Otto-von-Guericke Universität Magdeburg

Fakultät für Informatik

Evaluation of Velodyne sensors for outdoor environments

Sensors for outdoor robotics

Teamprojektbericht

Alexander Wagner (217884)

Sommersemester 2018

Professor: Jun.-Prof. Dr. Sebastian Zug

Betreuer: Maik Riestock

Inhaltsverzeichnis

1. Einleitung

Das Ziel dieser Ausarbeitung ist die Evaluierung eines Velodyne VLP-16 LIDAR (Light Detection and Ranging) Sensors für die Lokalisierung in Outdoorszenarien. Dabei wird untersucht, inwieweit der VLP-16 unter Outdooreinflüssen als Referenzsystem zur Erstellung von Karten der Umgebung genutzt werden kann. Für die Generierung dieser Karten können SLAM (Simultaneous Localization and Mapping) Verfahren verwendet werden, die aus den aufgenommenen Lidardaten iterativ eine Karte aufbauen. Dafür werden im Kontext dieser Ausarbeitung bestehende, kostenfreie LIDAR SLAM Implementierungen, mit Hilfe einer stichwortbasierten Github Suche recherchiert, deren Funktionsweise verglichen und die Ergebnisse anhand von Testdaten miteinander verglichen. Die Aufnahme der Daten erfolgt auf dem Gebiet der Otto-von-Guericke Universität Magdeburg. Die Datenerfassung erfolgt mit Hilfe des „Transformes“ Projekts der Otto-von-Guericke Universität, einem autonom fahrenden Fahrrad. Abschließend erfolgt eine Evaluierung der aufgenommenen Daten im Hinblick auf die Qualität der erzeugten Karte unter Berücksichtigung der Einflüsse der unterschiedlichen Aufnahmebedingungen und der genutzten Bibliothek.

Die Arbeit gliedert sich in sechs Kapitel. In Kapitel 2 Stand der Technik erfolgt eine Zusammenfassung relevanter Literatur auf dem Themengebiet der SLAM Algorithmen, sowie deren Einsatz in Outdoorszenarien. Des Weiteren erfolgt die Beschreibung der Recherchemethodik zur Identifizierung relevanter SLAM Implementierung sowie die Ergebnisse dieser Recherche. In Kapitel 3 erfolgt die genaue Beschreibung der Anforderungen an die SLAM Bibliotheken sowie die Untersuchung der in Kapitel 2 identifizierten Bibliotheken im Hinblick auf diese Kriterien. Außerdem erfolgt eine Beschreibung des Velodyne VLP-16 sowie die Beschreibung des Versuchsaufbaus und der gefahrenen Route für die Datenaufnahme. Des Weiteren werden die Testbedingungen definiert, mit denen der Sensor getestet werden sollte. Die Beschreibung der Inbetriebnahme des gesamten Versuchsaufbaus sowie die Installation der verwendeten Algorithmen erfolgt in Kapitel 4. Die Evaluierung der generierten Karten im Hinblick auf die verschiedenen Bibliotheken und die unterschiedlichen Outdoorszenarien erfolgt in Kapitel 5. In Kapitel 6 wird ein Ausblick gegeben, inwieweit eine Erkennung von Objekten in den Lidardaten möglich ist und eine mögliche Bibliothek für diesen Zweck beschrieben.

2. Stand der Technik

In diesem Kapitel erfolgt eine kurze Beschreibung des generellen SLAM Problems sowie die Zusammenfassung des aktuellen Forschungstandes im Hinblick auf den Einsatz und Herausforderungen bei der Anwendung in Outdoor Szenarien. Des Weiteren wird eine Rechercheübersicht existierender SLAM Implementierungen präsentiert. Für die Identifizierung relevanter Forschungsarbeiten und deren Implementierungen, wurde eine stichwortbasierte Google Scholar sowie Github Recherche durchgeführt.

2.1 Literatur

Das Forschungsgebiet der SLAM Algorithmen beschäftigt sich mit der Fragestellung, wie mobile Roboter eine Karte einer unbekannten Umgebung erzeugen und gleichzeitig diese Karte zur eigenen Lokalisierung nutzen können. Die Grundlagen von SLAM Algorithmen werden in [1], [2] detailliert beschrieben. Es erfolgt eine generelle Unterscheidung zwischen EKF-SLAM (Extended Kalman Filter) und FastSLAM (Rao-Blackwellised particle filters). In [3] werden drei Phasen der Entwicklung der SLAM Algorithmen beschrieben. Laut [3] befinden wir uns derzeit in der dritten Phase, dem „robust-perception age“, in dem es darum geht, robustere und effizientere SLAM Lösungen zu finden. Das generell SLAM Problem gilt nämlich für spezifische Situationen, mit bestimmter Sensorik als weitestgehend gelöst (Vgl. [1]). Es existieren für unterschiedliche Einsatzszenarien mit unterschiedlicher Sensorik, verschiedene Umsetzungen. Allerdings existieren offene Probleme, bei dem Einsatz eines SLAMs in Outdoorszenarien. Im Gegensatz zu Anwendungen im Inneren, treten bei Outdooranwendungen unterschiedliche Wetterbedingungen auf. Die unterschiedlichen Wettereinflüsse haben einen Einfluss auf die Aufnahmequalität der Sensorik und somit auch auf die Ergebnisse der SLAM Algorithmen. Im Rahmen dieser Literaturrecherche wurden Informationen zu den Einflüssen der Wetterbedingungen Sonnenschein, Regen und Schnee gefunden. Generell können durch Wettereinflüsse unterschiedliche Probleme auftreten. Zum einen können Wettereinflüsse die Laserstrahlen abschwächen und zum anderen aber auch reflektieren, wodurch möglicherweise Hindernisse erkannt werden, die in der Realität gar nicht vorhanden sind.

Ein Überblick und eine grundlegende Einführung über den Einfluss von Wetterbedingungen auf LIDAR Sensorik wird in [4] beschrieben. Bei Regen, Nebel und Schnee kann es vor allem im nahen Bereich vor dem Sensor zu Reflektionen und somit falschen Ergebnissen kommen. Regen und Nebel bestehen aus kleinen, unterschiedlich großen Wassertropfen, die die Laserstrahlen unterschiedlich stark brechen und reflektieren. Schnee hingegen besteht aus verschieden großen Eiskristallen. Es wurde festgestellt, dass die Verteilung der Größen der Regentropfen sowie der Eiskristalle angenähert und in den Datenaufnahmeprozess integriert werden kann.

Die Konfigurationen und Parameter der Verteilungen für die unterschiedlichen Formen von Regen und Nebel werden detailliert in [4] beschrieben. Eine genauere Betrachtung der Auswirkungen von Schneefall auf unterschiedliche LIDAR Sensorik erfolgt in [6]. Die dort beschriebenen Testversuche mit unterschiedlichen LIDAR Systemen haben gezeigt, dass der in dieser Arbeit verwendete Sensor, wenig anfällig für Fehler durch Schneefall ist. Des Weiteren werden ab einer Entfernung von 10 Metern keine Reflektionen durch Schneefall vom Sensor mehr festgestellt.

Bei starker Sonneneinstrahlung können in bestimmten Eintrittswinkeln und durch Reflektionen ebenfalls nicht vorhandene Objekte in den Laserdaten entstehen. Allerdings wird in [5] ein Verfahren zur Unterscheindung von echten Lidardaten und Reflektionen durch Sonneneinstrahlung beschrieben.

Zusammenfassend lässt sich festhalten, dass durch unterschiedliche Wettereinflüsse die Gesamtreichweite des Sensors sinkt, die Integration solcher Einflüsse in den Datenaufnahmeprozess aber prinzipiell möglich ist.

Neben den unterschiedlichen Wetterbedingungen muss der eingesetzte SLAM auch fähig sein, mit beweglichen Objekten in den Daten umgehen zu können. Das Problem dabei ist, dass in der generellen Definition des SLAM Problems eine statische Umgebung angenommen wird. Es existieren unterschiedliche Herangehensweisen für den Umgang mit beweglichen Objekten. Diese werden in der Literatur als *„SLAM with generalized Objects“* und *„SLAM with DATMO“* beschrieben (Vgl. [8]).

2.2 Implementierungen

Für die Identifizierung möglicher relevanter Implementierungen wurde eine stichwortbasierte GitHub Repository Suche durchgeführt. Dafür wurden die in Tabelle 1 definierten Stichpunkte verwendet.

Bei der Recherche stellte sich heraus, dass es eine vergleichsweise hohe Anzahl an Repositories gibt, die das Stichwort „SLAM“ beinhalten, aber die Anzahl der vorhandenen Repositories, die einen Velodyne nutzen, beziehungsweise einen SLAM implementieren, der mit Velodyne Daten arbeiten kann, erstaunlich gering ist.

Stichwort	**Anzahl an Repositories**
SLAM	4693
ROS SLAM	284
Velodyne	243
Simultaneous Localization and Mapping	101
LIDAR SLAM	71
VLP-16	32
Velodyne SLAM	15

Tabelle 1 Stichwortbasierte Github Repository Suche[1]

Die Ergebnisse der Github Suche wurden über zusätzliche Suchparameter eingeschränkt. Dabei wurden die Implementierungssprachen auf C++, Python und C eingeschränkt und der Scope der Suche auf den Inhalt des gesamten Repositorys, somit auf Titel, Beschreibung, etc. festgelegt. Nach dem Anwenden der definierten Filter ergaben sich die in Tabelle 2 dargestellten Ergebnisse.

Stichwort	**Anzahl an Repositories**
SLAM	1772
ROS SLAM	180
Velodyne	161
Simultaneous Localization and Mapping	52
LIDAR SLAM	37
VLP-16	17
Velodyne SLAM	12

Tabelle 2 Stichwortbasierte Github Repository Suche mit Filterung

[1] Die Ergebnisse wurden mit der Suchfunktion auf https://github.com/ am 14.07.18 erzielt

Für sämtliche Stichworte wurden die Ergebnisse nach „Best match“ (Übereinstimmung des Repository mit dem Suchbegriff), „Most stars“ (Interesse der Github Community an dem Projekt) und „Recently updated“ (Aktualität und Pflege des Repositorys) sortiert. Dadurch wurden die in Tabelle 3 beschriebenen Bibliotheken als relevant identifiziert und für die genauere Betrachtung ausgewählt.

Nr.	Repository	Stichwort
1	https://github.com/laboshinl/loam_velodyne	Velodyne SLAM
2	https://github.com/daobilige-su/loam_velodyne	Velodyne SLAM
3	https://github.com/koide3/hdl_graph_slam	Velodyne SLAM
4	https://github.com/victl/VeloSLAM	Velodyne SLAM
6	https://github.com/VerseChow/FETCH_ROBOT_SLAM	LIDAR SLAM
7	https://github.com/googlecartographer/cartographer	Simultaneous Localization and Mapping
8	https://github.com/zdzhaoyong/GSLAM	Simultaneous Localization and Mapping
9	https://github.com/aharmat/mcptam	Simultaneous Localization and Mapping
10	https://github.com/felixendres/rgbdslam_v2	ROS SLAM
11	https://github.com/tu-darmstadt-ros-pkg/hector_slam	ROS SLAM
12	https://github.com/ros-perception/slam_gmapping	ROS SLAM
13	https://github.com/ethz-asl/okvis_ros	ROS SLAM
14	https://github.com/raulmur/ORB_SLAM2	SLAM
15	https://github.com/tum-vision/lsd_slam	SLAM
16	https://github.com/raulmur/ORB_SLAM	SLAM

Tabelle 3 Relevante Repositories der Github Recherche

Des Weiteren wurde die folgende Bibliothek über Google und YouTube Recherchen gefunden:

Nr.	Repository	Stichwort
17	https://github.com/erik-nelson/blam	Outdoor SLAM

Tabelle 4 Weitere Google und YouTube Recherche

Insgesamt wurden 17 potentiell nutzbare Repositories identifiziert, die in Kapitel 3.2 genauer untersucht werden.

3. Konzept

Die zu nutzende SLAM Implementierung muss spezifischen Kriterien genügen. Dafür erfolgt in Kapitel 3.1 eine Definition der verpflichtenden sowie der optionalen Anforderungen. In Kapitel 3.2. erfolgt eine Auswahl, der in Tabelle 3 und 4 identifizierten Bibliotheken, auf Basis dieser Anforderungen. In Kapitel 3.3. wird der Versuchsaufbau zur Datenaufnahme sowie die genutzte Sensorik detailliert beschrieben. Die Beschreibungen von möglichen Testbedingungen und einer möglichen Strecke zur Datenaufnahme auf dem Universitätscampus erfolgt in Kapitel 3.5 und 3.6.

3.1. Anforderungen

Die Anforderungen an die zu nutzende SLAM Bibliothek lassen sich in verpflichtende und optionale Anforderungen unterteilen (siehe Tabelle 5). Die Anforderungen Nr. 1, 2, 3 und 5 waren bereits im Vorfeld dieser Arbeit vorgegeben, während die Anforderungen Nr. 4 und Nr. 6 im Rahmen der Bearbeitung dieses Projektes definiert wurden. Die Bibliothek muss als ROS Paket vorhanden sein, da die Ergebnisse dieser Ausarbeitung womöglich in ein bestehendes (C++) Projekt integriert werden müssen, in dem sämtliche Komponenten in ROS umgesetzt wurden. Des Weiteren muss die Bibliothek die Nutzung eines Velodyne VLP-16 beziehungsweise die Einbindung der Daten des Sensors unterstützen, da dies der zu untersuchende Sensor ist. Außerdem muss die Bibliothek kostenfrei verfügbar sein, da sie möglicherweise in ein universitäres Forschungsprojekt integriert wird. Ein Forschungspaper als Basis der Implementierung wäre sehr nützlich, da dadurch eine theoretische, dokumentierte und nachvollziehbare Beschreibung der genutzten Algorithmen vorhanden wäre. Die Spezialisierung der Bibliothek auf den Einsatz in Outdoorszenarien wäre ebenfalls sehr hilfreich, da dadurch auch die Unterschiede von „herkömmlichen“ SLAM Bibliotheken, mit spezialisierten Outdoor SLAM Bibliotheken erarbeitet werden könnte.

Nr.	**Anforderung**	**Relevanz**
1	Installation/Nutzung als ROS Paket	Pflicht
2	Unterstützung für Velodyne VLP-16 Daten	Pflicht
3	Kostenfrei verfügbar	Pflicht
4	Basierend auf einem veröffentlichten Forschungspaper	Optional
5	Umsetzung in C++	Optional
6	Speziell für Outdoorszenarien entwickelt	Optional

Tabelle 5 Anforderungen an die SLAM Implementierung

3.2. Lösungsauswahl

Die Auswahl der zu nutzenden SLAM Bibliotheken erfolgt anhand der in Abschnitt 3.1 definierten Anforderungen und ist in Abbildung 1 dargestellt. Die Farbskala am rechten Rand gibt an, ob eine Bibliothek für eine genauere Untersuchung installiert wird.

Nr.	Repository	ROS Paket	ROS Version	Eingabedatentyp	Vlp-16	Bag files	Lizenz	Referenz-paper	Programmier-sprache	Ergebnis
1	https://github.com/laboshinl/loam_velodyne	√	Indigo	Velodyne LIDAR	√	√	All rigths reserved	[9]	C++	
2	https://github.com/daobilige-su/loam_velodyne	√	Indigo	Velodyne LIDAR	√	√	All rigths reserved	[9]	C++	
3	https://github.com/koide3/hdl_graph_slam	√	Indigo	3D LIDAR	√	√	?	[10]	C++	
4	https://github.com/victl/VeloSLAM	X	X	LIDAR	X	X	?	X	C++	
6	https://github.com/VerseChow/FETCH_ROBOT_SLAM	√	Indigo	LIDAR	X	X	?	X	C++	
7	https://github.com/googlecartographer/cartographer	√	Indigo Kinetic	2D/3D LIDAR	√	√	Apache License 2.0	[11]	C++	
8	https://github.com/zdzhaoyong/GSLAM	X	X	Monocular, Stereo, RGB-D	X	X	BSD 2-Clause License	X	C++	
9	https://github.com/aharmat/mcptam	√	Hydro	RGB-D	X	X	GNU General Public License v3.0	[12]	C++	
10	https://github.com/felixendres/rgbdslam_v2	√	Kinetic	RBG-D	X	X	GNU General Public License v3.0	[13]	C++	
11	https://github.com/tu-darmstadt-ros-pkg/hector_slam	√	Kinetic, Indigo	2D LIDAR	X	X	?	[14]	C++	
12	https://github.com/ros-perception/slam_gmapping	√	Kinetic, Indigo, Lunar	2D laser range data and odometry	X	X	?	[15]	C++	
13	https://github.com/ethz-asl/okvis_ros	√	Hydro,Indigo, Jade	Stereo	X	X	No license, but citation of paper	[16] - [18]	C++	
14	https://github.com/raulmur/ORB_SLAM2	√	Indigo	Monocular, Stereo, RGB-D	X	X	GNU General Public License v3.0	[19]	C++	
15	https://github.com/tum-vision/lsd_slam	√	Indigo	Monocular	X	X	GNU General Public License v3.0	[20], [21]	C++	
16	https://github.com/raulmur/ORB_SLAM	√	Indigo	Monocular, Stereo, RGB-D	X	X	GPLv3 license	[22]	C++	
17	https://github.com/erik-nelson/blam	√	Indigo and newer	Velodyne LIDAR	√	√	All rigths reserved	X	C++	

Abbildung 1 Vergleich verfügbarer SLAM Bibliotheken (GIthub)

Insgesamt erfüllen wenig der in Kapitel 2.3 recherchierten Implementierungen, den in Tabelle 5 definierten Anforderungen. Bibliotheken ohne ein ROS Interface können nicht verwendet werden (Nr. 4, 8). Ebenso können keine Bibliotheken verwendet werden, die nicht mit LIDAR Daten, beziehungsweise Daten des Velodyne VLP-16 arbeiten können (Nr. 6, 9, 10, 11, 12, 13, 14, 15, 16). Möglicherweise können diese Bibliotheken angepasst werden, sodass sie mit den Daten des Velodyne VLP-16 arbeiten können. Dies ist nicht Teil dieser Ausarbeitung.

Die verbleibenden Implementierungen (Nr. 1, 2, 3, 7, 17) müssen für eine genauere Untersuchung installiert und getestet werden (siehe Kapitel 4.3). Die Bibliotheken erfüllen die verpflichtenden Kriterien Nr. 1-3 und weitestgehend (Ausnahme Bibliothek Nr. 17) auch die optionalen Anforderungen Nr. 4, 6. Leider existiert unter den recherchierten Bibliotheken keine Implementierung, die speziell für den Outdooreinsatz entwickelt wurde.

3.3. Versuchsaufbau

Der verwendete Velodyne VLP-16 Sensor ist ein LIDAR Sensor mit einer Reichweite von 100 Metern. Der Sensor hat einen vertikalen Blickwinkel von 360°, einen horizontalen Blickwinkel von 30° (±15°) und erzeugt circa 300.000 Punkte pro Sekunde[23]. Der Sensor ist in Abbildung 2 dargestellt.

Für die Aufnahme der Daten wird das „Transformer“ Projekt der Otto-von-Guericke Universität Magdeburg genutzt (siehe Abbildung 3). Das Transformers Projekt ist ein aktuelles Forschungsprojekt, bei dem ein autonom fahrendes Fahrrad entwickelt wird. Der Velodyne Sensor wird in einer Höhe von 80 Zentimetern horizontal, vor dem Lenkrad, auf dem Fahrrad montiert. Die Lidardaten werden als ROS Bagfile aufgezeichnet und können nach den Testfahrten mit den verschiedene SLAM Implementierungen getestet werden.

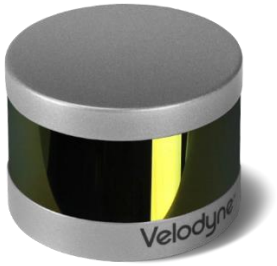

Abbildung 2 Velodyne VLP-16

Abbildung 3 Montage des Velodyne

3.4. Dokumentation

Im Rahmen dieser Ausarbeitung werden sämtliche aufgenommene Daten des Velodyne Sensors gesichert. Die Speicherung dieser Daten erfolgt in Form von ROS Bag Files. Zusätzlich werden alle durchgeführten Testfahrten mit einer Kamera festgehalten. Die final erzeugten Karten, nach Anwendung der SLAM Algorithmen, werden ebenfalls gespeichert. Sämtliche Installationen und Aufnahmen werden auf einem Universitätslaptop durchgeführt.

3.5. Testbedingungen

Um möglichst repräsentative Ergebnisse für die Evaluierung des Einsatzes des Velodyne Sensors in Outdoorszenarien zu liefern, muss der Sensor unter verschiedenen Bedingungen getestet werden, die in einem realen Betrieb auftreten können.

Da das Fahrrad auf dem Universitätscampus Magdeburg bewegt wird, muss es mit verschiedenen beweglichen Hindernissen umgehen können. Das bedeutet der SLAM Algorithmus sollte bewegliche Objekte erkennen und diese nicht in die erzeugten Karten integrieren. Bewegliche Objekte sind nicht fest definiert und umfassen dabei Menschen, Fahrradfahrer, Autofahrer, Straßenbahnen und weitere unvorhergesehene Objekte. Des Weiteren muss bei dem Einsatz in Outdoorszenarien auch der Einfluss des Wetters beachtet werden. Unterschiedliche Witterungen wie Regen, Schnee, Nebel oder direkte Sonneneinstrahlung können einen starken Einfluss auf die Sensorik und die aufgenommenen Daten haben. Der zu untersuchende Velodyne VLP-16 hat ein horizontales Field of View von 360 Grad und ein eingeschränktes vertikales Field of View von 30 Grad. Damit der Velodyne optimale Ergebnisse liefert, muss die Datenaufnahme in verschiedenen Befestigungshöhen sowie unterschiedlichen Neigungswinkeln getestet werden, um vergleichen zu können, welche Einstellung die besten Aufnahmen ermöglicht.

Die Komplexität des SLAM Algorithmus darf außerdem nicht zu hoch werden, da der Algorithmus die Karte in Echtzeit erstellen sollte. Da ROS ein Abspielen der aufgezeichneten Bag Files mit simulierten Zeitstempeln unterstützt, kann die Echtzeitfähigkeit auf unterschiedlichen Systemen auch nach der eigentlichen Datenaufnahme im Nachhinein untersucht werden.

Die Testbedingungen lassen sich insgesamt in vier Punkten zusammenfassen:

1. Bewegliche Objekte
2. Wetterbedingungen
3. Sensorhöhe
4. Echtzeitfähigkeit

3.6 Versuchsgelände

Die Aufnahme der Velodyne Daten erfolgt auf dem in Abbildung 4 dargestellten Gebiet, dem Vorplatz der Mensa der Otto-von-Guericke Universität Magdeburg und der angrenzenden Umgebung. In Abbildung 4 ist eine mögliche Route dargestellt, auf der die Datenaufnahme erfolgen könnte. Die Route hätte folgende Vorteile:

- Räumliche Nähe zur Informatik Fakultät
- Unterschiedliche Testbedingungen (Kapitel 3.5) in dem Gebiet vorhanden
- Unterschiedliche „Gelände“ sind vorhanden (freier Platz vor der Universitätsmensa, normaler Bürgersteig an der Gustav-Adolf-Straße und ein Fahrradweg an der Markgrafstraße)
- Loop Closing kann getestet werden
- Weitere Außeneinflüsse wie Personen, Radfahrer, Autofahrer und auch bisher unbeachtete Einflüsse können gegebenenfalls untersucht werden

Die Route hat eine Länge von 0.57 Kilometer.

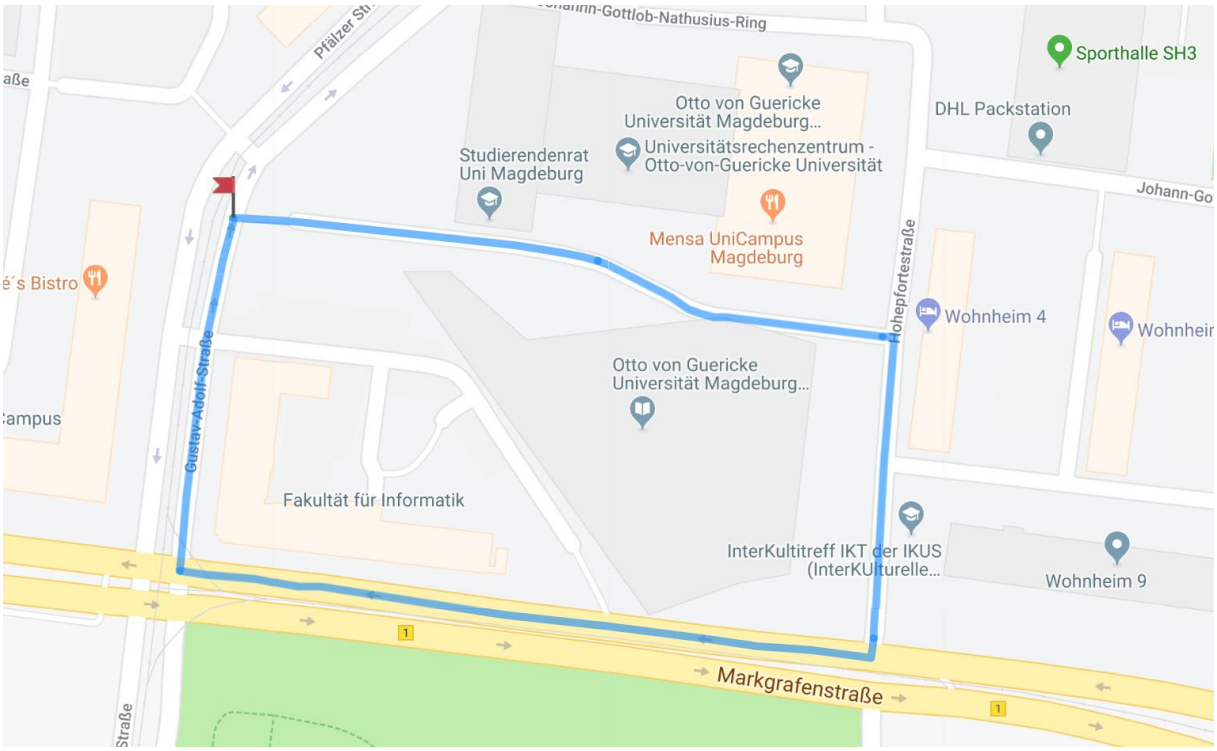

Abbildung 4 Datenaufnahmeroute Otto-von-Guericke Universität Magdeburg[2]

[2] Erstellt mit https://www.mapsdirections.info/de/zeichnen-zie-eine-route-google-maps/ (15.07.2018)

4. Inbetriebnahme

Die Einrichtung des Velodyne VLP-16 und die Installation der SLAM Bibliotheken erfolgt auf einem Universitätslaptop (Dell Latitude E6430 ATG Nr. 3).

4.1 Computer

Für die Vergleichbarkeit und Nachvollziehbarkeit der erzielten Ergebnisse sind die Hardwarekomponenten des genutzten Laptops in Tabelle 6 aufgelistet. Des Weiteren sind die installierten Softwarekomponenten mit den entsprechenden Versionen in Tabelle 7 aufgelistet. Die Softwarekomponenten beinhalten die generelle Software, wie Betriebssystem und ROS-Version sowie sämtliche Bibliotheken, die für die Anwendung der SLAM Algorithmen in Kapitel 4.3 benötigt werden.

Komponente	**Bezeichnung**
Modell	Dell Latitude E6430 ATG
CPU	Intel Core i7-3540M 4*3.00 Ghz
Arbeitsspeicher	8 GB
Grafikkarte	Nvidia NVS 5200M 1 GB
Speicher	Samsung SSD 256 GB

Tabelle 6 Computerhardware

Des Weiteren sind die folgenden Softwarekomponenten installiert.

Komponente	**Version**
Betriebssystem	Ubuntu 16.04 LTS
ROS-Version	ROS Kinetic Kame
cMake	3.5.1
GTSAM	4.X
Boost	1.55.0.2

Tabelle 7 Computersoftware

4.2 Sensor

Der Velodyne VLP-16 Lasersensor wird entsprechend der beiliegenden Betriebsanleitung mit dem Computer und der Stromquelle verbunden. Für die Nutzung des Velodyne unter ROS, werden die Schritte im offiziellen ROS Velodyne Tutorial befolgt (siehe [24]). Die durchzuführenden Schritte lassen sich in drei Punkten zusammenfassen:

1. Konfiguration des Netzwerksadapters des Laptops zur Ethernet Kommunikation
2. Installation der ROS Abhängigkeiten
3. Installation des VLP-16 Treibers

4.3 SLAM Algorithmen

Im Folgenden wird die Installation und das Vorgehen zur Nutzung der in Kapitel 3.2. identifizierten SLAM Bibliotheken beschrieben. Die Voraussetzung für die Installation der Bibliotheken sind ROS Kinetic Kame, Ubuntu 16.04 und cMake. Diese Pakete werden nicht mehr explizit als Voraussetzung erwähnt.

4.3.1 Laser Odometry and Mapping (Loam)

Referenz: Basiert auf [9]

Quelle: https://github.com/laboshinl/loam_velodyne

Installation:

Voraussetzung: Keine

Vorgehen:

1. **Herunterladen des Workspaces:**
   a. *cd ~/catkin_ws/src/*
   b. *git clone https://github.com/laboshinl/loam_velodyne.git*
2. **Installieren der Bibliothek:**
   a. *cd ~/catkin_ws*
   b. *catkin_make -DCMAKE_BUILD_TYPE=Release*
   c. *source ~/catkin_ws/devel/setup.bash*

Dauer: < 0.5 Stunden

Anwendung:

1. Starten des SLAMS: *roslaunch loam_velodyne loam_velodyne.launch*
2. Eingabe der Laserdaten (Mehrere Möglichkeiten):
   - Livedaten (Daten unter Topic **/velodyne_points** publishen):
     - *roslaunch velodyne_pointcloud VLP16_points.launch*
   - Abspielen eines Bag Files:
     - *rosbag play ~/Downloads/velodyne.bag* (Beispielpfad)
   - Lesen eines VLP16 Beispiel PCAP:
     - *pcap:="/home/laboshinl/Downloads/velodyne.pcap"*

Hinweise:

- Keine Unterscheidung zwischen Online und Offline Version
- Eingabelidardaten müssen unter dem ROS Topic */velodyne_points* als *sensor_msgs::PointCloud2* gepublished werden
- Kein Loop Closing
- Verbesserungen dieses SLAMs (zum Teil auch für Outdoor Umgebungen) werden in https://ceen.et.byu.edu/sites/default/files/snrprojects/wolfe_derek.pdf beschrieben
- Parametereinstellungen des SLAMs (https://github.com/laboshinl/loam_velodyne/pull/30):
    - the number of feature regions per scan
    - the size of the curvature region
    - the maximum number of sharp / less sharp corner points per feature region
    - the maximum number of flat surface points per feature region
    - the surface curvature threshold below / above which a point is considered a flat / corner point
    - the voxel size used in filtering less flat surface points
- Speichern von registrierten Pointclouds:
  rosrun pcl_ros pointcloud_to_pcd input: = /velodyne_cloud_registered

4.3.2 Laser Odometry and Mapping (Loam)

Referenz: Basiert auf [9]

Quelle: https://github.com/daobilige-su/loam_velodyne

Installation/Anwendung: Siehe Kapitel 4.3.1

Hinweise:

- Die Änderungen in dieser Version des LOAM SLAMs beziehen sich lediglich auf die Verarbeitung eines bestimmten Datensatzes

4.3.3 3D Graph SLAM with NDT

Referenz: Basiert auf [10]

Quelle: https://github.com/koide3/hdl_graph_slam

Installation:

Voraussetzung:

Bibliotheken:

- OpenMP
- PCL 1.7
- g2o

Ros-Pakete:

- geodesy
- nmea_msgs
- pcl_ros
- ndt_omp

Vorgehen:

1. **g2o**: Die Bibliothek muss von diesem Commit (*„a48ff8c42136f18fbe215b02bfeca48fa0c67507“)* installiert werden. Die anderen Versionen funktionieren laut Beschreibung im Git Repository nicht:
   git clone https://github.com/RainerKuemmerle/g2o.git
   cd g2o
   git checkout a48ff8c42136f18fbe215b02bfeca48fa0c67507
   mkdir build && cd build
   cmake .. -DCMAKE_BUILD_TYPE=RELEASE
   make -j8
   sudo make install
2. **PCL:** Die Installation von PCL funktioniert nicht über den Standardweg. Die Bibliothek muss für Ubuntu 16.04 über die Vorgehensweise, beschrieben in https://stackoverflow.com/questions/39874875/error-in-install-point-cloud-library-ubuntu16-04, installiert werden

3. **Ros Pakete:**

   sudo apt-get install ros-indigo-geodesy ros-indigo-pcl_ros ros-indigo-nmea-msgs

   (Rechtschreibfehler: `ros-indigo-pcl-ros`)

   cd ~/catkin_ws/src

   git clone https://github.com/koide3/ndt_omp.git

4. **Installation der Bibliothek:**

   cd ~/catkin_ws/src

   git clone https://github.com/koide3/hdl_graph_slam.git

   catkin_make

Die Installation der Bibliothek (Ausführung von *catkin_make*) funktioniert auf dem genutzten System nicht.

Hinweise:

- Die Installation funktioniert angeblich mit Ubuntu 14.04 und ROS Jade (siehe https://github.com/koide3/hdl_graph_slam/issues/4)
- Des Weiteren existiert eine Docker Version, die ebenfalls funktionieren soll
- Das Referenzpaper ist nicht öffentlich, kann aber per Email beim Autor angefragt werden

4.3.4 Cartographer

Referenz: Basiert auf [11]

Quelle: https://github.com/googlecartographer/cartographer

Installation:

Es existiert keine offizielle Installationsanweisung für die Nutzung des Cartographer SLAMs mit den Daten des Velodyne VLP-16. Allerdings beschreiben einige Nutzer die Anwendung des SLAMs mit Daten des Velodyne. Des Weiteren existiert eine ROS Implementierung des Cartographer SLAMs (https://github.com/googlecartographer/cartographer_ros), welche in einer bestimmten Konfiguration, 2D Karten aus 3D Velodyne Daten erzeugt. Eine Installation dieser Bibliothek war im Rahmen dieser Ausarbeitung nicht möglich. Weitere Informationen für die Installation sind aber unter den folgenden Quellen zu finden:

https://github.com/googlecartographer/cartographer_ros/issues/534

https://github.com/googlecartographer/cartographer_ros/issues/259

https://github.com/googlecartographer/cartographer_ros/issues/425

4.3.5 Berkeley Localization And Mapping (BLAM)

Referenz: Keine

Quelle: https://github.com/erik-nelson/blam

Installation:

Voraussetzungen:

Bibliotheken:

- GTSAM
- Boost

Vorgehen:

1. Installation der benötigten Bibliotheken:
    a. Installation von Boost:
        i. sudo apt-get install libboost-all-dev
    b. Installation von GTSAM:
        i. Es muss die neuste Version von GTSAM installiert werden (4.X). Diese befindet sich im Developer Branch des Repositorys. Sämtliche andere Versionen von GTSAM funktionieren nicht! Weiterhin müssen die Befehle für die Installation von GTSAM mit Sudo Rechten durchgeführt werden. Ansonsten funktioniert später die Erstellung der gesamten Bibliothek mit cmake nicht!
        ii. git clone https://bitbucket.org/gtborg/gtsam.git
        iii. cd gtsam
        iv. git checkout develop
        v. **sudo** mkdir build
        vi. cd build
        vii. **sudo** cmake ..
        viii. **sudo** make check
        ix. **sudo** make install
2. Herunterladen und Erstellung des BLAM Repositorys:
    a. cd ~/catkin_ws/src
    b. git clone https://github.com/erik-nelson/blam.git
    c. cd blam
    d. ./update

3. In der ROS Version Kinetic kommt es bei der Anwendung der Bibliothek zu dem Fehler *„ros/ros.h nicht gefunden". Beheben durch*[3]*:*
    a. *cd blam/internal/src/geometry_utils*
    b. Einfügen der beiden Zeilen in die package.xml:
        i. *<build_depend>roscpp</build_depend>*
        ii. *<run_depend>roscpp</run_depend>*
    c. Änderung der CMakeList.txt:
        i. *find_package(catkin REQUIRED COMPONENTS roscpp)*
        ii. *include_directories(include ${catkin_INCLUDE_DIRS})*

Dauer: 1 Stunde

Anwendung:

1. **Starten des SLAMS:**
    a. **Online-Modus (Abspielen der Bag Files oder Live Daten):** *roslaunch blam_example test_online.launch*
    b. **Offline-Modus (Definition eines Bag Files, das so schnell wie möglich verarbeitet wird):** *roslaunch blam_example test_offline.launch*
2. **Abspielen des ROS Bag files:** *rosbag play <bagfile.bag>*
3. **Visualisierung in rviz starten:** *blam_example/rviz/lidar_slam.rviz*

Hinweise:

- Eingabelidardaten müssen unter dem ROS Topic */velodyne_points* gepublished werden
- Anpassungen des Codes nötig: https://github.com/laboshinl/loam_velodyne/issues/41
- Darstellung unter Rviz funktioniert noch nicht richtig. Anstatt dessen können die generierten Karten auch direkt in .pcd Dateien gespeichert werden:
    - rosrun pcl_ros pointcloud_to_pcd input:=/blam/blam_slam/octree_map_updates (Speichert die schrittweisen Updates der Karte)
    - rosrun pcl_ros pointcloud_to_pcd input:=/blam/blam_slam/octree_map (Speichert die gesamte Karte)
    - Parametrisierung (https://github.com/erik-nelson/blam/issues/19):
      Laut Github bestimmen die Parameter *pcl::GeneralizedIterativeClosestPoint, pcl::IterativeClosestPoint, pcl::IterativeClosestPointWithNormals* wie detailliert die Karte dargestellt wird und damit auch die Geschwindigkeit

[3] https://github.com/erik-nelson/blam/issues/8

5. Evaluierung

Bei der Installation der Bibliotheken wurde festgestellt, dass die SLAMs in Kapitel 4.3.1 und 4.3.2 beide auf dem LOAM Paper basieren und der Letztere lediglich kleine Unterschiede zum Originalpaper aufweist. Dementsprechend wird im Folgenden nur noch die Originalimplementierung aus Kapitel 4.3.1. verwendet. Der SLAM, beschrieben in Kapitel 4.3.3, konnte auf dem verwendeten Computer nicht installiert werden. Die Installation auf Ubuntu 16.04 in Kombination mit ROS Kinetic ist nicht möglich. Eine Installation auf Ubuntu 14.04 in Kombination mit ROS Jade ist laut Github Issues möglich. Es ist eine Docker Version vorhanden, die aber auf Grund von mangelnden Kenntnissen der Dockersoftware ebenfalls nicht verwendet werden konnte. Für den Cartographer SLAM (Kapitel 4.3.4) existiert ebenfalls keine vollständige Anleitung zur Nutzung mit den Daten des Velodyne VLP-16. Dies ist aber laut Github Issues prinzipiell möglich. Der BLAM SLAM konnte durch einige Veränderungen der Installationsanleitung auf dem verwendeten Computer genutzt werden. Im Rahmen dieser Ausarbeitung wurde der LOAM sowie der BLAM SLAM erfolgreich installiert. Der größte Unterschied zwischen den beiden SLAM Implementierungen besteht darin, dass der LOAM SLAM, im Gegensatz zum BLAM SLAM, kein Loop Closing unterstützt. Die Anwendung der SLAMs erfolgt auf den aufgenommenen Bagfile Daten. Die generierten Karten wurden mit Hilfe der ROS Bibliothek pcl_ros[4] im .pcd Format gespeichert und mit der Software CloudCompare[5] dargestellt.

Für die Evaluierung der SLAM Bibliotheken werden die Kriterien aus Kapitel 3.5 verwendet. Das erste Testkriterium, der Umgang mit beweglichen Objekten, kann anhand der erstellten Karten betrachtet und verglichen werden. Die Einflüsse unterschiedlicher Wetterbedingungen kann mit den vorhandenen Datensätzen nicht untersucht werden. Für die Überprüfung der Echtzeitfähigkeit kann die Abspielgeschwindigkeit der Bagfiles variiert und die Qualität der generierten Karten miteinander verglichen werden. Eine Variation der Sensorhöhe wird nicht durchgeführt. Es erfolgt lediglich eine Betrachtung, welche Entfernungen in der aktuellen Konfiguration gemessen werden können und wie sich eine Veränderung der Sensorhöhe auf diese Entfernungen auswirken würde.

[4] http://wiki.ros.org/pcl_ros
[5] https://www.danielgm.net/cc/

5.1 Bewegliche Objekte

Zur Überprüfung der SLAM Implementierungen in dynamischen Umgebungen wurden Karten aus Aufnahmen erzeugt, die bewegliche Objekte enthalten. Die resultierenden Karten werden im Folgenden visuell miteinander verglichen.

Die Abbildung 5 zeigt die Ergebnisse des LOAM SLAMs und die Abbildung 6 die Ergebnisse des BLAM SLAMs von Aufnahmen vor dem Mensavorplatz der Otto-von-Guericke Universität. In Abbildung 5 ist zu erkennen, dass der LOAM SLAM bewegliche Objekte in die erzeugte Karte integriert. In diesem Fall sind Personen als Artefakte in der erzeugten Karte zu erkennen. Der BLAM SLAM hingegen erzeugt bei beweglichen Objekten Rauschen in den erstellten Karten (siehe Abbildung 6).

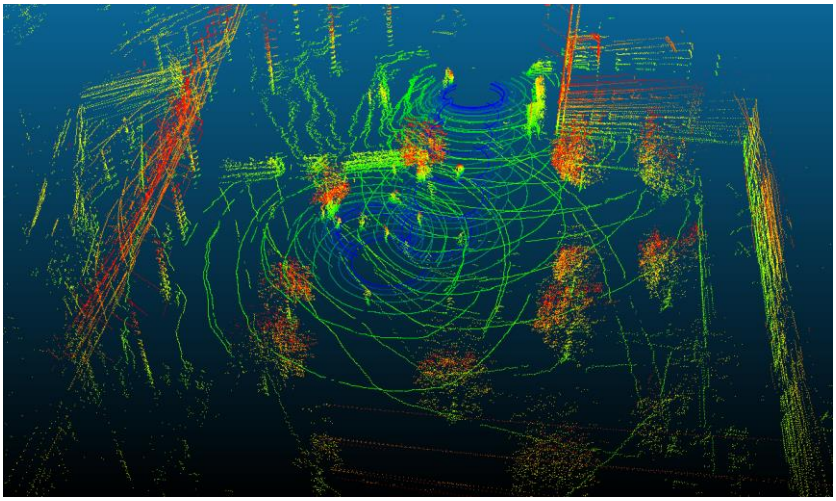

Abbildung 5 LOAM SLAM Universitätsvorplatz

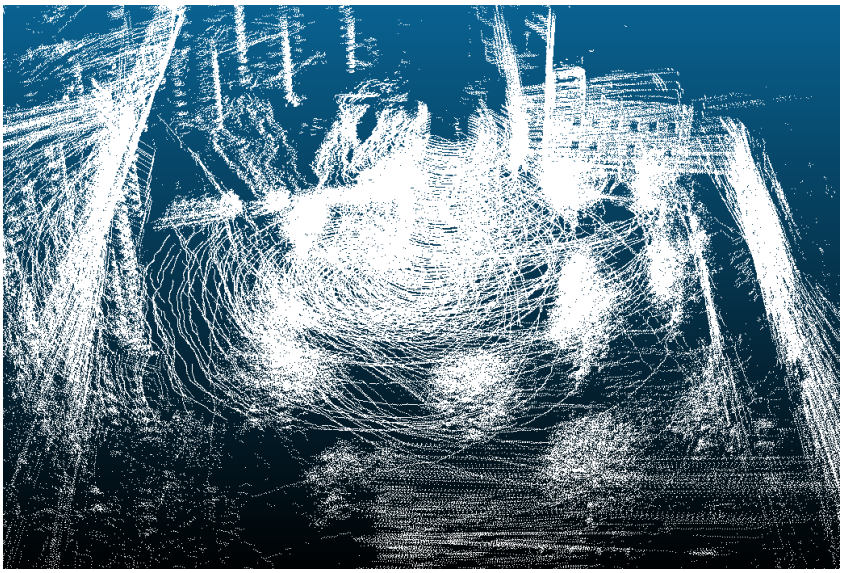

Abbildung 6 BLAM SLAM Universitätsvorplatz

5.2 Sensorhöhe

Die Höhe der Anbringung des Velodyne Sensors beeinflusst, welche Objekte in der Umgebung erfasst werden. In Abbildung 7 und Abbildung 8 ist zu erkennen, dass der Sensor so wohl sehr große Objekte als auch sehr kleine Objekte ausreichend gut erfasst. Des Weiteren lässt sich rechnerisch feststellen, welche Objekte im Umfeld des Fahrrads erkannt werden. Die Berechnung erfolgt unter Nutzung des Sinussatzes. In einem Meter Entfernung erkennt der Sensor Objekte in einer Höhe von 0.53 Metern bis 1,83 Metern. In zwei Meter Entfernung erkennt der Sensor Objekte in einer Höhe von 0.26 Metern bis 2,87 Metern. Die maximale Ausstrahlung der Laserstrahlung beträgt 100 Meter. Dabei werden Bereiche vom Boden bis zu einer Höhe von 27,59 Metern erfasst.

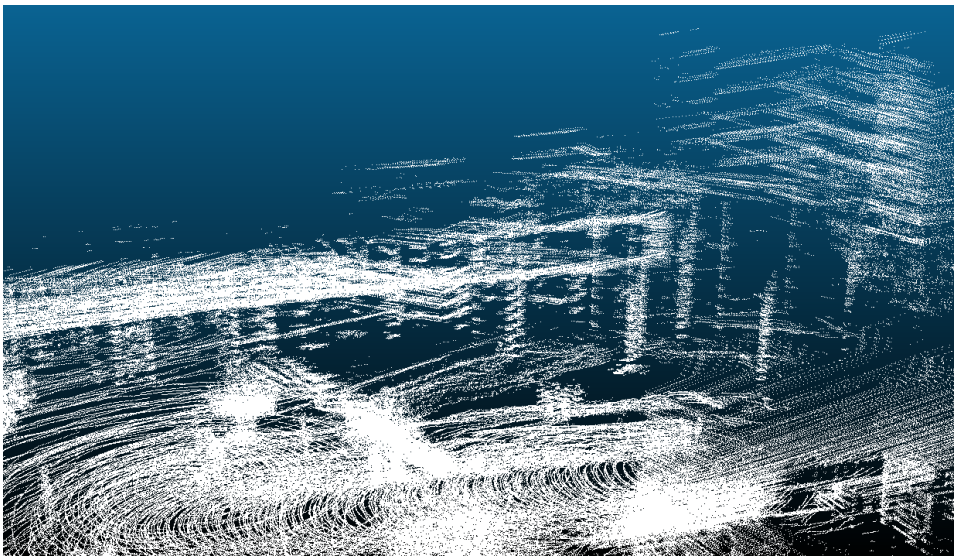

Abbildung 7 Erfassung hoher Objekte, wie beispielsweise der Universitätsbibliothek

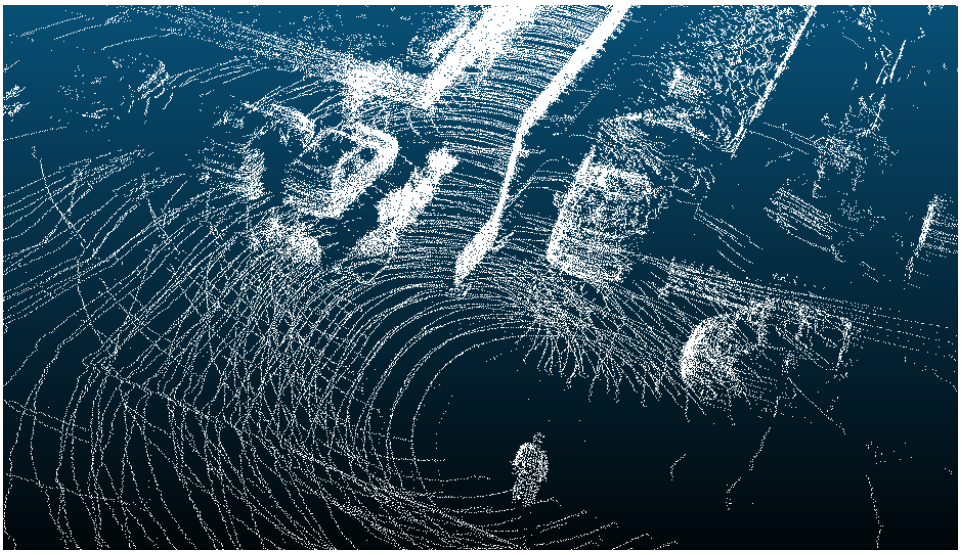

Abbildung 8 Erfassung flacher Objekte, wie Autos oder Personen

5.3 Echtzeitfähigkeit

Zur Untersuchung der Echtzeitfähigkeit der Algorithmen wurden die Daten in den Bagfiles mit unterschiedlichen Geschwindigkeiten simuliert abgespielt, mit den SLAMs verarbeitet und die Ergebnisse im Anschluss visuell miteinander verglichen. Zum einen wurden die Daten in Echtzeit verarbeitet und zum anderen wurden Karten erzeugt, bei denen die Daten mit halber Geschwindigkeit abgespielt wurden. Die Abbildung 9 und 11 zeigen einen Ausschnitt der in Echtzeit erzeugten Karten während Abbildung 10 und 12 die, mit halber Geschwindigkeit erzeugten, Karten zeigt.

Die Ergebnisse innerhalb eines SLAMs unterscheiden sich bei der Variation der Wiedergabegeschwindigkeit kaum. Die Anzahl der Punkte in den Punktewolken, dargestellt in Abbildung 9 und 10, unterscheidet sich beispielsweise lediglich den Faktor 0.001. Dasselbe gilt für den LOAM SLAM.

Der LOAM sowie der BLAM SLAM sind somit auf dem genutzten System echtzeitfähig. Allerdings unterscheiden sich die Größen der generierten Punktewolken der SLAMs sehr stark. Die Karten des BLAM SLAMs beinhalten das Zehnfache der Punkte des LOAM SLAMs.

Weiterhin ist bei der Untersuchung zu beachten, dass die SLAMs mit größer werden Karten langsamer werden. In diesen Experimenten ging es um die Erstellung einer Karte. Für die Navigation kann es auch beispielsweise schon ausreichen, lediglich das lokale Umfeld zu beachten und alte Teile der Karte zu verwerfen.

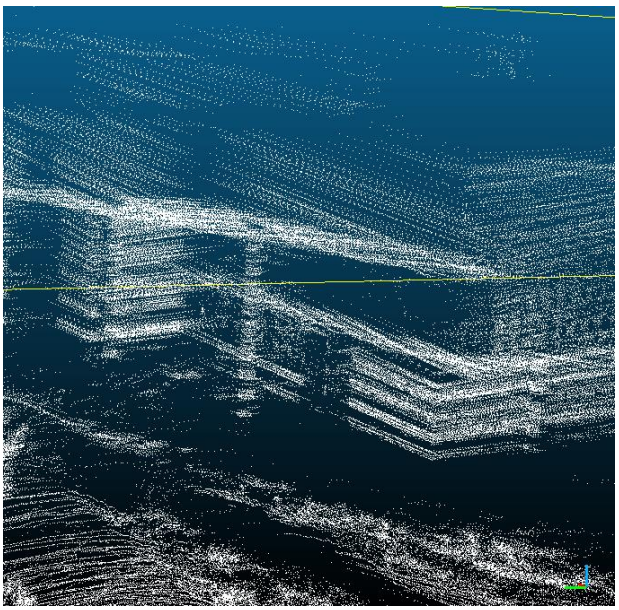

Abbildung 10 BLAM: Datenverarbeitung bei halber Geschwindigkeit

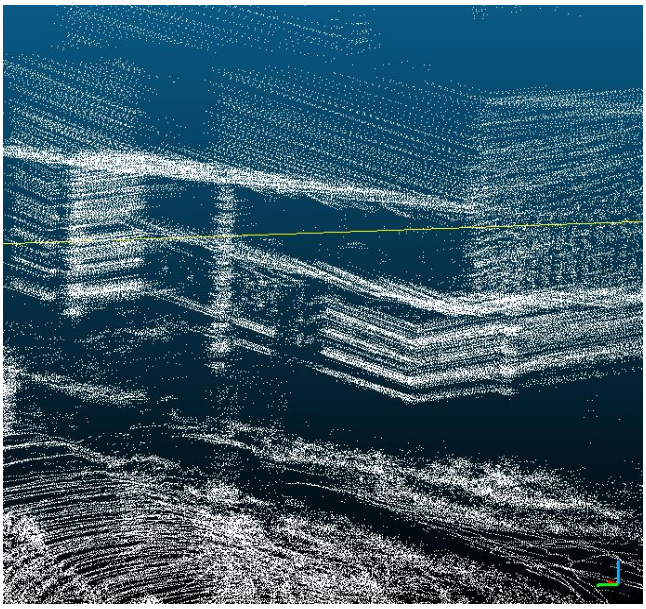

Abbildung 9 BLAM: Datenverarbeitung in Originalgeschwindigkeit

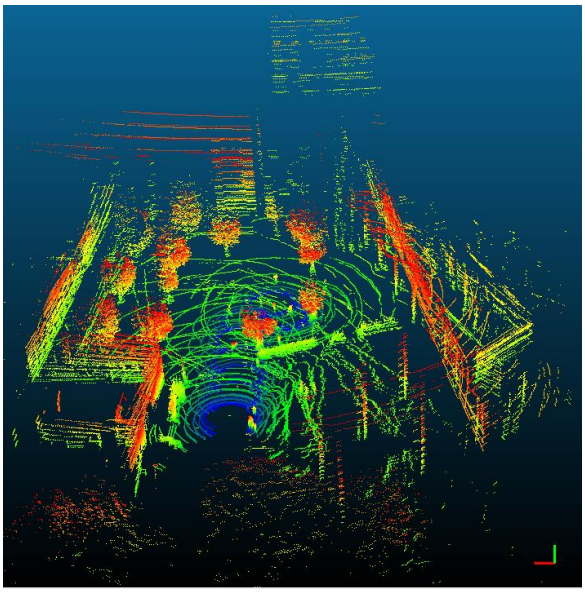

Abbildung 12 LOAM: Datenverarbeitung bei halber Geschwindigkeit

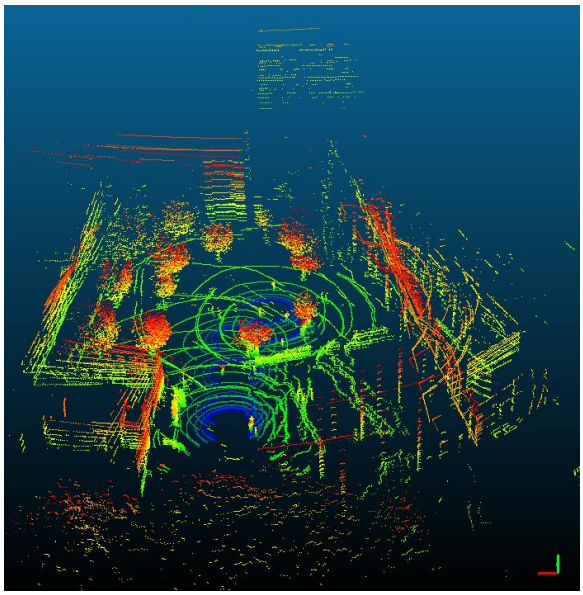

Abbildung 11 LOAM: Datenverarbeitung bei Originalgeschwindigkeit

5.4 Fazit

Sowohl der LOAM als auch der BLAM SLAM liefern gute Ergebnisse bei der Erstellung der Karten. Der BLAM SLAM erzeugt im Gegensatz zum LOAM SLAM größerer Punktewolken und liefert somit detailliertere Ergebnisse. Allerdings verarbeitet der BLAM SLAM bewegliche Objekte nicht so gut wie der LOAM SLAM und erzeugt ein Rauschen in der Karte. Der LOAM SLAM hingegen integriert die beweglichen Objekte in die erzeugte Karte. Abhängig vom Anwendungsfall kann so eine Auswahl, des zu nutzenden Algorithmus, getroffen werden. Womöglich kann das Rauschen nachträglich herausgefiltert werden. Ein Nachteil des LOAM SLAMs ist ein Drift der Punktewolke in Richtung der Z-Achse (siehe Abbildung 13). Dies trat beim BLAM SLAM nicht auf.

Die Sensoranbringung in einer Höhe von 80 Zentimetern ist ebenfalls ausreichend, um sämtliche Hindernisse in der näheren Umgebung zu erfassen

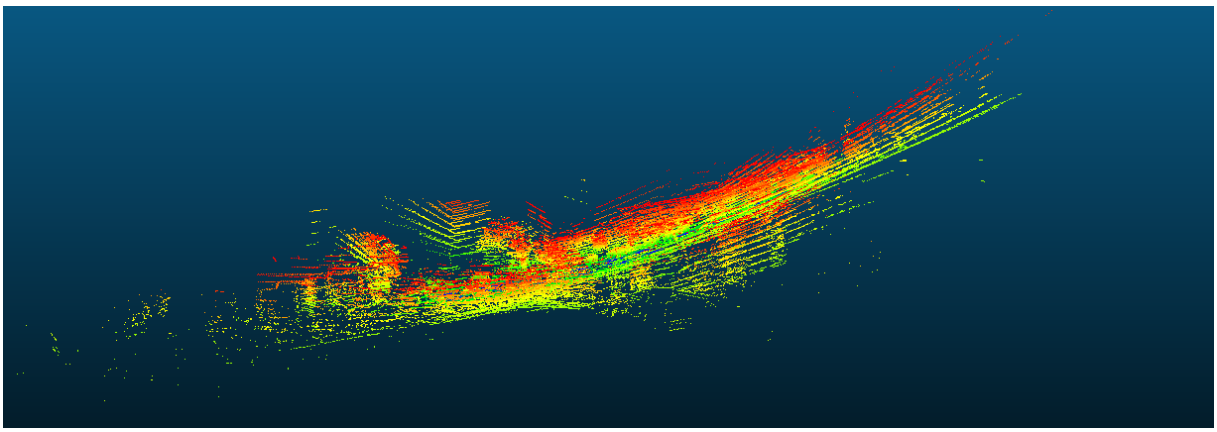

Abbildung 13 Datenabweichung in Z-Richtung

6. Ausblick

Neben der Erzeugung von Kartendaten zur Navigation, ist für das „Transformer“ Projekt der Universität Magdeburg, vor allem auch die Erkennung von Objekten in den Lidardaten von Interesse. In hoch dynamischen Umgebungen wie dem Universitätsgelände oder im Allgemeinen auch dem öffentlichen Straßenverkehr, müssen Objekte in Echtzeit erkannt werden, um eine kollisionsfreie Navigation zu ermöglichen. Bei einer weiteren Github Recherche wurde eine kostenfreie Bibliothek identifiziert, die eine Objekterkennung in Pointcloud Daten von Velodyne Sensoren erlaubt. Die Bibliothek wurde ebenfalls auf dem, in Abschnitt 4.1 beschriebenen, Computer installiert und mit den aufgenommenen Bagfiles getestet.

Referenz: [25], [26]

Lizenz: GNU General Public License

Quelle: https://github.com/PRBonn/depth_clustering

Installation:

Voraussetzungen:

- OpenCV
- QGLViewer
- FreeGLUT
- QT4/QT5

Vorgehen:

1. Installation der vorausgesetzten Bibliotheken:
    - *sudo apt install libopencv-dev libqglviewer-dev freeglut3-dev qtdeclarative5-dev*

   Achtung: **libqt5-dev,** wie in der Anleitung beschrieben, funktioniert nicht. Anstatt dessen muss qtdeclarative5-dev installiert werden (in obigem Installationsbefehl bereits geändert)
2. Erstellung des Projektes:
    - *mkdir build*
    - *cd build*
    - *cmake ..*
    - *make -j4*
    - *ctest -VV*

Dauer: < 0.5 Stunden

Anwendung:

1. Navigation zu den Binaries: cd <path_to_project>/build/devel/lib/depth_clustering
2. Ausführung der Bibliothek:
    a. **Mit Bilddaten (.png):** *./show_objects_moosmann --path data/scenario1/*
    (Die Daten müssen im Vorfeld mit dem Befehl *mkdir data/; wget http://www.mrt.kit.edu/z/publ/download/velodyneslam/data/scenario1.zip -O data/moosmann.zip; unzip data/moosmann.zip -d data/; rm data/moosmann.zip* heruntergeladen werden)
    b. **Mit Velodyne Daten:** *./show_object_node –num_beams 16*
    Parameter:
    --num_beams 16: für Velodyne VLP-16. Andere Velodyne Sensoren werden ebenfalls unterstützt
    --angle <int>: Volumen der Bounding Boxes, die erkannt werden sollen
3. Abspielen des Bag Files (Daten müssen unter dem Topic */velodyne_points* gepublished werden)

Hinweis:

- Die Bounding Boxes der erkannten Objekte werden nur dargestellt aber nicht unter einem ROS Topic gepublished
- Diese Software ist lediglich für die Darstellung der erkannten Objekte nützlich

Variante:

Für eine weitere Verwendung der erkannten Objekte, sollte der Repository Fork **https://github.com/bgheneti/depth_clustering** verwendet werden. In diesem Fork erfolgt das Publishen der Bounding Boxes der erkannten Objekte als ROS Topic. Diese Informationen können beispielweise von einem weiteren ROS Node für eine Kollisionserkennung genutzt werden.

Ergebnisse:

Das Clustering der Punkte der Lidardaten erfolgt in Echtzeit und erfasst sämtliche Hindernisse und bewegliche Objekte in der näheren Umgebung (siehe Abbildung 13). Durch die detaillierte Erfassung des Umfeldes, können mit Hilfe des Laserscanners gute Ergebnisse für die Objekterkennung erreicht werden. Bei der Ausführung der Bibliothek kann eingestellt werden, welche Objektgrößen erkannt werden sollen und mit einer Bounding Box markiert werden (siehe blaue Bounding Boxes in Abbildung 14). In der Standardeinstellung werden Objekte mit einem Volumen der Bounding Box kleiner als 10 Kubikmeter markiert.

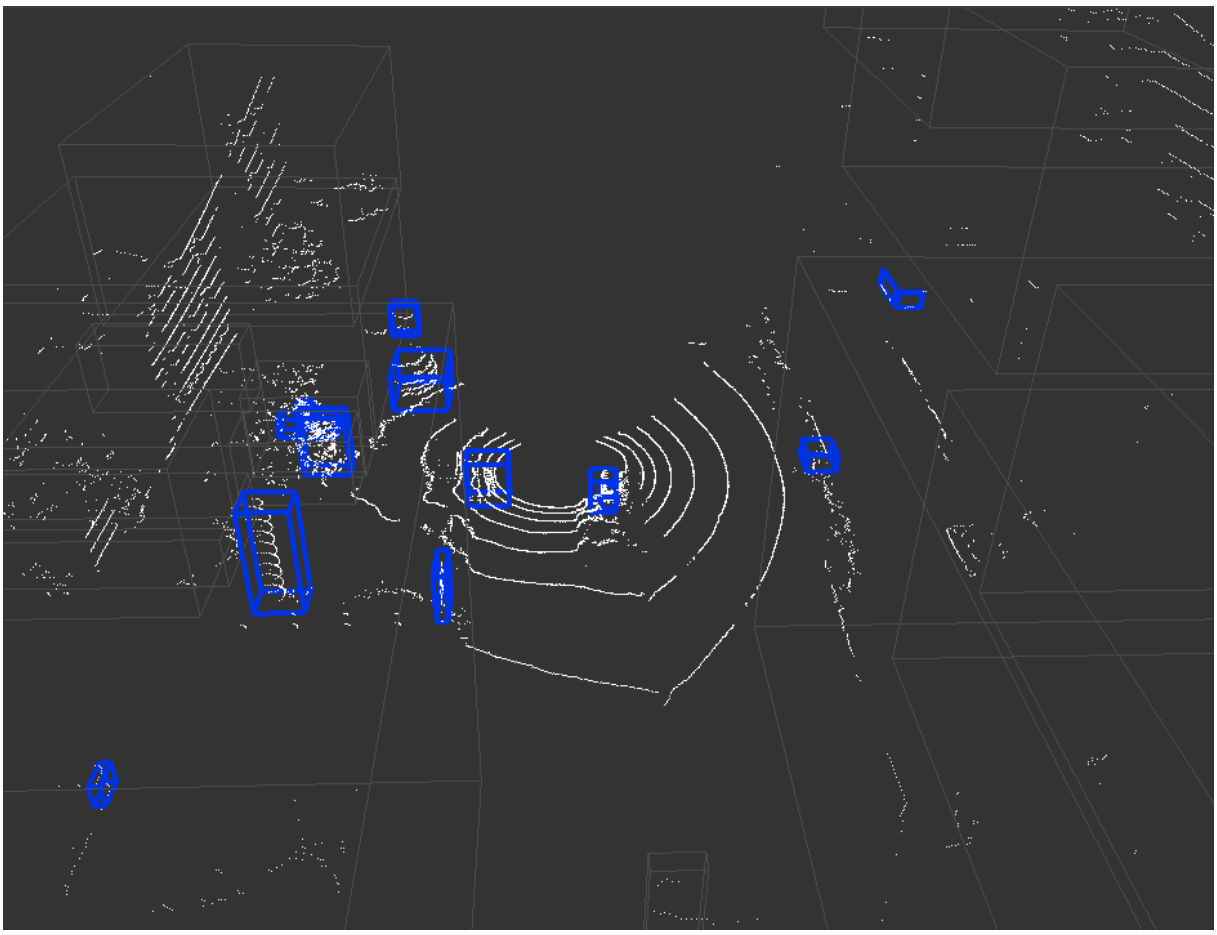

Abbildung 14 Objekterkennung in Velodyne VLP-16 Lidardaten

7. Fazit

Das Ziel dieser Arbeit war die Evaluierung eines Velodyne VLP-16 als Referenzsystem zur Kartenerstellung in Outdoorszenarien. Im Rahmen der Arbeit erfolgte eine Einrichtung des Velodyne VLP-16 Lidarsensors, eine Datenaufnahme auf dem Mensavorplatz der Otto-von-Guericke Universität, eine Recherche und Installation von LIDAR SLAM Bibliotheken, sowie die Untersuchung unterschiedlicher Außeneinflüsse auf die Ergebnisse der installierten SLAM Bibliotheken, mit den aufgenommenen Testdaten.

Die Installation des Velodyne VLP-16 Treibers und eine Einrichtung des Velodyne Sensors erfolgte auf einem Universitätslaptop. Die Aufnahme der Daten erfolgte mit Hilfe des Transformers Projektes der Otto-von-Guericke Universität auf dem Mensavorplatz der Universität. Sämtliche Daten wurden in ROS Bagfiles gespeichert. Eine Auswertung der Laserdaten erfolgt mit Hilfe von verschiedenen SLAM Bibliotheken. Die Identifizierung, der zu nutzenden Bibliotheken, erfolgte auf Basis einer stichwort-basierten Github Recherche. Weiterhin wurde die Installation und Konfiguration der installierten Bibliotheken detailliert beschrieben. Bei der Recherche nach Literatur und Implementierungen für LIDAR SLAM Anwendungen stellte sich heraus, dass wenig kostenfreie Implementierungen existieren (siehe Kapitel 3.2). Im Allgemeinen existiert keine Bibliothek, die speziell für den Outdooreinsatz entwickelt wurde. Von allen recherchierten Bibliotheken, wurden fünf Bibliotheken identifiziert, die womöglich gute Ergebnisse liefern könnten, von denen allerdings nur zwei Bibliotheken auf dem genutzten System installiert werden konnten. Dabei handelt es sich um den LOAM sowie den BLAM SLAM. Die Installation der Bibliotheken war sehr aufwändig und die Installationsanleitungen fehlerhaft und unvollständig.

Bei der Anwendung in Outdoorszenarien muss der SLAM Algorithmus mit unterschiedlichen Außeneinflüssen umgehen können. Das wichtigste dabei ist vor allem der Umgang mit beweglichen Objekten. Der LOAM SLAM integriert bewegliche Objekte als Artefakte in die Karte, während der BLAM SLAM verrauschte Karten, in Bereichen mit vielen dynamischen Objekten erzeugt.

Generell müssen auch die Auswirkungen unterschiedlicher Wetterbedingungen auf die Sensorik beachtet werden. Grundsätzlich können schlechte Wetterbedingungen, wie starker Regen, Nebel oder Schneefall aber auch starke Sonneneinstrahlung die Qualität der aufgenommenen Daten stark beeinträchtigen. Problematisch sind vor allem Reflektionen der Laserstrahlen bei der weiteren Verwendung der Daten. Diese simulieren Objekte, die in der realen Situation nicht vorhanden sind. Die Auswirkungen von Regen, Schnee und Nebel können allerdings über Wahrscheinlichkeitsverteilungen in den Datenaufnahmeprozess integriert werden.

Des Weiteren hat auch die Höhe der Anbringung des Sensors einen Einfluss auf die Erkennung von Objekten. Unter Nutzung des Sinussatzes wurde festgestellt, dass bei einer Anbringungshöhe von 80 Zentimetern, in einem Meter Entfernung, Objekte mit einer Höhe von 0.53 Metern bis 1,83 Metern erkannt werden. Bei zwei Metern Entfernung beträgt die Erkennungshöhe 0.26 bis 2,87 Meter. Dies ist für eine Kollisionserkennung ausreichend.

Insgesamt können mit Hilfe des Velodyne VLP-16 in Kombination mit dem LOAM sowie BLAM SLAM aussagekräftige Karten, auch im Outdooreinsatz, erzeugt werden. In folgenden Untersuchungen können die, in Kapitel 4.3 beschriebenen, Parameter der unterschiedlichen SLAMs untersucht und ihr Einfluss auf die generierte Karte betrachtet werden. Weiterhin kann die in Kapitel 6 beschriebene Bibliothek für eine Kollisionserkennung mit Hilfe von Lidardaten evaluiert werden. Allerdings ist der Preis für einen Velodyne VLP-16 für eine serienmäßige Nutzung zu hoch. Eine mögliche Alternative könnte die einmalige Erstellung von Basiskarten des Gebiets der Otto-von-Guericke Universität, mit Hilfe des Velodyne, sein. Die Navigation und Kollisionserkennung kann im Anschluss mit kostengünstigeren Kameras durchgeführt werden.

Literaturverzeichnis

[1] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: part I,” *IEEE Robot. Autom. Mag.*, vol. 13, no. 2, pp. 99–110, Jun. 2006.

[2] T. Bailey and H. Durrant-Whyte, “Simultaneous localization and mapping (SLAM): part II,” *IEEE Robot. Autom. Mag.*, vol. 13, no. 3, pp. 108–117, Sep. 2006.

[3] C. Cadena *et al.*, “Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age,” *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1309–1332, 2016.

[4] R. H. Rasshofer, M. Spies, and H. Spies, “Influences of weather phenomena on automotive laser radar systems,” *Adv. Radio Sci*, vol. 9, pp. 49–60, 2011.

[5] W. Sun *et al.*, “Technique to separate lidar signal and sunlight References and links,” 2016.

[6] S. Michaud, J.-F. Lalonde, and P. Gigù Ere, “Towards Characterizing the Behavior of LiDARs in Snowy Conditions.”

[7] D. Moratuwage, B.-N. Vo, and D. Wang, “Collaborative Multi-Vehicle SLAM with Moving Object Tracking.”

[8] C.-C. Wang, C. Thorpe, M. Hebert, S. Thrun, and H. Durrant-Whyte, “Simultaneous Localization, Mapping and Moving Object Tracking.”

[9] J. Zhang and S. Singh, “LOAM: Lidar Odometry and Mapping in Real-time.”

[10] and E. M. Kenji Koide, Jun Miura, “A Portable 3D LIDAR-based System for Long-term and Wide-area People Behavior Measurement.”

[11] W. Hess, D. Kohler, H. Rapp, and D. Andor, “Real-time loop closure in 2D LIDAR SLAM,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2016.

[12] A. Harmat, M. Trentini, and I. Sharf, “Multi-Camera Tracking and Mapping for Unmanned Aerial Vehicles in Unstructured Environments,” *J. Intell. Robot. Syst.*, vol. 78, no. 2, pp. 291–317, May 2015.

[13] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, “3-D Mapping with an RGB-D camera,” *IEEE Trans. Robot.*, 2014.

[14] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, “A flexible and scalable SLAM system with full 3D motion estimation,” in *9th IEEE International Symposium on Safety, Security, and*

Rescue Robotics, SSRR 2011, 2011.

[15] G. Grisetti and C. Stachniss, “Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling.”

[16] S. Leutenegger, “Unmanned Solar Airplanes Design and Algorithms for Efficient and Robust Autonomous Operation.”

[17] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, “Keyframe-Based Visual-Inertial Odometry Using Nonlinear Optimization.”

[18] S. Leutenegger, P. Furgale, V. Rabaud, M. Chli, K. Konolige, and R. Siegwart, “Robotics: Science and Systems ssss Keyframe-Based Visual-Inertial SLAM Using Nonlinear Optimization.”

[19] R. Mur-Artal and J. D. Tardós, “ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras,” 2017.

[20] J. Engel, T. Schöps, and D. Cremers, “LSD-SLAM: Large-Scale Direct monocular SLAM,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014.

[21] J. Engel, J. Sturm, and D. Cremers, “Semi-dense visual odometry for a monocular camera,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2013.

[22] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, “ORB-SLAM: a Versatile and Accurate Monocular SLAM System.”

[23] V. L. Inc., “www.velodynelidar.com,” *VLP-16 Data sheet*. [Online]. Available: http://velodynelidar.com/vlp-16.html.

[24] ROS, “Getting Started with the Velodyne VLP16.” [Online]. Available: http://wiki.ros.org/velodyne/Tutorials/Getting Started with the Velodyne VLP16.

[25] I. Bogoslavskyi and C. Stachniss, “Fast Range Image-Based Segmentation of Sparse 3D Laser Scans for Online Operation.”

[26] I. Bogoslavskyi and C. Stachniss, “Efficient Online Segmentation for Sparse 3D Laser Scans.”

Abbildungsverzeichnis

Tabellenverzeichnis