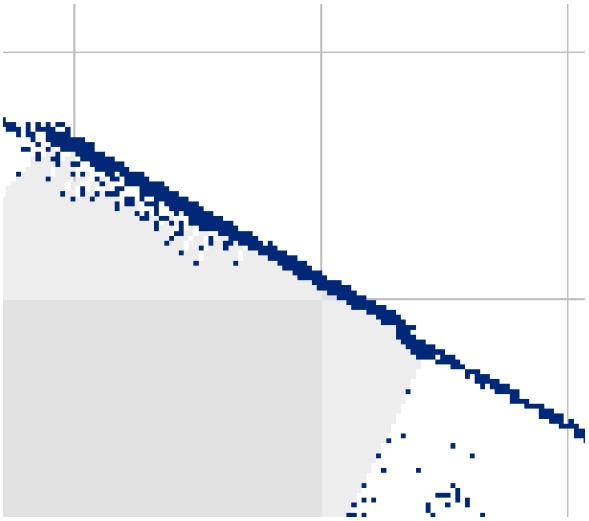

Abbildung 7.7.: Starke Vergrößerung der Karte mit einzelnen sichtbaren Pixeln, die jeweils einen Zentimeter entsprechen

Die gezeichnete Trajektorie entspricht augenscheinlich der wirklich gefahrenen Trajektorie. Start- und Zielposition scheinen kongruent zu sein, so dass Hector-SLAM beim wieder einnehmen der gleichen Position in der Realität auch die gleiche Position in der Karte berechnet. Leider Zeichnet das Programm einen gelben Pfeil an die letzte bestimmte Position, so dass die Abweichung zwischen Start- und Ziel nur erahnt werden kann, wie in Abbildung 7.8 zu sehen ist.

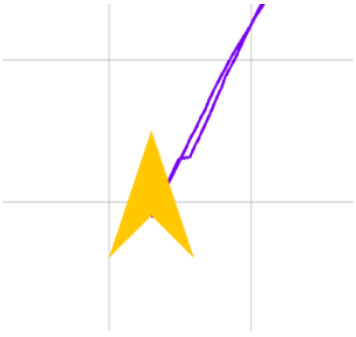

Abbildung 7.8.: Vergrößerte Darstellung der Start- und Zielposition der Trajektorie

Die Maßhaltigkeit der erzeugten Karte bezieht sich nicht nur auf die Abstände zwischen Objekten oder Mauern bzw.Wänden, sondern auch auf entstehende Winkelfehler der verbundenen räumlichen Strukturen zueinander. Da das Gebäude in dem die Messfahrt durchgeführt wurde nicht ausreichend genau vermessen werden kann, wird davon ausgegangen, dass innerhalb eines Raumes und zwischen den Räumen kein Winkelversatz existiert. Somit können perfekt geometrische Strukturen in die Karte eingeblendet werden und so die Abweichungen von der perfekten Form aufzeigen. In Abbildung 7.9 werden die einzelnen Räume verglichen, und in Abbildung 7.10 der komplette Verbund, also wie die Räume zueinander stehen.

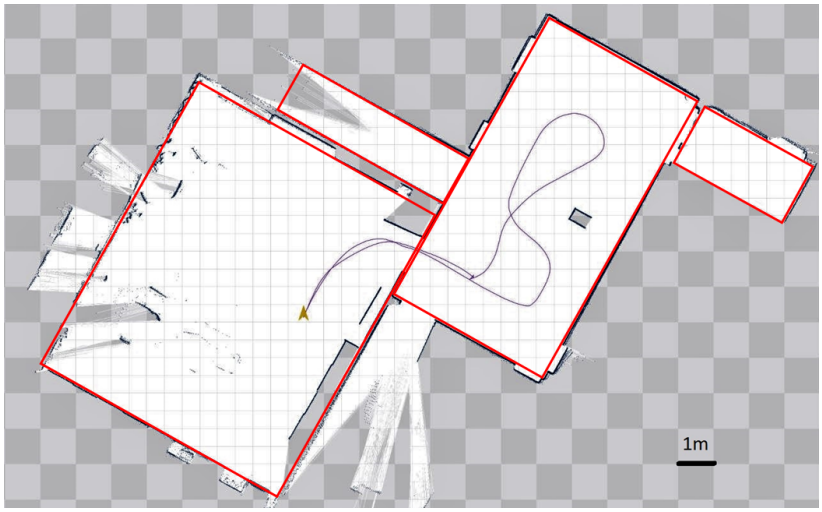

Abbildung 7.9.: Vergleich der einzelnen Räume mit der idealen Form

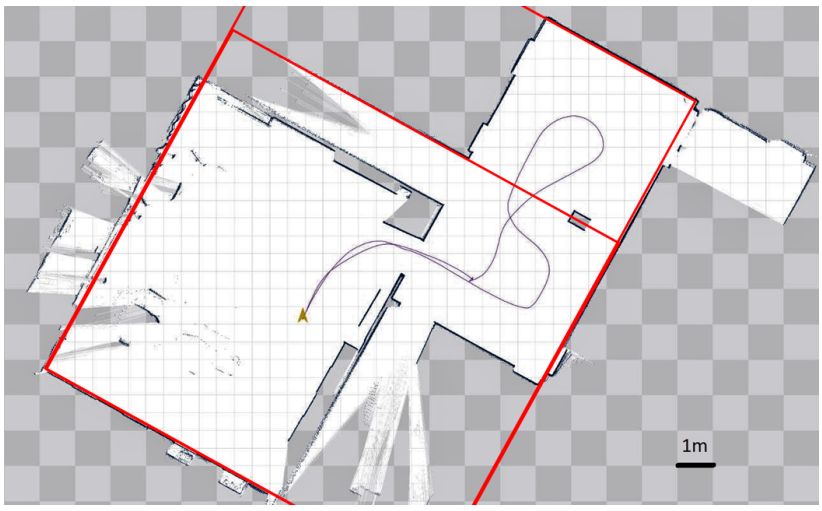

Abbildung 7.10.: Vergleich der Winkelabweichung der Räume zueinander

Sowohl die einzelnen Räume, als auch die Räume zueinander, scheinen relativ dich an der idealen Form zu liegen, die in den Abbildungen rot markiert ist. Lediglich der große Raum weicht an der Außenseite etwas ab, wie in Abb 7.11 zu sehen ist.

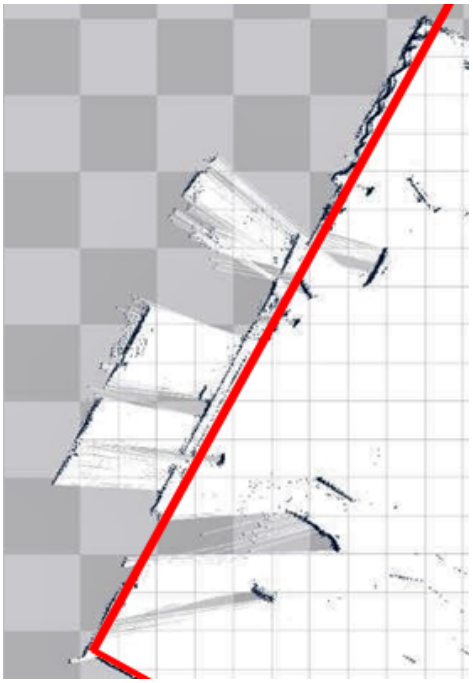

Abbildung 7.11.: Ausschnittsvergrößerung der linken Außenwand des großen Raums

Eine Erklärung für die Abweichung könnte die offensichtlich geringere Anzahl an Messungen sein, die auf dieser Seite gemacht wurden. Wandstrukturen in der Nähe der Trajektorie werden definierter dargestellt, da mit geringerer Entfernung die Abtastpunkte des Lidars enger beieinander liegen. In der Abbildung ist ebenfalls zu sehen, dass teilweise durch Fenster hindurch Messungen nach draußen stattfanden. Diese scheinen recht weit gestreut und könnten den Scanmatcher-Algorithmus dazu veranlasst haben, das Minimum der Abweichungen an leicht falscher Stelle anzunehmen.

7.2. Test in offenem Gelände

Das System wird in der Nähe von Gebäuden auf dem Campusgelände in Betrieb genommen. Es wird zwischen den Gebäuden gemessen und anschließend eine Straße passiert. Mit diesem Test soll die Leistungsfähigkeit und die Grenzen der SLAM-Lösung unter anspruchsvollen Umgebungsbedingungen getestet werden.

7.2.1. Versuchsaufbau

Der Versuchsaufbau entspricht dem Aufbau vom ersten Test unter Laborbedingungen.

7.2.2. Versuchsbeschreibung

Der Versuch zielt darauf ab herauszufinden, inwiefern das System unter den Umgebungsbedingen eines Einsatzortes für das autonome Fahren funktionieren kann. Hierzu wird der Versuchsaufbau zwischen Gebäuden und auf einer Straße mit vielen parkenden Autos bewegt. Es wird nur die SLAM-Funktion mittels des Velodyne-Lidars betrachtet.

7.2.3. Versuchsdurchführung

Entsprechend dem ersten Test wir das System gestartet. Auf die Datenerfassung mit der Ladybug-Kamera wird bei diesem Test jedoch verzichtet. Als Startpunkt ist ein Ort in direkter Nähe zu einem Gebäude gewählt, damit sichergestellt ist, dass eine erste Position durch die Gebäudewände auf jeden Fall gefunden wird. Ist dies geschehen, wird der Versuchsaufbau vom Gebäude weg bewegt entlang eines Gehweges zwischen zwei Gebäuden in Richtung einer Straße. Um ein realistisches Szenario zu erhalten wird das Messfahrzeug ca. 30 Meter so auf der Straße bewegt, wie auch ein Auto die Straße entlang fahren würde. Die Geschwindigkeit während der Messung beträgt Schrittgeschwindigkeit (ca. 4-7 Kmh).

7.2.4. Ergebnis

Das automatische Erstellen der Karte musste nach einer Strecke von ca. 50 Metern abgebrochen werden, da offensichtliche Fehler in der Lokalisierung auftraten. Sie machten sich durch sprunghafte Veränderungen in der aktuell angezeigte Position bemerkbar. In der Abbildung 7.12 ist die erstellte Karte sowie die letzte Position bevor das SLAM beendet wurde.

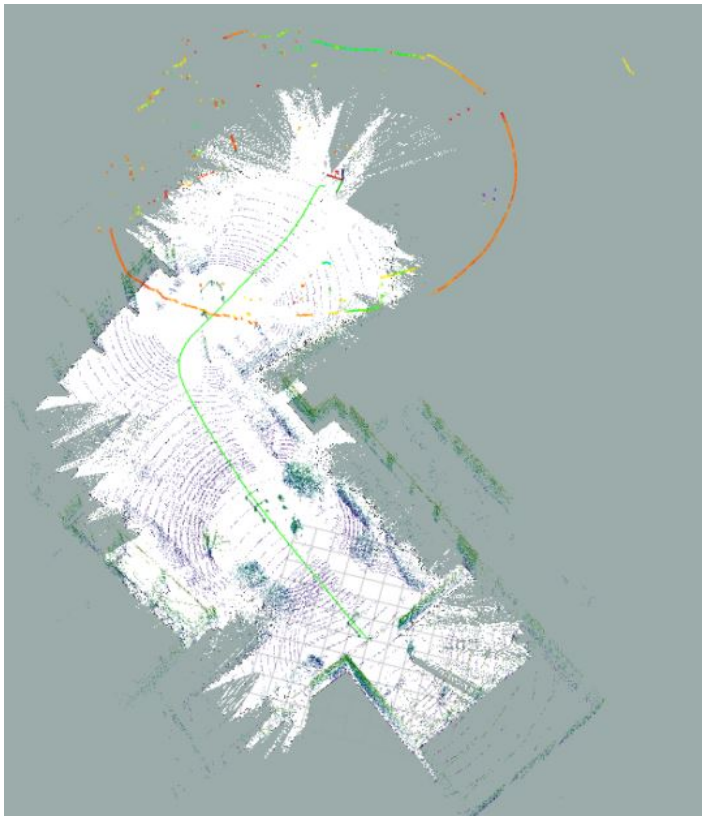

Abbildung 7.12.: Ausschnitt der letzten Ansicht in Rviz vor Beenden der SLAM-Funktion

Der zweidimensionalen Karte von Hector-SLAM ist die zu dem Zeitpunkt nicht vollständige dreidimensionale Karte aus OctoMap überlagert. Die hellgrüne Linie stellt die Trajektorie dar. Der farbige Kreis in der oberen Hälfte der Abbildung zeigt die Entfernungsmessungen eines einzelnen Ringes des Velodyne-Lidar, die für den SLAM-Algorithmus genutzt werden.

Der letzte Abschnitt der Strecke vor dem Verlust der Lokalisierung wurde vier weitere Male anhand der aufgezeichneten Daten neu ausgewertet, um zu sehen, ob der Fehler reproduzierbar ist. Das Lokalisierungsproblem ist jedes Mal an sehr ähnlicher Position aufgetreten.

In Abbildung 7.13 ist ein Ausschnitt mit einem Lokalisierungsfehler zu sehen. Er zeichnet sich durch sprunghaftes verändern der angenommenen Position innerhalb von Sekundenbruchteilen aus. Durch die schnellen Sprünge und weit auseinander liegenden Fehlpositionen besteht die Trajektorie nur noch aus geraden Linien.

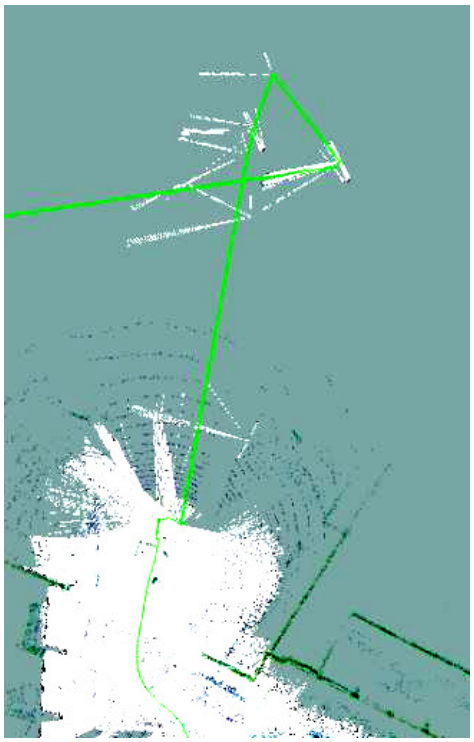

Abbildung 7.13.: Ausschnitt der Ansicht in Rviz bei Positionsfehlern

7.2.5. Auswertung

In direkter Gebäudenähe hat die SLAM-Funktion gut funktioniert, das System zeichnet eine Trajektorie, die subjektiv der zurückgelegten Strecke entspricht. Die dazu erzeugte Karte scheint ebenfalls die Positionen von Häusern und Gelände der Wirklichkeit zu entsprechen.

Der Fehler, der auftritt wenn man sich von den Gebäuden entfernt, lässt sich in zwei grundlegende Fehlerquellen einteilen. Die erste ist, dass die Entfernungsinformationen vom Scan-Matcher in Hector-SLAM falsch interpretiert werden. Die zweite ist, dass nicht genügend Informationen der Umgebung in den Daten enthalten sind, um damit eine eindeutige Position bestimmen zu können. Die letztere Variante ist in der Funktionsweise des Scan-Matchers begründet. Als Beispiel für einen zu geringen Informationsgehalt kann die Fahrt durch einen

Tunnel herangezogen werden. Verdeutlicht wird die Problematik in Abbildung 7.14. Durch zu wenige Informationen von Objekten oder Strukturen an der Seite bzw. vor oder hinter dem Sensor, hat sich aus sicht des Scan-Matchers nichts verändert. Die Lösung der SLAM-Funktion ist für diesen Fall, dass sich die Position nur minimal nach links verschoben hat und der Sensor zwischen den Tunnelwänden stehen bleibt, obgleich sich das Fahrzeug in der Realität fortbewegt.

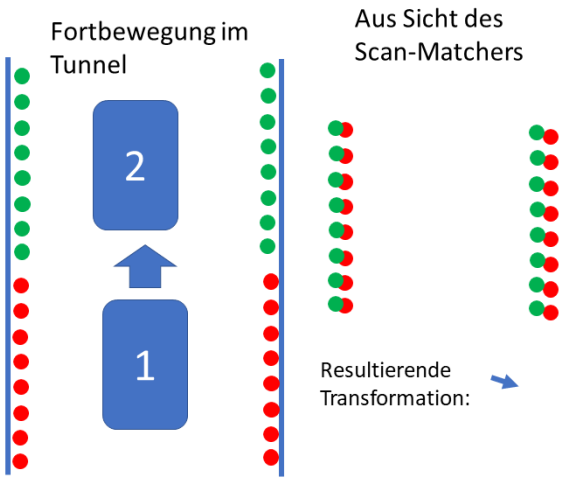

Abbildung 7.14.: Positionsfehler durch zu wenig Umgebungsinformationen

Bei der anderen Fehlervariante sind in den Scans zwar genügend Informationen vorhanden, aufgrund der Algorithmik des Scan-Matchers werden diese jedoch falsch interpretiert. Der Algorithmus in Hector-SLAM versucht, die Verschiebung zwischen den aktuellen Sensordaten und der bisherigen gespeicherten Karte zu finden, in dem eine Transformation mit der größten Übereinstimmung gesucht wird. Bedingt durch diese Eigenschaft kann es vorkommen, dass nicht nutzbare Sensordaten besser übereinstimmen, als die Sensordaten mit richtigem Informationsgehalt.

Abbildung 7.15 zeigt einen Ausschnitt der Ansicht in Rviz kurz nach dem Start des Versuchs. Eindeutig zu sehen ist die von einer einzelnen Ebene des Velodyne-Lidars erfasste Gebäudestruktur. Die Daten des Scans sind farbig in die Karte eingezeichnet. Der Farbton entspricht dem gemessenen Reflexionsgrad. In Gebäudenähe hat die SLAM-Methode gut Funktioniert, da hier viele Referenzstrukturen erfasst werden, die der echten Position von Umgebungsdetails entsprechen.

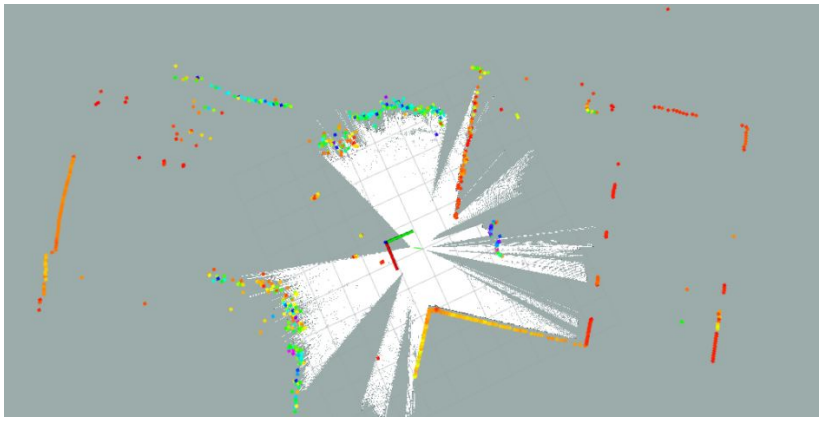

Abbildung 7.15.: Scan mit nutzbaren Sensordaten

In Abbildung 7.16 ist dies nicht der Fall. In der Momentaufnahme ist ein Scan zu sehen, bei dem die meisten Scan-Endpunkte eine Ringstruktur um den Sensor bilden. Dieser Ring entspricht aber keiner tatsächlich vorhanden Struktur mit einer festen Position in der Umgebung, sondern dem Boden. Bewegt sich der Sensor in weitläufigem Gelände, wird jedes mal durch den Boden diese Ringstruktur erzeugt. Sie macht den größeren Teil der Messpunkte aus, wodurch der Scan-Matcher eine optimale Lösung in der Überlagerung der Ringe sieht und dadurch Fehler in der Lokalisierung erzeugt.

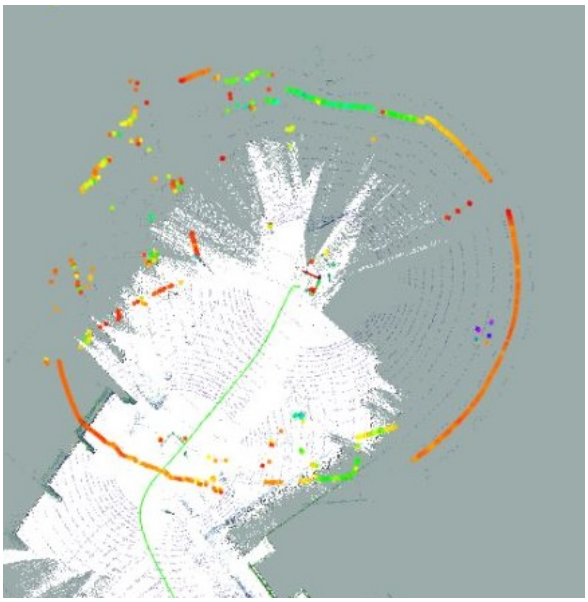

Abbildung 7.16.: Scan mit nicht nutzbaren Sensordaten

Der Grund, weshalb der Boden auf den Aufnahmen sichtbar wird liegt in dem Aufbau des Velodyne-Lidars sowie in der Weiterverarbeitung der Sensordaten. Die einzelnen Zeilen der dreidimensionalen Scans entstehen durch Laserentfernungssensoren, die in bestimmten Winkeln zueinander angeordnet sind, um einen vertikalen Öffnungswinkeln von -30,67° bis +10,67° gegenüber einer horizontalen Achse aufzuspannen (vgl. Kapitel 2.2.2). Wird aus den Sensordaten mittels der velodyne_laserscan Nodelet ein Ring extrahiert, muss darauf geachtet werden, welchen Winkel der dazugehörige Sensor hat. Ist der Sensor in Richtung des Bodens ausgerichtet, kommt es schon bei geringen Distanzen zu der gezeigten Ringstruktur in den Scans.

In der C++ Datei „VelodyneLaserScan.cpp“ kann entsprechend der Abbildung 7.17 bestimmt werden, welcher Ring aus den Daten extrahiert wird. Die Datei befindet sich innerhalb des velodyne_master Pakets im ROS workspace. Ring Nummer 23 liegt dicht an der horizontalen Ebene (vgl. VelodyneLidar, 2013). Vorher war Ring 16 gewählt, wodurch der abtastende Laserstrahl in Richtung Boden zeigt.

```
// Select ring to use
uint16_t ring;
if ((cfg_.ring < 0) || (cfg_.ring >= ring_count_)) {
  // Default to ring closest to being level for each known sensor
  if (ring_count_ > 32) {
    ring = 57; // HDL-64E
  } else if (ring_count_ > 16) {
    ring = 23; // HDL-32E default = 23
  } else {
    ring = 8; // VLP-16
  }
} else {
  ring = cfg_.ring;
}
```

Abbildung 7.17.: Ausschnitt der C++ Datei der velodyne_laserscan Nodelet

Die Veränderung des verwendeten Ringes hat zu einer scheinbar leichten Verbesserung der Positionserkennung geführt, wie in Abb. 7.18 zu erkennen ist. Es treten dennoch Fehler auf, die vor allem zu einer falsch angenommenen Winkeländerung führen. Man kann deutlich sehen, dass die Reichweite der Entfernungsmessungen gestiegen ist, verglichen mit dem vorigen Fall, in dem der Boden erfasst wurde.

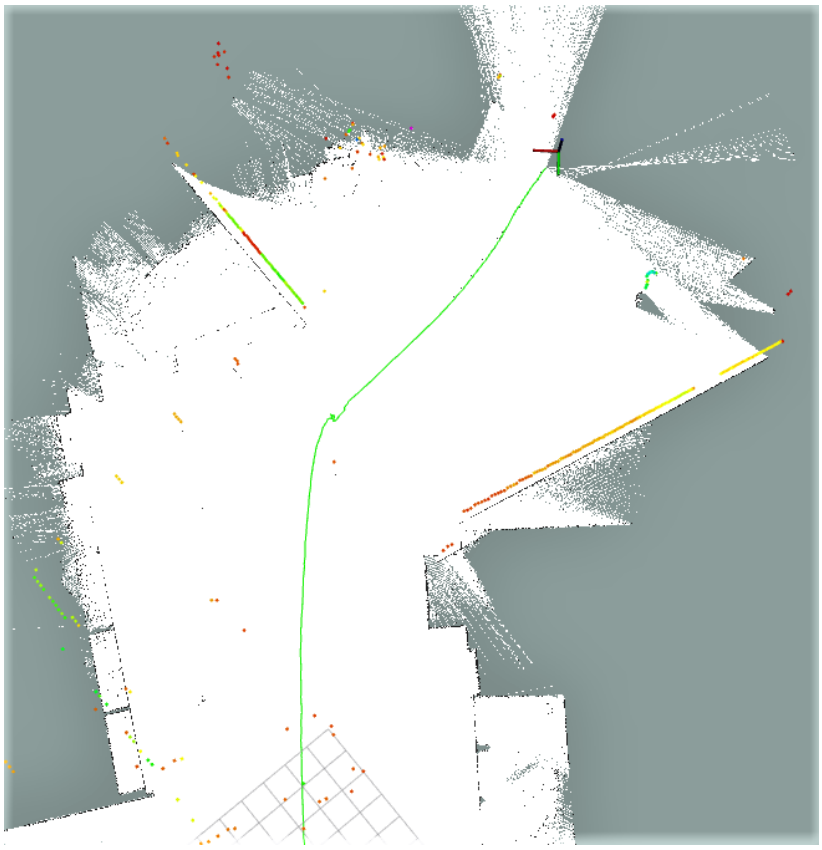

Abbildung 7.18.: Kartenerzeugung mit horizontaler Scan-Linie

7.3. Test in einer Tiefgarage

Als Nutzungsfall für die SLAM-Funktion ist der Einsatz in einer Tiefgarage prädestiniert, da hier keine GPS-Signale empfangen werden können, um die Lokalisierung zu unterstützen. Mit diesem Test wird unter realen Bedingungen untersucht, ob das System plausible Ergebnisse bezüglich der Lokalisation und somit dem erstellen einer Karte liefert.

7.3.1. Versuchsaufbau

Der Versuchsaufbau entspricht dem Aufbau des Versuchsfahrzeugs in Kapitel 3.1 mit der realisierten Hardware im Kofferraum.

7.3.2. Versuchsbeschreibung

Das Sensorsystem wird vom Fahrzeuginneren bedient und Sensordaten aufgezeichnet und gleichzeitig verarbeitet. Der Versuch zielt darauf ab, ein natürliches Fahrverhalten innerhalb der Garage nachzubilden. Dies wird durch ein Wendemanöver, eine Geradeausfahrt und eine Einparksituation am Ende getan. Die Geschwindigkeit entspricht dabei einer Schrittgeschwindigkeit von ca. 4-7 Km/h. Die SLAM-Lösung soll mit den Daten eine zwei- und dreidimensionale Karte erzeugen, die wieder herangezogen wird, um die Genauigkeit und Plausibilität zu überprüfen.

7.3.3. Versuchsdurchführung

Gestartet wird an einem Ende der Garage vor einem verschlossenem Tor. Die Aufzeichnung des Velodyne-Lidars wird, wie in dem Test unter Laborbedingungen beschrieben, gestartet. Als erstes wird Rückwärts aus der Ausfahrt manövriert und gewendet, um dann vorwärts eine Strecke geradeaus fahren zu können. Am Ende der Garage ist eine weitere Auffahrt und freie Parkplätze, die zum Einparken genutzt werden. Anschließend wird die Aufzeichnung beendet.

7.3.4. Ergebnis

Die direkte Auswertung der Sensordaten während der Fahrt hat schon nach wenigen Metern zu großen Fehlern geführt. Deshalb wurden die aufgezeichneten Sensordaten herangezogen, um sie unter Berücksichtigung der Erkenntnisse des vorhergehenden Versuchs nachträglich auszuwerten. Die erzeugte Karte kurz vor dem Ende der Garage ist in der Abbildung 7.19 zu sehen.

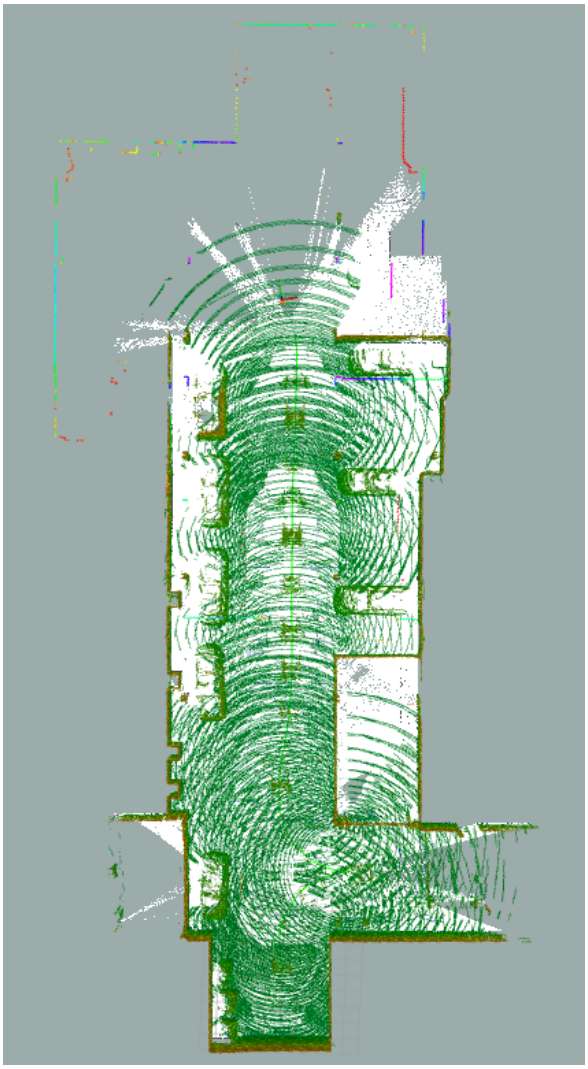

Abbildung 7.19.: Erzeugte dreidimensionale OctoMap-Karte mit der Ansicht von oben auf die Teststrecke

Nach dem kartiertem Streckenabschnitt in der Abbildung folgt ein etwas weitläufigerer Abschnitt mit einer Auffahrt auf die nächste Ebene. In diesem wird eine kurze Linkskurve gefah-

ren, um dann rückwärts einzuparken. Ab dem Zeitpunkt, in dem nach links eingeschwenkt wird, verliert die SLAM-Lösung die Orientierung und es kommt zu Positionsabweichungen, in dem hauptsächlich ein falscher Winkel angenommen wird. Der Fehler ist mit kleinen Abweichungen reproduzierbar, wenn die aufgezeichneten Sensordaten wiederholt ausgewertet werden. Abbildung 7.20 zeigt einen Ausschnitt, auf dem der entstehende Fehler zu sehen ist.

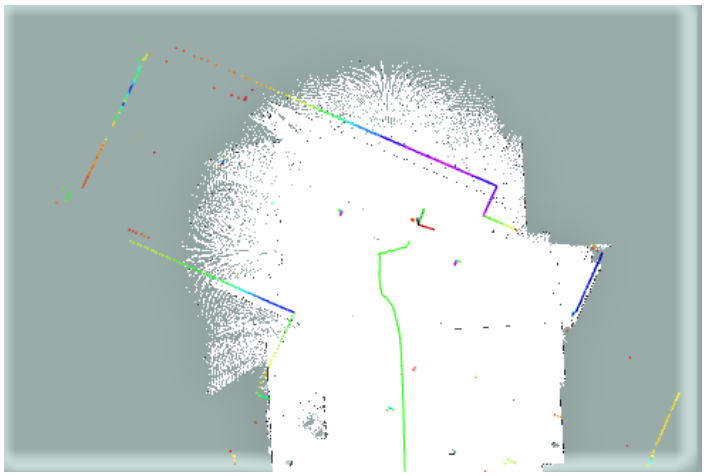

Abbildung 7.20.: Teil der Garage, an dem die SLAM-Lösung Fehler erzeugt

Die farbigen Umrisse sind wieder das Ergebnis der Abtastung durch eine Zeile vom Velodyne-Lidar. Zu sehen ist, dass eine falsche Position des Sensors angenommen wird, und die Sensordaten des Scans somit falsch in die Karte transformiert werden. Die Gebäudestruktur an dieser Stelle entspricht den Umrissen des Scans im oberen Teil der Abbildung 7.19. Durch den Fehler wird vom System jedoch angenommen, dass die Struktur in einem Winkel von ca. 45 Grad zur bisherigen Struktur gedreht ist.

7.3.5. Auswertung

Abgesehen von dem auftretendem Fehler im letzten Teil der Strecke scheint das System eine Trajektorie des gefahrenen Weges zu zeichnen, die Subjektiv dem tatsächlichem Fahrverlauf entspricht. Überprüfen lässt sich dies wieder anhand der maßhaltigkeit der erzeugten zweidimensionalen Karte, die in Abb 7.21 zu sehen ist. Die Karte enthält ebenfalls die zur Überprüfung mit dem Bosch PLR 25 gemessenen Strecken.

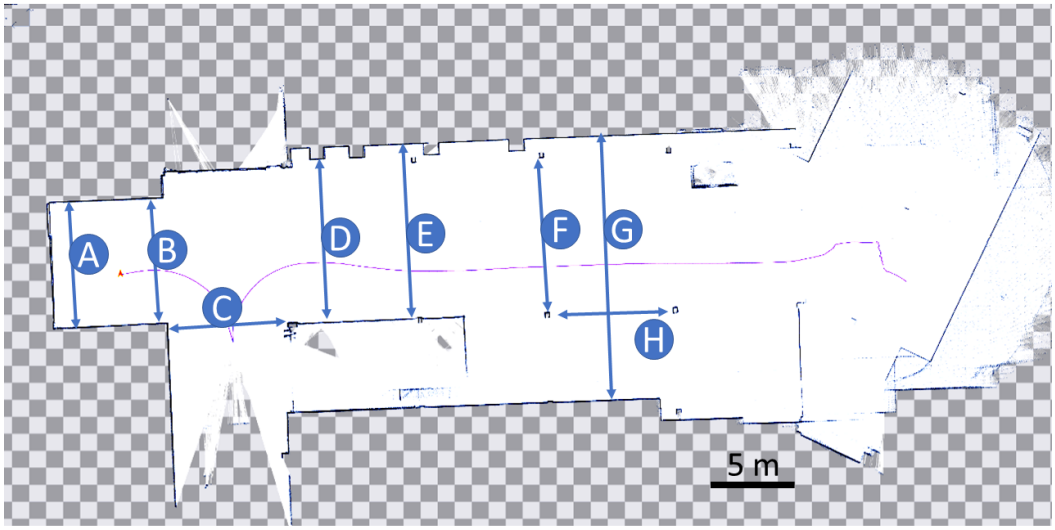

Abbildung 7.21.: Ansicht der Karte mit markierten Referenzmessstrecken

Messstrecke	SLAM [m]	Bosch PLR 25 [m]	Differenz [m]
A	7,39	7,336	0,050
B	7,37	7,345	0,025
C	7,01	7,025	0,015
D	9,64	9,591	0,049
E	10,29	10,194	0,096
F	9,06	9,087	0,027
G	15,53	15,251	0,279
H	7,23	7,210	0,020
		Mittelwert:	0,070

Abbildung 7.22.: Tabelle mit dem Vergleich der Messwerte

In Abbildung 7.22 sieht man eine Tabelle mit dem Vergleich der Abstandsmessungen mittels dem Bosch Entfernungsmesser und durch vermessen der erstellten Karte. Der Maßstab der Karte liegt bei 300 Pixeln pro Meter. Es zeigt sich, dass die Abweichungen im Mittel mehr als doppelt so groß sind, wie der Test unter Laborbedingungen gezeigt hat. Insbesondere bei den längeren Distanzen im Bereich von zehn Metern und mehr betragen die Abweichungen bis zu 0,270 Meter. Erklären lässt sich die Zunahme des Fehlers bei großen Distanzen unter anderem durch einen wachsenden Fehler beim Messen mit dem PLR 25. Durch die größeren Unebenheiten der Wände in der Garage gestaltet sich ein exaktes peilen als schwierig. Auch

gibt es in der Garage verschiedene Strukturen, so dass die Höhe in der gemessen wird eine Rolle spielen kann.

Eine andere Möglichkeit ist, dass die SLAM-Lösung bei der Fahrt mit Schritttempo und größeren Distanzen einen größeren Fehler verursacht, als bei sehr langsamer Fahrt mit geringen Distanzen.

Die Karte lässt sich weiterhin auf Richtigkeit überprüfen, in dem Sie wieder mit künstlichen idealen Strukturen verglichen wird. In der nachfolgenden Abbildung 7.23 sind ideale Wandstrukturen in Form von roten Linien eingeblendet. Die Abweichung ist über die gezeigte Strecke gering und beläuft sich auf wenige Pixel. Eine genauere Aussage lässt sich mit der verwendeten Methode jedoch nicht machen.

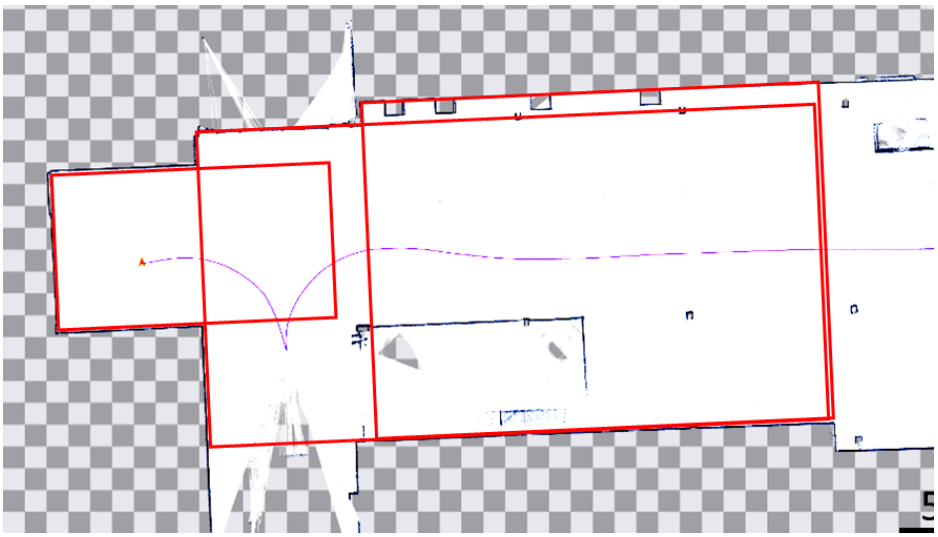

Abbildung 7.23.: Abweichung von der idealen Form

Insgesamt betrachtet zeit der Versuch, dass Hector-SLAM nur in direkter Nähe zu deutlich abgegrenzter Umgebung, wie z.B. Mauerwerk, in der Lage ist eine plausible Karte der Umgebung abzubilden. Ist zu wenig Referenzstruktur mit nutzbarem Informationsgehalt des Umfeldes in den Sensordaten, treten schnell große Fehler auf. Insbesondere Drehungen um die Hochachse des Sensors, also Lenkbewegungen des Fahrzeugs, führen in einer solchen Umgebung zu Fehlern. Aus dem Versuch geht auch hervor, dass Hector-SLAM nicht in der Lage ist, eine einmal verloren Position wiederzufinden und die bis dahin falsch gezeichnete Karte zu korrigieren. Dieses Verhalten hängt mit der Funktionsweise von Hector-SLAM zusammen, der als Scan-Matcher über eine Wahrscheinlichkeitsverteilung funktioniert. Ein Fehler in der Positionsbestimmung führt unverzüglich zum Zeichnen eines neuen, falschen Kartenteils. Dieser falsche Kartenteil wird dann wieder für den Matching-Prozess verwendet, wodurch keine Möglichkeit einer nachträglichen Korrektur besteht.

8. Fazit

Die realisierte Hardware für die Inline-Auswertung von Sensordaten hat sich in der Praxis bewährt. Der ausgewählte Industrie-Computer auf Basis eines Intel Core i7 Prozessors hat sich in den durchgeführten Versuchen als Leistungsfähig genug erwiesen, um die Sensordaten der Ladybug 5+ Kamera und des Velodyne HDL-35E Lidars sowohl aufnehmen zu können, als auch gleichzeitig eine Lokalisierung und Kartenerstellung mit den 3D-Scans des Velodyne auszuführen. Die Leistungsaufnahme des Computers liegt während der Datenerfassung und aktiver SLAM-Funktion bei ca. 150 Watt, gemessenen an einem Graupner Polaron SMPS 1500 Netzteil (vgl. Graupner). Die Einsatzdauer des Gesamtsystems mit dem Akku beträgt mehr als 2,5 Stunden.

Die Hardware, bestehend aus dem Indrustrie-Computer, einem 40Ah 12V Akku und Stromverteiler, ist auf einer Siebdruckplatte aus Holz montiert. Die Rüstzeit, in der das System in den Kofferraum des Versuchsfahrzeugs integriert und mit den Sensoren auf dem Dachträger verkabelt werden kann, hat sich als gering herausgestellt und ist von zwei Personen in ca. 10 Minuten durchführbar. Das Handling der 900x1070 mm messenden Plattform ist dank der montierten Griffe gut.

Als Softwareplattform für die Aufnahmeprogramme und verschiedenen Algorithmen wurde das Robot Operating System (ROS) auf einer Linux-Distribution ausgewählt. ROS hat sich als sehr flexible Variante für das Zusammenfügen verschiedenster Programm-Pakete herausgestellt, was vor allem durch den internen Kommunikationsstandard der Fall ist. Als eine leichte Schwäche ist die relativ lange Einarbeitungszeit zu betrachten, da ROS hauptsächlich über Terminal-Befehle kontrolliert wird.

Für die Lokalisierung und Kartografie mit dem Lidar wurden die ROS-Pakete Hector-SLAM und OctoMap ausgewählt und zusammen mit den Hardware-Treibern verknüpft. Die Funktion, die mit den Paketen realisiert wurde ist eine Lösung des SLAM-Problems. Dieses besteht darin, während der Fortbewegung eine Umgebungskarte zu erstellen, ohne beim Start des Systems die eigene Position zu kennen.

Der Lösungsansatz von Hector-SLAM basiert auf einem zweidimensionalem Scan-Matcher, der eine Ebene des Lidars für den Matching-Prozess nutzt. OctoMap nutzt die berechnete Position von Hector-SLAM für das Erstellen einer dreidimensionalen Karte. In den Versuchen hat sich herausgestellt, dass die Zuverlässigkeit und Genauigkeit innerhalb von Gebäuden

am größten ist (mittlere Abweichung 3,3 cm). Insbesondere Objekt- und Strukturarme Umgebungen sind für die SLAM-Lösung allerdings weniger geeignet. Dies hat sich bei einem Versuch außerhalb, aber in der Nähe von Gebäuden gezeigt. Auch in einem Versuch, bei dem eine weitläufigere Tiefgarage mit dem Versuchsfahrzeug befahren wurde, sind teilweise große Positionsabweichungen entstanden. Die Fehler traten vor allem dann auf, wenn ein Wechsel der Umgebung stattfand und weniger Objekte und klar abgegrenzte Struktur in Form von Mauerwerk in der Nähe des Sensors waren, was mit der Funktionsweise des Scan-Matchers zusammenhängt.

Bezogen auf das autonome Fahren ist die angewandte Lösung nicht zuverlässig und genau genug, um den Ansprüchen an eine zentimetergenaue Lokalisierung und Kartenerstellung gerecht zu werden. Zwar wurden Streckenteile mit teilweise guter Genauigkeit kartografiert, aber vor allem die Zuverlässigkeit über einen kompletten Streckenverlauf einer Teststrecke unter realen Bedingungen war in dem Versuch nicht gegeben. Die Versuche haben gezeigt, dass das System nicht in der Lage ist, nach einmal verlorener Lokalisation, die Orientierung durch neue Sensordaten wiederzufinden. In einem autonomen Einsatz führt diese Eigenschaft zu gefährlichen Situationen, da selbst ein kurzer Verlust der Lokalisation zu großen Abweichungen und Kartenfehlern führt. Dies kann zusammen mit einer Routenplanung zu unvorhersehbarem Fahrverhalten des autonomen Fahrzeugs führen.

9. Ausblick

Das realisierte System mit ROS als Softwareplattform und der vorgestellten SLAM-Lösung besitzt viel Potential für Verbesserungen und Erweiterungen, um es für das autonome Fahren besser Nutzbar zu machen. Mögliche Ansätze bestehen darin, die Funktionalität von Hector-SLAM zu erweitern, oder durch Fusion mit weiteren Sensoren genauer und zuverlässiger zu gestalten. Ebenso denkbar ist die Wahl eines anderen SLAM-Ansatzes, der eine entsprechend erweiterte Funktionen bietet.

Eine einfache Möglichkeit, den Scan-Machting-Prozess zu unterstützen besteht darin, die Sensorwerte zu filtern, bevor diese dem Algorithmus zugeführt werden. Je klarer definiert die Scan-Endpunkte Umgebungskonturen wiedergeben, desto besser findet der Algorithmus eine Transformation zwischen der Karte und dem aktuellen Abbild des Umfeldes. Vegetation zum Beispiel hat abgesehen von Baumstämmen häufig keine klare Abgrenzung. Die Laserstrahlen treffen auf verschiedene Blätter und Äste, wodurch als Abbild eine Punktwolke mit stochastischer Verteilung innerhalb des Strauches o.Ä. entsteht. Der Scan-Matcher sucht nach der maximalen Übereinstimmung aller Punkte, die durch die stochastische Verteilung bei jeder Umdrehung des Lidars nicht mehr bei der tatsächlichen Position liegt. Mittels einer Filter-Node könnten in ROS die Daten der velodyne_laserscan-Nodelet vorverarbeitet werden, bevor man sie Hector-SLAM zugeführt. Der Filter könnte als ein unabhängiges Modul realisiert werden, dass zwischen der Nodelet und Hector-SLAM platziert wird, und alle Scan-Endpunkte herausfiltert, die in einem bestimmten Radius keine weiteren Punkte oder Punktegruppen haben.

Die Realisierung einer Fusion von Sensordaten mit verschiedenen Sensoren ist ebenfalls eine Lösung, die bereits in heutigen Fahrzeugen zum Einsatz kommt. Dafür in Frage kommen sowohl weitere bildgebende Sensoren, wie Kameras, aber auch Inertial Measurement Units (IMUs).

Das Hector-SLAM-Paket ist bereits dafür vorgesehen, eine IMU in den Prozess zu integrieren (vgl. Kohlbrecher u. a., 2011, S. 1) . Vor allem interessant für das Verbessern der Genauigkeit ist die Drehrate um die Hochachse des Sensors und somit des Fahrzeugs. Da schnelle Drehungen in Kombination mit relativ wenig klarer Struktur der Umgebung führen schnell zu Winkelfehlern. Wenn man die Drehrate Integriert und mit einer Gewichtungsfunktion behaftet, um die Sensordrift gering zu halten, erhält man für kurze Zeiträume eine relativ genaue Angabe der Winkeländerung. Die Winkelangabe kann dann dem Scan-Matching-Prozess

zugeführt werden, und dient diesem als Startposition des Prozesses, wodurch zuverlässiger eine richtige Lösung gefunden werden könnte.

In Kapitel 7.2.5 wird ein Problem geschildert, dass bei der Lokalisation in einem Tunnel auftritt. Um dieses Problem zu beheben und auch die Lokalisierung in der Nähe von langen Strukturläusen Gebäudeteilen zu verbessern, könnte man die Datenauswertung des Lidars mit einer Auswertung von Bilddaten des Ladybug 5+ fusionieren. Hierzu könnte ausgenutzt werden, dass der Velodyne nicht nur Informationen über die Entfernung und Position von Hindernissen und Objekten sammelt, sondern auch die Intensität des reflektierten Strahls misst. Somit könnte man Markierungen anbringen oder bereits vorhandenen Straßenmarkierungen nutzen, um die relative Position der stark reflektierende Fläche der Markierung mit dem Velodyne zu identifizieren. Mittels dieser Position und Bildverarbeitung kann durch die Kameraaufnahmen die Markierung optisch vermessen werden und über Winkelbeziehungen die Lokalisation verbessern. Für diesen Ansatz wird allerdings zusätzlich eine andere SLAM-Lösung benötigt, da das speichern und assoziieren von Markierungen mit eigenschaftsbasierten Karten gelöst wird. Abbildung 9.1 stellt den möglichst gleichen Bildausschnitt mit dem Velodyne-Lidar und der Ladybug-Kamera dar und die Verknüpfung über das Straßenschild auf den Aufnahmen. Die höhere Auflösung der Kamera ist bei der Mustererkennung und Peilung von Markierungen dem Lidar überlegen.

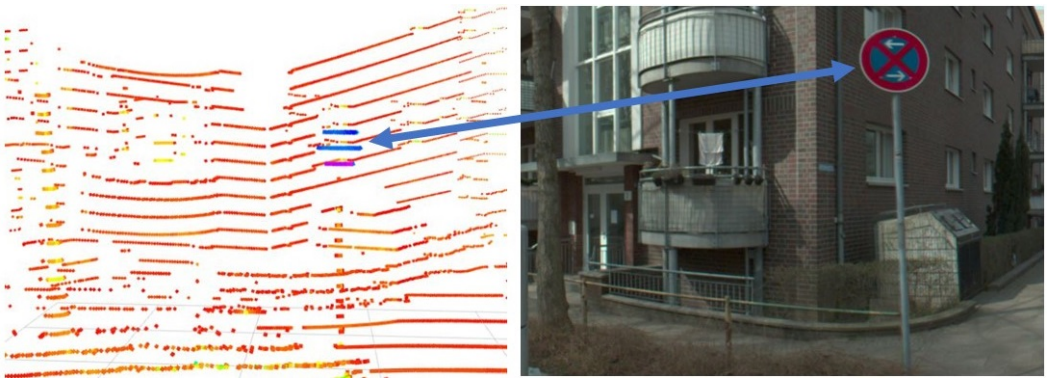

Abbildung 9.1.: Velodyne- und Ladybug-Aufnahme des gleichen Bildausschnitts

Literaturverzeichnis

[Audi 2017a] AUDI: Der neue Audi A8 â hochautomatisiertes Fahren auf Level 3. (2017). – URL https://www.audi-mediacenter.com/de/per-autopilot-richtung-zukunft-die-audi-vision-vom-autonomen-fahren-9der-neue-audi-a8-hochautomatisiertes-fahren-auf-level-3-9307. – Abruf: 2018-02-19

[Audi 2017b] AUDI: Pilotiertes Fahren. (2017). – URL https://www.audi.com/corporate/de/innovationen/mobilitaet-und-technologie/piloted-driving.html. – Abruf: 2018-03-22

[Bergen 2018] BERGEN, Mark: Nobody Wants to Let Google Win The War of Maps All Over Again. (2018). – URL https://www.bloomberg.com/news/features/2018-02-21/nobody-wants-to-let-google-win-the-war-for-maps-all-over-again. – Abruf: 2018-03-24

[Bosch] BOSCH: Elektrowerkzeuge fÃ¼r Heimwerker -Digitaler Laser-Entfernungsmesser PLR 25. . – URL https://www.bosch-do-it.de/de/de/bosch-elektrowerkzeuge/werkzeuge/plr-25-3165140847308-199864.jsp. – Abruf: 2018-04-06

[CARMEN] CARMEN: Introduction - Whats Carmen? . – URL http://carmen.sourceforge.net/intro.html. – Abruf: 2018-02-18

[ClearpathRobotics 2015] CLEARPATHROBOTICS: Nodelet Everything. (2015). – URL http://www.clearpathrobotics.com/assets/guides/ros/Nodelet%20Everything.html. – Abruf: 2018-03-18

[docs.ros.org] DOCS.ROS.ORG: sensor$_m$sgs/PointCloud2Message. . – – URL. – Abruf: 2018-04-04

github.com/ros drivers DRIVERS github.com/ros: ROS device drivers. . – URL https://github.com/ros-drivers. – Abruf: 2018-03-14

Flir 2017a FLIR: Ladybug5 30 MP USB 3 sphaerische digitale Video-Kamera, rot. (2017). – URL https://www.ptgrey.com/ladybug5-plus-30-mp-usb-3-spherical-digital- – Abruf: 2017-12-07

Flir 2017b FLIR: *Technical Reference Flir Ladybug 5+ Version 1.0*. -: , 2017. – URL https://www.ptgrey.com/support/downloads/10128. – Abruf: 2017-12-08

Gasser u. a. 2012 GASSER, Tom M. ; ARZT, Clemens ; AYOUBI, Mihiar ; BARTELS, Arne ; EIER, Jana ; FLEMISCH, Frank ; HÃCKER, Dirk ; HESSE, Tobias ; HUBER, Werner ; LOTZ, Christine ; MAURER, Markus ; RUTH-SCHUMACHER, Simone ; SCHWARZ, JÃ¼rgen ; VOGT, Wolfgang: Rechtsfolgen zunehmender Fahrzeugautomatisierung. In: *Forschung kompakt 11/12* (2012). – URL http://www.bast.de/DE/Publikationen/Foko/Downloads/2012-11.pdf?__blob=publicationFile. – Abruf: 2018-02-19

Gleitsmann und Audi-Deutschland GLEITSMANN, Jan ; AUDI-DEUTSCHLAND: Eine groÃe Anzahl von Sensoren sowie redundante Sensortechnologien stellen eine zuverlÃssige Erkennung der Umgebung sicher. . – URL https://www.mobilegeeks.de/artikel/ein-selbstversuch-pilotiertes-fahren-mit-dem-audi-a7-jack/. – Abruf: 2018-03-22

Graupner GRAUPNER: Anleitung Polaron SMPS 1500. . – URL https://www.graupner.de/media/pdf/4c/40/5d/S2024_Polaron_SMPS_1500_de5a31183639296.pdf. – Abruf: 2018-04-11

Grisetti u. a. GRISETTI, Giorgio ; KÃ¼MMERLE, Rainer ; STACHNISS, Cyrill ; BURGARD, Wolfram: A Tutorial on Graph-Based SLAM. . – URL http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti10titsmag.pdf. – Abruf: 2018-03-29

Hornung u. a. 2013 HORNUNG, Armin ; WURM, Kai M. ; BENNEWITZ, Maren ; STACHNISS, Cyrill ; BURGARD, Wolfram: OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. In: *Autonomous Robots* (2013). – URL http://www2.informatik.uni-freiburg.de/~hornunga/pub/hornung13auro.pdf. – Abruf: 2018-03-02

http://wiki.ros.org/ 2015 HTTP://WIKI.ROS.ORG/: pointgrey$_c amera_d river.(2015).--URL$. – Abruf: 2018-03-14

http://wiki.ros.org 2015 HTTP://WIKI.ROS.ORG: Setting up your robot using tf. (2015). – URL http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF. – Abruf: 2018-03-13

Intel INTEL: Intel Core i7 7700k Prozessor. . – URL https://ark.intel.com/de/products/97129/Intel-Core-i7-7700K-Processor-8M-Cache-up-to-4_50-GHz. – Abruf: 2018-02-12

Janos 2008 JANOS, Tamas: (2008). – URL http://www.tankonyvtar.hu/en/tartalom/tamop425/0032_precizios_mezogazdasag/ch02s04.html. – Abruf: 2018-24-03

janztec.com JANZTEC.COM: Endeavour MB. . – URL https://www.janztec.com/industrie-pc/wall-mounted-pc/endeavour-mb/. – Abruf: 2018-02-09

Kohlbrecher u. a. 2011 KOHLBRECHER, Stefan ; STRYK, Oskar von ; MEYER, Johannes ; KLINGAUF, Uwe: A Flexible and Scalable SLAM System with Full 3D Motion Estimation. (2011). – URL http://www.sim.informatik.tu-darmstadt.de/publ/download/2011_SSRR_KohlbrecherMeyerStrykKlingauf_Flexible_SLAM_System.pdf. – Abruf: 2018-03-01

Lee 1996 LEE, David: *The Map-Building and Exploration Strategies of a Simple Soar-Equipped Mobile Robot*, Diplomarbeit, 1996. – URL https://books.google.de/books?id=9eEYRZA1zroC&pg=PA29&lpg=PA29&dq=feature+based+maps+efficient&source=bl&ots=qjtW6WSLbH&sig=fOb2Qs2Kj0BbWoVMFfnU6oXFV_0&hl=de&sa=X&ved=0ahUKEwj9urbKgY_aAhWrLsAKHSPlCQ4Q6AEIUDAD#v=onepage&q=feature%20based%20maps%20efficient&f=false. – Abruf: 2018-03-28

Maurer u. a. 2015 MAURER, Markus (Hrsg.) ; GERDES, J. C. (Hrsg.) ; LENZ, Barbara (Hrsg.) ; WINNER, Hermann (Hrsg.): *Autonomes Fahren*. Springer Berlin Heidelberg, 2015

Micro-Epsilon 2018 MICRO-EPSILON: Laser-Triangulation. (2018). – URL https://www.micro-epsilon.de/service/glossar/Laser-Triangulation.html. – Abruf: 2018-03-24

Microsoft 2007 MICROSOFT: Roboter steuern mit Microsoft Robotics Studio. (2007). – URL https://msdn.microsoft.com/de-de/library/bb483065.aspx. – Abruf: 2018-02-28

mrpt.org MRPT.ORG: MRPT Empowering C++ development in robotics. . – URL https://www.mrpt.org/. – Abruf: 2018-02-28

NokiaHere 2014 NOKIAHERE: Nokia Here mapping Car. (2014). – URL https://www.wired.com/2014/12/nokia-here-autonomous-car-maps/. – Abruf: 2018-24-03

O'Kane 2013 O'KANE, Jason M.: *A Gentle Introduction to ROS*. CreateSpace Independent Publishing Platform, 2013. – URL https://www.amazon.com/Gentle-Introduction-ROS-Jason-OKane/dp/1492143235?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1492143235. – ISBN 978-14-92143-23-9

picoquant.com PICOQUANT.COM: LIDAR/Ranging/SLR. . – URL https://www.picoquant.com/applications/category/metrology/lidar-ranging-slr. – Abruf: 2018-03-24

github.com/tu-darmstadt-ros pkg/ PKG/ github.com/tu-darmstadt-ros: tu-darmstadt-ros-pkg/hector$_slam.. --URL$. – Abruf: 2018-03-15

Proxitron 2018 PROXITRON: Leddar VU8 Multisegment Sensor. (2018). – URL https://lidar-sensor.de/produkte/module/leddar-vu8/. – Abruf: 2018-03-25

Rauch u. a. RAUCH, Sebastian ; AEBERHARD, Michael ; KAEMPCHEN, Michael N.: Autonomes Fahren auf der Autobahn - Eine Potentialstudie fuer zukuenftige Fahrerassistenzsysteme. . – URL http://www.ftm.mw.tum.de/uploads/media/23_Aeberhard.pdf. – Abruf: 2017-12-11

rosindustrial.org ROSINDUSTRIAL.ORG: ROS-Industrial is an open-source project that extends the advanced capabilities of ROS software to manufacturing. . – URL rosindustrial.org/. – Abruf: 2018-04-04

Rudolph und Voelzke 2017 RUDOLPH, Gert ; VOELZKE, Uwe: Welche Rolle spielt Lidar fÃ¼r autonomes Fahren ? Und welche Radar und Kamera ? ein Vergleich. (2017). – URL http://www.all-electronics.de/welche-rolle-spielt-lidar-fuer-autonomes-fahren-und-welche-radar-und-k – Abruf: 2018-03-24

SAE 2014 SAE: Automated Driving. (2014). – URL https://web.archive.org/web/20170903105244/https://www.sae.org/misc/pdfs/automated_driving.pdf. – Abruf: 2018-02-19

Stachniss STACHNISS, Cyrill: Robot Mapping FastSLAM - Feature-bases SLAM with Particle Filters. . – URL http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/slam10-fastslam-4.pdf. – Abruf: 2018-04-01

Stachniss 2013 STACHNISS, Cyrill: Short Introduction to Particle Filters and Monte Carlo Localization. (2013). – URL http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam11-particle-filter.pdf. – Abruf: 2018-04-11

Statistisches Bundesamt STATISTISCHES BUNDESAMT: Anzahl der Verkehrstoten im StraÃenverkehr in Deutschland von 1991 bis 2016. . – URL https://de.statista.com/statistik/daten/studie/185/umfrage/todesfaelle-im-strassenverkehr/. – Abruf: 2018-04-10

synoceantech SYNOCEANTECH: Â±12$VDCIndustrialPCPS2ATXPowerSupply$(170$mmlength$).. – $-URL$. – Abruf: 2018-12-02

t3n.de 2017 T3N.DE: US-Regierungsstudie: Teslas Autopilot hilft, UnfÃlle zu verhindern. (2017). – URL https://t3n.de/news/tesla-autopilot-unfaelle-787327/. – Abruf: 2018-02-19

teamhector 2016 TEAMHECTOR: (2016). – URL http://www.teamhector.de/. – Abruf: 2018-03-01

Tesla 2018 TESLA: Hardware fÃ¼r autonomes Fahren in allen Fahrzeugen. (2018). – URL https://www.tesla.com/de_DE/autopilot?redirect=no. – Abruf: 2018-03-24

tesla.com TESLA.COM: Model S. . – URL https://www.tesla.com/de_DE/models. – Abruf: 2018-04-11

Thrun u. a. 2005 THRUN, Sebastian ; BURGARD, Wolfram ; FOX, Dieter: *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*. The MIT Press, 2005. – URL https://www.amazon.com/Probabilistic-Robotics-Intelligent-Autonomous-Ag dp/0262201623?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0262201623. – ISBN 978-0-262-20162-9

Vardhan 2017 VARDHAN, Harsha: HD Maps: New age maps powering autonomous vehicles. (2017). – URL https://www.geospatialworld.net/blogs/hd-maps-autonomous-vehicles/. – Abruf: 2018-04-10

Velodyne 2016 VELODYNE: User's Manual and Programming Guide. (2016). – URL http://velodynelidar.com/docs/manuals/63-9113%20REV%20K%20MANUAL,USER'S,HDL-32E_Outlined.pdf. – Abruf: 2017-12-07

Velodyne 2017 VELODYNE: Velodyne LiDar HDL-32E. (2017). – URL http://velodynelidar.com/docs/datasheet/97-0038_Rev%20K_%20HDL-32E_Datasheet_Web.pdf. – Abruf: 2017-12-07

VelodyneLidar 2013 VELODYNELIDAR: HDL-32E Envelope Drawing. (2013). – URL http://velodynelidar.com/docs/drawings/86-0106%20REV%20B,%20ENVELOPE,%20HDL-32E.pdf. – Abruf: 2018-04-08

velodynelidar.com VELODYNELIDAR.COM: HDL-32E. . – URL http://velodynelidar.com/hdl-32e.html. – Abruf: 2018-04-10

VictronEnergy 2017 VICTRONENERGY: *Victron Energy Blue Power Power Peak Pack*. -: Victron Energy B. V. (Veranst.), 2017. – URL https://www.victronenergy.de/upload/documents/Manual-Peak-Power-Pack-EN-NL-FR-DE-ES-SE.pdf. – Abruf: 2018-02-09

wiki.ors.org 2017 WIKI.ORS.ORG: Ubuntu install of ROS Kinetic. (2017). – URL http://wiki.ros.org/kinetic/Installation/Ubuntu. – Abruf: 2018-03-07

wikipedia.de 2018 WIKIPEDIA.DE: Robot Operating System. (2018). – URL https://de.wikipedia.org/wiki/Robot_Operating_System. – Abruf: 2018-14-02

wiki.ros.org 2016a WIKI.ROS.ORG: camera1394. (2016). – URL http://wiki.ros.org/camera1394. – Abruf: 2018-04-05

wiki.ros.org 2016b WIKI.ROS.ORG: velodyne$_{d}river$. (2016). $--URL$. – Abruf: 2018-03-10

wiki.ros.org 2017a WIKI.ROS.ORG: (2017). – URL http://wiki.ros.org/catkin/CMakeLists.txt. – Abruf: 2018-02-15

wiki.ros.org 2017b WIKI.ROS.ORG: Ubuntu install of ROS Kinetic. (2017). – URL http://wiki.ros.org/kinetic/Installation/Ubuntu. – Abruf: 2018-04-11

wiki.ros.org 2018 WIKI.ROS.ORG: rosbag/ Commandline. (2018). – URL http://wiki.ros.org/rosbag/Commandline. – Abruf: 2018-03-14

wiki.ros.org/octomap WIKI.ROS.ORG/OCTOMAP: octomap Package Summary. . – URL http://wiki.ros.org/octomap. – Abruf: 2018-03-15

wiki.ros.org/tf/Troubleshooting 2009 WIKI.ROS.ORG/TF/TROUBLESHOOTING: tf Troubleshooting. (2009). – URL http://wiki.ros.org/tf/Troubleshooting. – Abruf: 2018-03-19

Wolfram Burgard WOLFRAM BURGARD, Maren Bennewitz Diego Tipaldi Luciano S.: Introduction to Mobile Robotics SLAM - Grid-based FastSLAM. . – URL http://ais.informatik.uni-freiburg.de/teaching/ss13/robotics/slides/15-slam-gridrbpf.pdf. – Abruf: 2018-04-01

oliver Wulf u. a. 2004 WULF oliver ; ARRAS, Kai O. ; CHRISTENSEN, Henrik I. ; WAGNER, Bernardo: 2D Mapping of Cluttered Indoor Environments by Means of 3D Perception. In: ON, IEEE International C. (Hrsg.): *Robotics & Automation* Bd. 4 (April 26 2004-May 1 2004). IEEE, 2004, S. 4204–4208. – URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1308934. – Abruf: 2018-03-29

xyht.com 2017 XYHT.COM: Unmanned Lidar in the Air. (2017). – URL http://www.xyht.com/aerialuas/unmanned-lidar-air/. – Abruf: 2018-04-10

A. Anhang

Der Anhang umfasst nachfolgend aufgelistete Ausdrucke von Dateien:

- tutorial.launch (Hector-SLAM)
- default_mapping.launch (Hector-SLAM)
- octomap_velodyne.launch (OctoMap)
- ein Ausschnitt aus der VelodyneLaserScan.cpp (Velodyne Node)

Auf CD sind folgende Dateien hinterlegt:

- Die Bachelorarbeit in digitaler Form (PDF)
- Die Launch-Dateien der verwendeten ROS-Pakete
- Die VelodyneLaserScan.cpp Datei der ROS Velodyne-Treiber
- Die CAD-Dateien der Konstruktionen

tutorial.launch

```
<?xml version="1.0"?>

<launch>

  <arg name="geotiff_map_file_path" default="$(find
hector_geotiff)/maps"/>

  <param name="/use_sim_time" value="true"/>

  <node name="world_to_map"
      pkg="tf"
      type="static_transform_publisher"
      args="0 0 0 0 0 0 /world /map 100" />

  <node name="base_to_velodyne"
      pkg="tf" type="static_transform_publisher"
      args="0.0 0.0 2.0 0.0 0.0 0.0 /base_footprint /velodyne
10" />

  <node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find
hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>

  <include file="$(find
hector_mapping)/launch/mapping_default.launch"/>

  <include file="$(find
hector_geotiff)/launch/geotiff_mapper.launch">
    <arg name="trajectory_source_frame_name"
value="scanmatcher_frame"/>
    <arg name="map_file_path" value="$(arg
geotiff_map_file_path)"/>
  </include>

</launch>
```

mapping_default.launch

```
<?xml version="1.0"?>

<launch>
  <arg name="tf_map_scanmatch_transform_frame_name"
default="scanmatcher_frame"/>
  <arg name="base_frame" default="base_footprint"/>
  <arg name="odom_frame" default="base_footprint"/>
  <arg name="pub_map_odom_transform" default="true"/>
  <arg name="scan_subscriber_queue_size" default="5"/>
  <arg name="scan_topic" default="scan"/>
  <arg name="map_size" default="15000"/>

  <node pkg="hector_mapping" type="hector_mapping"
name="hector_mapping" output="screen">

    <!-- Frame names -->
    <param name="map_frame" value="map" />
    <param name="base_frame" value="$(arg base_frame)" />
    <param name="odom_frame" value="$(arg odom_frame)" />

    <!-- Tf use -->
    <param name="use_tf_scan_transformation" value="true"/>
    <param name="use_tf_pose_start_estimate" value="false"/>
    <param name="pub_map_odom_transform" value="$(arg
pub_map_odom_transform)"/>

    <!-- Map size / start point -->
    <param name="map_resolution" value="0.010"/>
    <param name="map_size" value="$(arg map_size)"/>
    <param name="map_start_x" value="0.5"/>
    <param name="map_start_y" value="0.5" />
    <param name="map_multi_res_levels" value="4" />

    <!-- Map update parameters -->
    <param name="update_factor_free" value="0.4"/>
    <param name="update_factor_occupied" value="0.9" />
    <param name="map_update_distance_thresh" value="0.02"/>
    <param name="map_update_angle_thresh" value="0.01" />
    <param name="laser_z_min_value" value = "-1.0" />
    <param name="laser_z_max_value" value = "1.0" />
    <param name="laser_min_dist" value = "0.8" />
    <param name="laser_max_dist" value = "12"/>
    <!-- Advertising config -->
    <param name="advertise_map_service" value="true"/>

    <param name="scan_subscriber_queue_size" value="$(arg
scan_subscriber_queue_size)"/>
    <param name="scan_topic" value="$(arg scan_topic)"/>

    <!-- Debug parameters -->
    <!--
      <param name="output_timing" value="false"/>
      <param name="pub_drawings" value="true"/>
      <param name="pub_debug_output" value="true"/>
```

```
    -->
    <param name="tf_map_scanmatch_transform_frame_name"
value="$(arg tf_map_scanmatch_transform_frame_name)" />
  </node>

  <!--<node pkg="tf" type="static_transform_publisher"
name="map_nav_broadcaster" args="0 0 0 0 0 0 map nav 100"/>-->
</launch>
```

octomap_velodyne.launch

```
<launch>
     <node pkg="octomap_server" type="octomap_server_node"
name="octomap_server">
          <param name="resolution" value="0.04" />

          <!-- fixed map frame (set to 'map' if SLAM or
localization running!) -->
          <param name="frame_id" type="string" value="map" />

          <!-- maximum range to integrate (speedup!) -->
          <param name="sensor_model/max_range" value="12.0"
/>

          <!-- data source to integrate (PointCloud2) -->
          <remap from="cloud_in" to="/velodyne_points" />
     </node>

<!--  <node pkg="tf"
           type="static_transform_publisher"
             name="map_broadcaster"
             args="1 0 0 0 0 0 1 velodyne map 100" />
-->


</launch>
```

VelodyneLaserScan.cpp

```
#include "VelodyneLaserScan.h"
#include <sensor_msgs/point_cloud2_iterator.h>

namespace velodyne_laserscan {

VelodyneLaserScan::VelodyneLaserScan(ros::NodeHandle &nh,
ros::NodeHandle &nh_priv) :
    ring_count_(0), nh_(nh), srv_(nh_priv)
{
  ros::SubscriberStatusCallback connect_cb =
boost::bind(&VelodyneLaserScan::connectCb, this);
  pub_ = nh.advertise<sensor_msgs::LaserScan>("scan", 10,
connect_cb, connect_cb);

  srv_.setCallback(boost::bind(&VelodyneLaserScan::reconfig,
this, _1, _2));
}

void VelodyneLaserScan::connectCb()
{
  boost::lock_guard<boost::mutex> lock(connect_mutex_);
  if (!pub_.getNumSubscribers()) {
    sub_.shutdown();
  } else if (!sub_) {
    sub_ = nh_.subscribe("velodyne_points", 10,
&VelodyneLaserScan::recvCallback, this);
  }
}

void VelodyneLaserScan::recvCallback(const
sensor_msgs::PointCloud2ConstPtr& msg)
{
  // Latch ring count
  if (!ring_count_) {
    // Check for PointCloud2 field 'ring'
    bool found = false;
    for (size_t i = 0; i < msg->fields.size(); i++) {
      if (msg->fields[i].datatype ==
sensor_msgs::PointField::UINT16) {
        if (msg->fields[i].name == "ring") {
          found = true;
          break;
        }
      }
    }
    if (!found) {
      ROS_ERROR("VelodyneLaserScan: Field 'ring' of type
'UINT16' not present in PointCloud2");
      return;
    }
    for (sensor_msgs::PointCloud2ConstIterator<uint16_t>
it(*msg, "ring"); it != it.end(); ++it) {
      const uint16_t ring = *it;
```

```
      if (ring + 1 > ring_count_) {
        ring_count_ = ring + 1;
      }
    }
    if (ring_count_) {
      ROS_INFO("VelodyneLaserScan: Latched ring count of %u",
ring_count_);
    } else {
      ROS_ERROR("VelodyneLaserScan: Field 'ring' of type
'UINT16' not present in PointCloud2");
      return;
    }
  }

  // Select ring to use
  uint16_t ring;
  if ((cfg_.ring < 0) || (cfg_.ring >= ring_count_)) {
    // Default to ring closest to being level for each known
sensor
    if (ring_count_ > 32) {
      ring = 57; // HDL-64E
    } else if (ring_count_ > 16) {
      ring = 19; // HDL-32E
    } else {
      ring = 8; // VLP-16
    }
  } else {
    ring = cfg_.ring;
  }
  ROS_INFO_ONCE("VelodyneLaserScan: Extracting ring %u",
ring);
```

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 11. April 2018	
Ort, Datum	Unterschrift