BACHELOR THESIS
Hauke Willi Kosmiter

Entwicklung und Implementierung eines dynamischen Karten-Systems für die Erkennung und Kommunikation von Parklücken für das automatisierte und vernetzte Fahren

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Hauke Willi Kosmiter

Entwicklung und Implementierung eines dynamischen Karten-Systems für die Erkennung und Kommunikation von Parklücken für das automatisierte und vernetzte Fahren

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Rasmus Rettig
Zweitgutachter: Prof. Dr. Karl-Ragmar Riemschneider

Eingereicht am: 23.05.2022

Hauke Willi Kosmiter

Thema der Arbeit

Entwicklung und Implementierung eines dynamischen Karten-Systems für die Erkennung und Kommunikation von Parklücken für das automatisierte und vernetzte Fahren

Stichworte

ROS, Autonomes Fahren, Local Dynamic Map, Parkplätze, LiDAR

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Entwicklung und Implementierung eines Systems, welches für die Auswertung des Status eines Parkplatzes genutzt werden kann. Dieses System stellt die ausgewerteten Daten der Parkplätze auf einer dynamischen Karte zum Abrufen bereit.

Hauke Willi Kosmiter

Title of Thesis

Development and implmentation of a dynamic map system for the detection and communication of parking spaces for automated and connected driving

Keywords

ROS, Autonomous driving, Local Dynamic Map, Parking lots, LiDAR

Abstract

This thesis is about development and implementaion of a system which evaluate the status of a parking lot. This system provide the data from the evaluation to a dynamic map.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Die Anzahl der Personenkraftfahrzeugen ist innerhalb von 10 Jahren von 42,302 Millionen im Jahre 2011 auf 48,249 Millionen im Jahre 2021 gestiegen [18]. Diese zunehmende Anzahl und im Zuge dessen der zunehmende Bedarf an Stellplätzen für die Fahrzeuge sorgen vor allem in Städten für oftmals langwierigen Suchen nach einem geeigneten Parkplatz. Laut dem ehemaligen Verkehrsminister Alexander Dobrindt in einem, dem Straubinger Tagblatt am 09.09.2017 gegebenen, Interview machen „40 Prozent des Verkehrs in Städten" die Suche nach Parkplätzen aus [16]. In Zeiten, in denen auf den Straßen viel Verkehr stattfindet, bedeutet die Suche nach einem geeigneten Parkplatz also viel zusätzlichen Verkehr, der sich vermeiden ließe. Dafür ist es hilfreich, ein System mit Kenntnisse über momentan freie Parkplätze, nutzen zu können. Ein solches System benötigt eine dynamische Karte der Umgebung, in der die freien Parkplätze eingetragen und abgerufen werden können. Dazu sind die Maße und die Ausrichtung des Parkplatzes sowie die Uhrzeit, bei derer der Status des Parkplatzes in die Karte eingetragen worden ist, interessant für den Endbenutzer dieser Karte. Dadurch wird es diesem ermöglicht eine bessere Entscheidung zur Parkplatzfindung zu treffen. Um freie Parkplätze in die Karte eintragen zu können und um diese Karte dynamisch zu halten, wird Sensorik benötigt, die durch entsprechende Programme die Parkplätze auswerten können. Zudem wird eine Datenbank benötigt, welche die Daten in der, für die Anwendungen, richtigen Struktur speichert. Diese Datenbank muss über eine Schnittstelle für das Fahrzeug, welches die Parkplätze auswertet und den Endbenutzer, welche die Parkplätze auf einer Karte angezeigt bekommt, erreichbar sein.

Dieses System bietet zudem die Möglichkeit, flexibel die Parkplatz Situationen in den Städten erfassen zu können, um dort einen Überblick zu bekommen, an welchen Orten oft Parkplätze belegt sind, also dort eine hohe Nachfrage stattfindet, an welchen Orten kaum Parkplätze belegt sind, also dort keine hohe Nachfrage stattfindet und zudem wie hoch die Fluktuation der belegten Parkplätze an bestimmten Orten ist. Diese Daten könnten hilfreich in die Verkehrsplanung einer Stadt einfließen.

Ein solches System kann die Zeit und den Verkehr, 40% des Stadtverkehrs, senken, weil der Endbenutzer durch das System schneller einen, für das eigene Fahrzeug, passenden Parkplatz finden kann, sofern ein freier Parkplatz am gewünschten Standort vorhanden ist. Zudem hat es variantenreiche Monetarisierungsmöglichkeiten. Der Hersteller des Fahrzeuges hat die Möglichkeit das System für einen Aufpreis mit zum Kauf des Fahrzeuges anzubieten oder kann sich für ein Abo-Modell entscheiden. Das Abo-Modell ließe sich auch Unternehmen anbieten, die keine eigenen Parkplätze für Mitarbeiter oder Kunden besitzen. In solchen Fällen könnte es nützlich sein, in einem bestimmten Radius um den Standort des Unternehmens herum die verfügbaren Parkplätze anzuzeigen.
Eine einmalige Nutzung dieses System als Service ließe sich bei Großveranstaltungen wie Fußballspiele, den Hamburger Dom oder Konzerte anbieten, sodass die Kunden, die es in unregelmäßigen Abständen nutzen, dies als Service-on-Demand Angebot bekämen. Der Veranstalter könnte wiederum die Nutzung dieses System seinen Kunden im Zeitraum der Veranstaltung mit dem Erwerb der Veranstaltungskarte ermöglichen. Dadurch könnte sich der Kunde einen Parkplatz reservieren, oder es wird ihm durch den Erwerb der Veranstaltungskarte direkt einer zugewiesen, falls die Parkplätze der Veranstaltung keine öffentlichen sondern private Parkplätze sind. Dies ermöglicht es, dem Kunden bei der Anreise zu der Veranstaltung direkt zum Parkplatz zu navigieren. Weiterhin kann die Abrechnung digital stattfinden, was bei privaten Parkplätzen den Personal- und Zeitaufwand für die Abrechnung der Parkplätze jedes einzelnen Kunden einspart.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist, einen Prototypen zu entwickeln, der ein solches System umsetzen kann. Dieser Prototyp soll am Ende in der Lage sein, Parkplätze auswerten zu können, diese dann an eine Datenbank zu senden und eine Webkarte für den Endbenutzer bereitstellen, auf der die freien Parkplätze angezeigt werden. Das Hauptaugenmerk liegt bei dieser Arbeit dabei auf den Entwurf und die Prüfung der Umsetzung der einzelnen Teilaufgaben für ein solches System. Dabei soll geprüft inwieweit sich die einzelnen Teilaufgaben umsetzen lassen können.

1.3 Aufbau der Arbeit

Zuerst wird in dieser Arbeit der Stand der Technik im Bezug auf schon vorhandene Umsetzungen von *Local Dynamic Maps*, sowie das Konzept der Parkplatzerkennung in Fahrzeugen, die Möglichkeiten der Darstellung von geographischen Daten in Webanwendungen und die Vorgaben bezüglich der Maße von Parkplätzen beschrieben. Zudem werden die schon vorhandenen Systeme aufgezählt. Das drauf folgende Kapitel umfasst die Aufstellung des Use Cases, die Beschreibung, Bewertung und Auswahl der Ansätze zur Umsetzung des Use Cases. Weiterhin werden in diesem Kapitel Kriterien aufgsetellt, an dem der Prototyp gemessen wird. Daraufhin beschreiben die nächsten beiden Kapitel den das Konzept zu den ausgewählten Ansatz zur Umsetzung des Use Cases und deren Realisierung. Zum Schluss folgt das Kapitel, welches die Messung und deren Auswertung aufführt und die Kriterien überprüft, abgeschlossen mit dem Kapitel welches das Fazit der Arbeit zieht und den Ausblick auf mögliche Erweiterungen und Anknüpfungspunkte an diese liefert.

2 Grundlagen und Stand der Technik

Dieses Kapitel umfasst die Grundlagen und den Stand der Technik zu der vorherigen Motivation dieser Arbeit.
Die Idee einer dynamischen Karte, die Informationen aus und für den Straßenverkehr enthält, welche zu dem nicht nur statische sondern auch dynamische Daten beinhaltet und darstellt, ist bereits mit dem Konzept einer *Local Dynamic Map* in verschiedenen Kontexten, zumeist im Bezug auf das autonome Fahren umgesetzt worden.
Das zugrundeliegende Konzept einer *Local Dynamic Map* und deren Umsetzung von einigen projetkbezogenen Kontexten wird in diesem Kapitel aufgeführt. Weiterhin werden hier verschiedene Ansätze und Anbieter von Darstellungen von geographischen Daten auf Online abrufbaren Karten beleuchtet und die Regularien bezüglich der Abmessungen von Parkplätzen erklärt. Ferner wird der derzeitige Stand der benötigten Sensorik zur Auswertung der Parkplätzen erläutert, sowie die vorhandenen Systeme, an welche in dieser Arbeit angeknüpft werden soll aufgeführt.

2.1 Local Dynamic Map

Ein wichtiger Aspekt für die erfolgreiche Umsetzung des autonomen Fahren ist die *Local Dynamic Map(LDM)*. Für das Konzept der *Local Dynamic Map* existieren Standardisierungen zur Umsetzung dieses Konzepts, zum Beispiel durch das European Telecommunications Standards Institute (ETSI) [17] oder von der International Organization for Standardization (ISO) [42], an welchen sich diese Arbeit orientiert. Laut der ISO-Norm ist eine *Local Dynamic Map* „eine Instanz die *Local Dynamic Map* Datenobjekte, Dienste und Schnittstellen zur Veränderungen dieser Datenobjekte enthält“ [43]. Diese Art der Karte beinhaltet Informationen über und um den Straßenverkehr, welche sich je nach Art der Information verschieden dynamisch, also in verschiedenen Zeitintervallen ändern. In den Zeitintervallen, in welchen sich die Informationen sich ändern, unterscheidet dass SAFESPOT-Projekt, ein Projekt zur Erstellung dynamischer, kooperierender Netzwerke, welche sich mit Fahrzeugen und Verkehrsinfrastruktur befassen, in vier unterschiedlichen Ebenen [3]. Diese sind:

- **Statische Karte:** Die Straßenkarte mit den eingezeichneten Straßen, Autobahnen und Wegen
- **Landmarken:** Feststehende Objekte wie Bäume, Häuser, Straßenschilder etc.
- **Temporäre Objekte:** Dies können Baustellen, Unfälle oder auch vorübergehende, den Verkehr einschränkende Wetterphänomene sein
- **Dynamische Objekte:** Im Verkehr dynamisch verändere Objekte wie die Postion des eigenen Autos, die Positionen anderer Verkehrsteilnehmer oder die aktuelle Ampelschaltung

Diese Unterscheidung des SAFESPOT-Projekts in in der Abbildung 2.1 dargestellt [3].

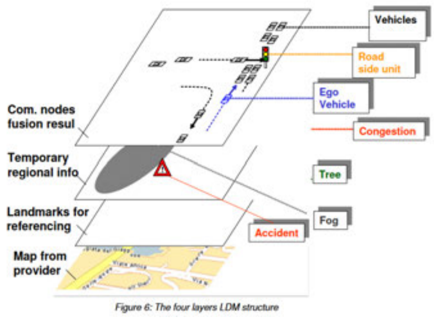

Abbildung 2.1: Die vier Ebenen einer *Local Dynamic Map* [3]

Für die Implementierung einer *Local Dynamic Map* benötigt es bei dem generellen Aufbau folgende Anforderungen:

- Eine Datenbank
- Ein Datenbankmanagmentsystem
- Eine API
- Eine Anwendung

Dieser Aufbau ist in der Abbildung 2.2 dargestellt [35].

Die Datenbank beinhaltet die Daten, die in der *Local Dynamic Map* dargestellt werden sollen. Die Form, in welcher die Daten in der Datenbank gespeichert werden, lässt sich

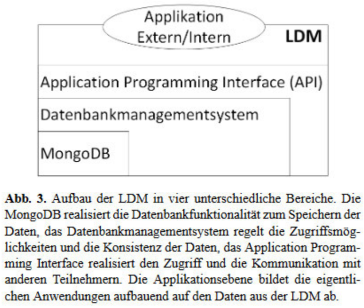

Abbildung 2.2: Anforderungen an den Aufbau einer Local Dynamic Map [35]

anhand eines Datenbankmanagmentsystem definieren. Vielfach genutzt wird hierbei die Datenbank *PostgreSQL* und die dazugehörige Erweiterung *PostGIS*, da damit die Daten in der Datenbank als Geodaten dargestellt werden können und räumliche Operationen zur Berechnungen bereitgestellt sind. Diese Art des Vorgehens wird im SAFESPOT-Projekt genauer erläutert und wird hierbei als PostGIS Local Dynamic Map (PG-LDM) bezeichnet. Dieser Ansatz baut auf der *Tele Atlas Automotive Research Platform* im *CIVIC* auf und wurde im Rahmen des Projekts um „leicht verschiedene Konfigurationen " erweitert [1].
Ein weiterer Ansatz ist die *NAVTEQ-LDM*, diese beinhaltet eine „einzelne statische Biblotheken Datei", welche die Funktionen und Klassen der API bereitstellt. Es baut im Gegensatz zur PG-LDM auf SQL-Lite auf [2].

Ein weiterer wichtiger Baustein zur Umsetzung einer *Local Dynamic Map*, ist die Schnittstelle (API), mit welcher die gewünschten Daten an die Datenbank übertragen und von dort wieder abgerufen werden können. Dadurch soll allen wichtigen Teilnehmer die Möglichkeit geboten werden, ihre gemessenen und für die *Local Dynamic Map* wichtigen Daten, auch an diese übersenden zu können. Zudem kann eine entsprechende Anwendung, zum Beispiel eine Karte, auf der die Daten angezeigt werden sollen, mittels API auf die *Local Dynamic Map* zugreifen.
Bei der Umsetzung der API wird unter anderem eine eigens programmierte API in C++ genutzt, um so für eine bessere Performance zu sorgen und die Latenzen für die Eintragungszeiten gering zu halten [41].
Ein Websever der RESTful konform ist, ist auch eine genutzte Umsetzung einer API, da hiermit mit JSON eine für den Menschen lesbare Übertragung der Daten an die Datenbank ermöglicht wird [36].

Die Anwendungen/Appliaktionen einer *Local Dynamic Map* können je nach Projekt sehr unterschiedlich ausfallen, was einen Vorteil der *Local Dynamic Map* darstellt. Diese kann den Anforderungen des Projektes entsprechend angepasst werden. Dabei steht im Vordergrund, die Daten aus der Datenbank dem Endbenutzer zu visualisieren oder den Fahrzeugen diese Daten verfügbar zu machen. Eine Unterscheidung zwischen internen und externen Anwendungen wird auch getroffen. Interne Anwendungen sind in der *Local Dynamic Map* enthalten und bieten deshalb von sich selbst heraus einen Mehrwert, während externe Anwendungen Daten durch Anfragen an die Datenbank übermittelt bekommen um diese weiter zu verarbeiten [36].
Der Artikel „Local Dynamic Map als modulares Software Framework für Fahrerassistenzsysteme“ [36] nennt als Anwendungen beispielhaft eine genaue Verkehrzeichenkarte, welche in ähnlicher Form in der Bachelorarbeit *Automatisierte Erstellung von dynamischen Karten für das Autonome Fahren in komplexen, urbanen Umgebungen* umgesetzt wurde [51] und das *Map Matching*. Bei ersten Beispiel, der Verkehrszeichenkarte, handelt es sich um ein Konzept bei dem zum einen die Lokalisierung der Fahrzeuge anhand der Verkehrszeichen im Straßenverkehr stattfinden soll und zum anderen die Lokalisierung und Erfassung von Verkehrsschildern aus der Bestimmung der eigenen Position des Fahrzeuges und die relative Entfernung dieses Fahrzeuges zu den Verkehrsschildern erfolgen soll. Beim zweiten Beispiel wird sich mit der Unfallvermeidung durch das Prinzip *Vehicle in Front* befasst. Hierbei sollen die Informationen aus der Umgebung aufgenommen werden und mittels *Map Matching* so extrahiert werden, dass andere Fahrzeuge auf den Spuren der Straße erkannt werden, also in eine *spurgenaue Umgebungskarte* transferiert werden können. Damit kann dann berechnet werden ob die Möglichkeit für das eigene Fahrzeug besteht, die anderen Verkehrsteilnehmer zu erreichen und ob somit eine Unfallgefahr, durch die daraus mögliche Kollision, besteht [36].
Der Artikel „Implementation and Evaluation of Local Dynamic Map in Safety Driving Systems“ befasst sich im Rahmen der Anwendung einer *Local Dynamic Map* mit einer sehr ähnlichen Problemstellung, auch mit einer Unfallvermeidungsanwendung, allerdings wird in diesem Rahmen auch noch die Berechnungszeiten je nach Verkehrsaufkommen, analysiert [41].
Es lässt sich festhalten, dass sich viele Projekte die sich mit dem Konzept der *Local Dynamic Map* auseinandersetzen und Problemstellungen mit diesem Konzept lösen, die vornehmlich die höchste und damit dynamischste Ebene betrachten und für ebenjene ihre Anwendungen erstellen. Zudem gibt es auch einige Anwendungen die sich mit dem

Eintragen und Updaten spezifischer Umfeldsmerkmale befassen, mit welchen auch eine Positionsbestimmung möglich ist.

2.2 Kartendarstellungen im Serverumfeld

Dieses Unterkapitel befasst sich mit Anbietern und Lösungen, die Server mit Geokarten bereitstellen, welche für diese Art von Darstellungen von Daten im Kontext von Geoinformationsystemen geeignet sind.

2.2.1 OpenStreetMap

OpenStreetMap [9] ist ein Open-Source-Projekt, welches es sich zur Aufgabe gemacht hat, mit Hilfe der Community eine Datenbank mit Karteninformationen zu erstellen und upzudaten, welche dazu kostenlos genutzt werden kann. Hierbei erhebt das Projekt keinen Anspruch auf absolute Vollständigkeit und Korrektheit, weist aber darauf hin, im Gegensatz zu kommerziellen Anbietern keine bewusst falschen Daten zum Plagiatsschutz in die Karten einzubauen [28].
Die Daten, welche OpenStreetMap anbietet, können entweder als Rohdaten genutzt werden oder in Kartenform, als vor berechnete Kartenbilder. Diese können einfach in die eigene Webseite eingebunden werden. Dies ist entweder als statisches Bild oder in HTML eingebettet möglich [26]. Die Rohdaten können über verschiedene Seiten, die mit OpenStreetMap arbeiten heruntergeladen werden. In Deutschland bietet die Geofabrik diese Option an [29].
OpenStreetMap bietet zusätzlich Möglichkeiten an, sowohl in Software- als auch in Webanwendungen genutzt werden zu können. Java-Script verfügt über die Bibliotheken *Leaflet* und *OpenLayers*. Außerdem bietet es die Möglichkeit mit GIS-Software (Software die mit Kartendaten arbeitet) kompatibel zu sein und kann aus den OpenStreetMap-Daten, *Shapefiles* und *PostGIS*-Datenbanken erstellen [25].
OpenStreetMap und deren Anwendungen bringen eigene Dateiformate mit. Einige Beispiele sind [24]

- *.osm* Format
- *.pbf* Format
- JSON in OSM Format

Das *.osm*-Format stellt ein XML-Dateiformat dar, welches die Grundelemente *nodes*, *ways* und *releations* sowie deren *tags* beinhaltet. Mit diesem Format lässt sich die Erde als Kartenformat darstellen [23].
Das *.pbf*-Format, ist ein Dateiforamt, welches für die Ablösung des *.osm*-Format angedacht ist, da weniger Speicher gebraucht wird und die Zugriffe schneller erfolgen können [22].
Es existiert auch ein JSON-OSM-Format, mit der sich die OSM-Daten mit leichten Anpassungen im Header im JSON-Format speichern lassen [27].

2.2.2 ArcGIS

ArcGIS umfasst viele verschiedene Softwareprodukte, die das Unternehmen ESRI anbietet. Diese Softwareprodukte sind alle im Themenbereich der Geoinformationen und Cloudservices angesiedelt. Bis zu einen bestimmten Grad, abhängig von der Größe der Karte und die Menge der Daten, können einige Softwareprodukte kostenlos verwendet werden, abseits dessen entstehen Kosten durch ein Abomodell.
Die Funktionen enthalten unter anderem die Möglichkeit eigene Karten zu erstellen, welche dann mit eigenen Daten gefüllt werden können. Diese Karten können auch dynamische Karten sein, dessen Darstellung der Daten sich bei Veränderungen der zugrundeliegenden Daten anpasst. Es lassen sich unter anderem animierte Karten mit zeitlichen Verlauf erstellen, interaktive 3-D Karten, welche die Möglichkeit bieten, bestimmte Daten als 3-D Objekte darzustellen und eine Interaktion mit diesen Objekten ermöglicht und Karten, welche die Daten in Abhängigkeit ihrer Größe auf der Karte visualisieren.
Zudem besteht die Möglichkeit für den Anwender der Karte, eigene Analyse Tools zu erstellen, mit welchen die Daten übersichtlicher und je nach Problemstellung, der Datenverarbeitung angepasst und abgebildet werden können. Hierbei lassen sich auf die einzelnen Layer mit den jeweiligen Daten zugreifen und darstellen, sowie die Polygone, Linien und Punkte auf den einzelnen Layer dem Problem angepasst zugänglich für den Endbenutzer nutzbar zu machen. Auch lassen sich auf den Daten mathematische Operationen durchführen, womit Muster in den eigenen Daten besser erkannt werden können. Mit der Möglichkeit zur Erstellung von 3-D Karten lassen sich auch Karten erstellen, in den Gebäude angeguckt und Landschaften mit vorgefertigten 3-D Objekten geplant werden können [5].
Die Struktur von ArcGIS-Karten basiert auf Layers, die sich in *FeatureLayers* und *GraficsLayers* unterscheiden. Ersteres befasst sich mit strukturierten Daten, letzteres mit

unstrukturierten Daten. Auch mit Subkategorien dieser Layer, wie zum Beispiel CSV-, GeoRSS-, GeoJSON-, WFS-, WMS- und OpenStreetMapLayers, kann in der Entwicklung gearbeitet werden [6].
ArcGIS verfügt über eine REST-API, mit welcher es möglich ist über HTTP-Request mit den ArcGIS-Server zu kommunizieren. Dabei lassen sich verschiedene Services, Funktionen und Daten aufrufen, Daten aktualisieren sowie die Zugriffsrechte der Endbenutzer anpassen [7].
Die Abbildung 2.3 ist ein Beispiel einer Implementation von Daten auf einer Webkarte. Die dargestellten Daten, sind die Schienen und Schienenendpunkte von der Umgebung um Los Angeles herum. Diese Daten stammen aus dem Tutorial *Import data as a feature layer* [8].

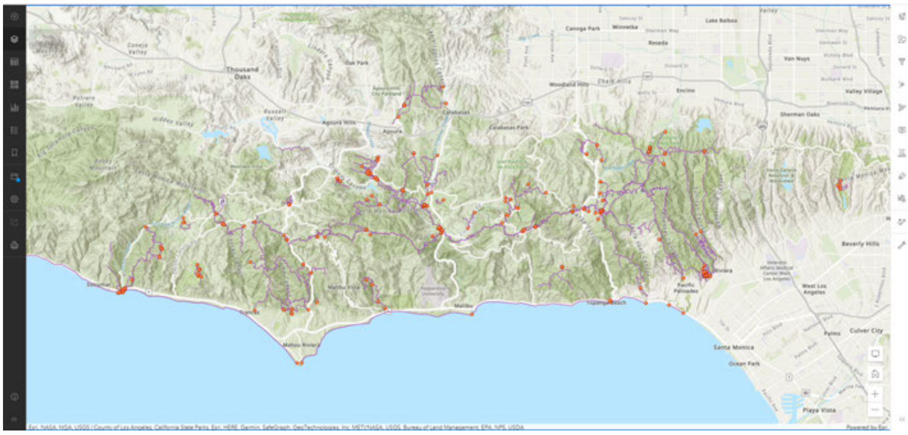

Abbildung 2.3: Beispiel einer Darstellung einer ArcGIS-Karte mit Schienenendpunkte in der Umgebung von Los Angeles

2.2.3 Geoserver

Geoserver ist im Gegensatz zu ArcGIS eine OpenSource und Free Software Lösung, um geographische Daten auf einer Karte auf einem Webserver anzeigen zu lassen. Nach eigenen Angaben ist der Vorteil von der Nutztung des Geoservers die hohe Flexibilität bei Erstellungen eigener Karten und dem Datenaustausch. Geoserver kann mit den Standards *Web Map Service (WMS)*, *Web Feature Service (WFS)* und *Web Coverage Service (WCS)* umgehen, ferner besteht mit der Bibliothek OpenLayers eine einfache Möglichkeit zur Kartenerstellung und -bearbeitung. Zudem können Shapefiles, welche mit ArcGIS kompatibel sind, da sie dafür entwickelt wurden, und PostGIS Tables genutzt werden um

Daten an den Webserver weiterzugeben [11].
PostGIS ist eine Erweiterungen für die Datenbank PostgreSQL. PostGIS bietet hierfür viele verschiedene räumliche Objekte und Funktionen für PostgreSQL an [30].
Da Geoserver mit OpenLayers eine Java-Script Schnittstelle besitzt, kann mittels Java-Script und HTML die Webkarte programmiert werden, die sehr genau auf die gewünschte Lösung einer Problemstellung angepasst werden kann. Im Gegensatz zu ArcGIS bietet Geoserver jedoch keine Tools zur Unterstützung der Darstellung der Daten. Die Abbildung 2.4 zeigt wie die beispielhafte Implementierung mit dem vorhandenen Workspace und Daten des Geoserver aussehen kann. Diese Daten beinhalten die Straßen von Manhatten. Mittels einer HTML-Datei ist der Layer mit dem Straßen von Manhatten mit einem Layer der Weltkarte überlagert.

Abbildung 2.4: Beispiel einer Darstellung einer Geoserver-Karte mit den Straßen von Manhatten

2.3 Automatische Parkplatz Erkennung

Der Einparkassistent, welcher es ermöglicht, dass das Auto automatische einparken kann, greift den Aspekt der Parklückenerkennung für die Funktionalität des Einparkens auf. Der Bosch Einparkassistent wird im Buch Fahrstabilisierungssysteme und Fahrerassistenzsysteme [34] beschrieben. Hierbei unterscheidet der Bosch Einparkassistent [34] die Umsetzungen in drei verschiedene Stufen. Interessant für diese Arbeit ist die erste Stufe des Einparkassistenten, die Parklückenvermessung. Die zweite Stufe ist der Informierender Einparkassistent, der den Lenkradeinschlag, für ein optimales Einparken anzeigt. Die dritte Stufe ist der lenkende Einparkassistent, bei dem das Fahrzeug sich selbst einparken kann. Hier wird nur die erste Stufe betrachtet. Hierbei wird die Breite und Länge der Parklücke über jeweils ein zusätzlichen Ultraschallsensor pro Fahrzeugseite und einen Radimpulssensor erfasst. Die beiden Ultraschallsensoren messen den Abstand der jeweiligen Fahrzeuge zum nächsten Hindernis, wie ein parkendes Fahrzeug. Dies stellt die Breite der Parklücke dar. Ist ein Fahrzeug in der Parklücke, wird eine zu kleine Parklückenbreite gemessen. Wenn kein parkendes Fahrzeug im Weg ist, wird der Abstand zum Bordstein gemessen und falls der Abstand groß genug ist, kann es eine, für das Fahrzeug, geeignete Parklücke darstellen. Des weiteren wird mit Hilfe des Radimpulssensors die Länge der Parklücke ausgemessen. Erst wenn die Länge und die Breite groß genug für das Fahrzeug zum Einparken ist, wird dieser dem Fahrer auf dem Display als möglicher Parkplatz angezeigt, in welcher dann auch mit Hilfe des Einparkassistenten automatisch eingeparkt werden kann. Dieses Vorgehen ist in der Abbildung 2.5 dargestellt.

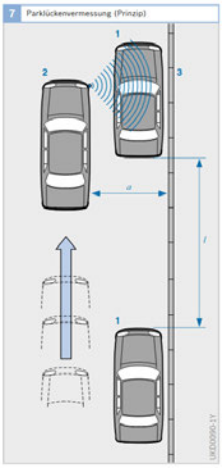

Abbildung 2.5: Das Prinzip der Parklückenvermessung [33]

Die Zahl *1* beschreibt auf dem Bild die geparkten Fahrzeuge, die Zahl *2* das Fahrzeug, welches einparken soll, die Zahl *3* die Parkfront. Der Buchstabe a gibt den Abstand vom einparkenden Auto zur Parkfront an und der Buchstabe l die Länge der Parklücke [34].

2.4 Bestimmungen zur Maße von Parkplätzen

Die Maße der Parkplätze sind vorgegeben. Hierbei wird grundlegend zwischen Parkplätzen in Parkanlagen wie Garagen und Parkhäusern zu Parkplätzen die am Straßenrand und außerhalb überdachter Parkanlagen sind, unterschieden.

Die Garagenverordnung des Bundeslands Hamburg hat im §6 Abs. 1 als Vorgabe folgende Bestimmungen für Parkplätze für Kleinfahrzeuge in Garagen [14], diese sind der Tabelle 2.1 aufgeführt.

Länge	Breite	Besondere Anforderungen
5 Meter	2,30 Meter	Keine Längsseite vorhanden
5 Meter	2,40 Meter	Eine Längsseite vorhanden
5 Meter	2,50 Meter	Beide Längsseiten des Stellplatzes durch Wände, Stützen, andere Bauteile oder Einrichtungen begrenzt
5 Meter	3,50 Meter	Stellplatz für Menschen mit Behinderung

Tabelle 2.1: Bestimmung für Garagenstellplätze

Nach der *Empfehlungen für Anlagen des ruhenden Verkehrs* [50] wird für die Abmessungen von Parkständen für PKWs im Straßenraum folgende Empfehlungen getroffen. Für Parkstände in Längsstellung gilt

- Die Breite beträgt 2 Meter
- Die Länge beträgt
  - 5,70 m für Rückwärtseinparken
  - 6,70 m für Vorwärtseinparken

Für Parkstände in Schräg- und Senkrechtstellungen gelten die Vorgaben der Tabelle 2.2. Hierbei stellt die Einheit *gon* den Vollwinkel in 400 °dar.

Breite	Länge	Aufstellungswinkel in gon
2,50 Meter	3,54 Meter	50 gon
2,50 Meter	3,09 Meter	60 gon
2,50 Meter	2,81 Meter	70 gon
2,50 Meter	2,63 Meter	80 gon
2,50 Meter	2,53 Meter	90 gon
2,50 Meter	2,50 Meter	100 gon

Tabelle 2.2: Bestimmungen für Parkstände in Schräg- und Senkrechtaufstellung

In der Abbildung 2.6 sind die drei verschiedenen Arten der Orientierung der Parkplätze abgebildet.

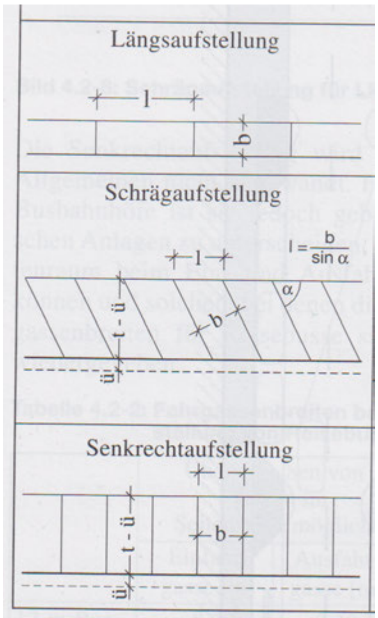

Abbildung 2.6: Abbildung zur Orientierung bezüglich der Parkplatz Maßen

2.5 Vorhandene Systeme

Dieses Unterkapitel beinhaltet eine Übersicht zu den schon vorhandenen Systemen, die Hardware und Software, die zur Verfügung steht, auf welche aufgebaut werden kann und soll, um die Problemstellung dieser Arbeit zu lösen.
Das ganze System, sowohl die Hardware und die Software befindet sich in einem Tesla Model S welches dem HAW Urban Mobility Lab gehört und somit für solche Arbeiten genutzt werden kann.

2.5.1 Hardware

Zwei der schon vorhandenen Systeme werden für die Umsetzung der Aufgabenstellung genutzt. Das ist der LiDAR-Sensor *Velodyne HDL-32E* [48] und der GNSS-Sensor *SBG System Apogee-D* [44]. Für den Aufbau werden die Sensoren, der Velodyne, der SBG-Apogee-D, die Ladybugkamera und die Stereokamera auf einer Vorrichtung auf das Dach des Teslas montiert. Dies ist in der Abbildung 2.7 dargestellt. Die Kabel führen über eines der hinteren Fenster in den Kofferraum. Im Kofferraum befinden sich der Computer, der Akku und die Sensoren mit Verbindung zum Computer, wie in Abbildung 2.8 abgebildet ist.

Abbildung 2.7: Dachaufbau mit den Sensoren

Abbildung 2.8: Der Computer, der Akku und die Sensoren im Kofferraum

LiDAR: Velodyne HDL-32E

Zum Messen der Umgebung mit einem LiDAR-Sensor wird der Velodyne HDL-32E verwendet. Die Messung mit einem LiDAR-Sensor zeichnet sich dadurch aus, dass die Umgebung mittels Lasern abgetastet wird und diese Daten in einer Punktwolke gespeichert und dargestellt werden können. Im Bereich des autonomen Fahrens wird häufig ein LiDAR-Sensor eingesetzt, der die Umgebung um das Fahrzeug abtastet und daraus eine Punktwolke rund um das Fahrzeug herum erstellen kann.
Der Velodyne HDL-32E ist zu dieser Messung auch in der Lage. Hierfür benutzt der Sensor 32 Laser. Die Anzahl der Laser eines Sensor bestimmt die Anzahl der Punkte in den Punktwolken und somit auch die Auflösung in welcher die Umgebung ertastet werden kann.
In der Tabelle 2.3 sind einige wichtige Spezifikationen und deren Parameter aufgeführt.

Spezifikationen	Parameter
Messreichweite	Bis zu 100 m
Messgenauigkeit	Typischerweise bis zu $\pm$ 2 cm
Vertikale Sichtweite	10,67 °bis -30,6 °(41,33 °)
Vertikale Auflösung	1,33 °
Horizontale Sichtweite	360 °
Horizontale Auflösung	0,1 °bis 0,4 °
Rotation Rate	5 Hz - 20 Hz
Energieverbrauch	12 Watt
Spannungsversorung im Betrieb	9 V - 18 V
Generierte 3D-LiDAR Datenpunkte	Ca. 690.000 Punkte im Single Return Mode und ca. 1.390.000 im Dual Return Mode

Tabelle 2.3: Spezifikation und Parameter zu dem Velodyne HDl-32E [15]

Für die Umsetzung der Arbeit ist die Anbringung des Velodyne HDL-32E und somit die Messhöhe wichtig. In der Nachfolgenden Abbildung 2.9 ist ein CAD-Modell von dem Tesla und dessen Aufbau dargestellt [47]. Der Velodyne-Sensor ist in der Abbildung 2.10 abgebildet [15].

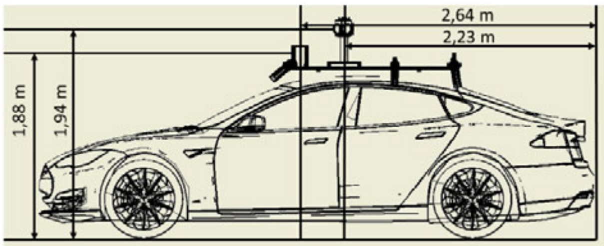

Abbildung 2.9: Das CAD-Modell mit den Aufbauten des Teslas [47]

Abbildung 2.10: Abbildung des Velodyne 32-HDL-E [15]

GNSS: SBG Systems Apogee-D

Zur Positionsbestimmung des Autos, wird der vorhandene GNSS-Sensor SBG System Apogee-D genutzt. Dieser besitzt zwei Antennen, mit denen eine Positionsbestimmung in Echtzeit ermöglicht wird, auch Echtzeitkinematik genannt. Die benötigten Daten, Longitude, Latitude und die Orientierung des Sensors im Bezug zum Norden, erfasst dieser Sensor. In der Tabelle 2.4 sind einige wichtige Spezifikationen und deren Parameter aufgeführt.

Spezifikationen	Parameter
Energieverbrauch	7 Watt
Spannungsversorung im Betrieb	9 V - 36 V
Update Rate	200 Hz
Kaltstart	45 s
Warmstart	15 s
Horizontale Messgenauigkeit	0,6 cm - 100 cm

Tabelle 2.4: Spezifikation und Parameter zu den SBG Systems Apogee-D [45]

Der Apogee-Sensor ist in der Abbildung 2.11 abgebildet [45].

Abbildung 2.11: Abbildung des SBG-System-Apogee-D [45]

2.5.2 Software

Um die vorhandene Sensorik auszulesen, wird das *Robot Operating System (ROS)* [38] benutzt, mit ROS werden die Daten der Sensorik über schon vorhandene Programme erfasst und verarbeitet. ROS ist ein Open-Souce-Meta-Betriebssystem, welches auf den Umgang mit Robotern und deren Sensorik spezialisiert ist. Das Konzept von ROS besteht aus Nodes, die untereinander kommunizieren. Die Nodes werden in verschiedenen Packages gefasst, welche eine Einheit darstellt, die den Programmcode und deren Abhängigkeiten sowie die ausführbare Datei enthält. Die Kommunikationen zwischen den Nodes findet über Topics statt, die mittels einen Publisher veröffentlicht und mittels einen Subscriber abonniert werden können. Die Topics haben verschiedene Topic-Typen in denen die Struktur der übertragenden Daten aufgebaut ist. Dies können die üblichen, aus den gängigen Programmiersprachen bekannten Datentypen sein, wie zum Beispiel Bool, Byte, Float, Int und String, aber auch für die Sensorik spezialisierte Datenstrukturen der Nachrichten, auch Messages genannt, wie zum Beispiel PointCloud2-Daten, Navigationsdaten oder verschiedene Geometriedaten sein. Es können auch eigene Nachrichten erstellt werden, diese haben den Vorteil, dass hierbei die Struktur der übertragenen Daten mit den jeweiligen Datentypen selbst gewählt werden kann und deshalb ebenjene Nachrichten spezifisch an die eigene Problemstellungen anpassbar sind. Somit ermöglicht ROS einen Datenaustausch von üblichen Datentypen, auf Sensorik spezialisierte Datentypen oder eigens entwickelte Datentypen.
Die vom Computer genutzte ROS-Version ist die Version *ros-kinetic* [39].

3 Use Case und Anforderungen

In diesem Kapitel wird auf die zugrundeliegende Idee dieser Arbeit näher eingegangen und der daraus folgende Use Case definiert. Danach werden zwei verschiedene Ansätze für den Use Case vorgestellt. Diese beiden Ansätze werden miteinander verglichen, die Unterschiede, sowie die Vor- und Nachteile beider Ansätze betrachtet.
Im zweiten Teil dieses Kapitel werden die Anforderungen an die beiden Ansätze beschrieben und dann wird eine Bewertung der Ansätze, im Bezug auf die Erreichbarkeit der Anforderungen, vorgenommen.

3.1 Use Case

Der Use Case dieser Arbeit ist, dass ein Endbenutzer anhand einer Webkarte sich die freien Parkplätzen am gewünschten Ort anzeigen lassen kann. Daraus sind folgende vier verschiedene Fälle des Use Cases abgeleitet, wie der Use Case vom Endbenutzer aufgefasst werden kann.

1. Der Endbenutzer möchte an seinem Standort den nächstgelegenen Parkplatz angezeigt bekommen

2. Der Endbenutzer möchte an seinem Standort in einem bestimmten Radius alle verfügbaren Parkplätze angezeigt bekommen

3. Der Endbenutzer möchte an seinem Zielstandort den nächstgelegenen Parkplatz angezeigt bekommen

4. Der Endbenutzer möchte an seinem Zielstandort in einem bestimmten Radius alle verfügbaren Parkplätze angezeigt bekommen

Der Endbenutzer hat die Auswahl zwischen dem eigenen oder einem Zielstandort als gewünschten Standort, bei dem die freien Parkplätze angezeigt werden sollen. Zu dem besteht für den gewünschten Standort die Auswahlmöglichkeit den nächstgelegenen Parkplatz oder alle Parkplätze innerhalb eines gewünschten Radius anzeigen zu lassen.
Ein System, welches den Use Case für den Endbenutzer erfüllt, muss in der Lage sein freie Parkplätze zu erfassen, abzuspeichern und je nach gewünschter Option und Standort dem Endbenutzer anzuzeigen.

3.2 Ansätze für das Erfassen der Parkplätze

Dieses Unterkapitel befasst sich mit zwei verschiedenen Ansätzen wie die freien Parkplätze erfasst werden können. Hierbei gibt es den Ansatz die Parkplätze selbst zu erfassen und in eine Datenbank zu speichern wenn diese frei sind und den Ansatz vor eingetragene Parkplätze aus einer Datenbank zu überprüfen, ob diese frei sind.

3.2.1 Ansatz I: Parkplatz selbst erfassen und abspeichern

Dieser Ansatz beruht darauf mittels Sensoren einen Parpklatz zu erkennen, die Abmaße und Orientierung zu messen, sowie die Koordinaten und die Uhrzeit zu erfassen. Diese Informationen werden dann in die Datenbank der *Local Dynamic Map* eingetragen. Daraus lässt sich, dass

- auf einer zweispurigen Straße, während einer Fahrt mit dem Fahrzeug, der vom Fahrer gesehene rechte Straßenrand nach Parkplätzen, mittels Sensoren, erfasst wird und bei einem erfolgreichen Erkennen eines Parkplatzes folgende Informationen zu dem vermessenen Parkplatz auf eine Karte eingetragen werden:
  - Länge
  - Breite
  - Orientierung
  - Uhrzeit
  - GPS Position

3.2.2 Ansatz II: Parkplatz aus Datenbank überprüfen und Status aktualisieren

Dieser Ansatz nutzt eine schon vorhandene Liste an Parkplätzen in einer Datenbank der *Local Dynamic Map*. In dieser Datenbank ist zu jedem Parkplatz deren Abmessungen, Orientierung und die Koordinaten vorhanden. Während der Fahrt soll mittels Sensoren überprüft werden, ob der Parkplatz „Frei“ oder „Belegt“ ist und diese Informationen in die Datenbank eingetragen werden. Daraus lässt sich ableiten, dass

- auf einer zweispurigen Straße, während einer Fahrt mit dem Auto, der vom Fahrer aus gesehene rechte Straßenrand nach Parkplätzen, welche in der Datenbank der *Local Dynamic Map* vorhanden sind, mittels Sensoren, erfasst und überprüft wird, ob diese Parkplätze „Frei“ oder „Belegt“ ist. Diese Information soll mit der Uhrzeit der Messung an die Datenbank der *Local Dynamic Map* übergeben werden.

3.2.3 Unterschiede der Ansätze zur Parkplatzerfassung

Dieses Unterkapitel befasst sich mit den Unterschieden der Ansätze *Parkplatz selbst erfassen und abspeichern* und *Parkplatz aus Datenbank überprüfen und Status aktualisieren.* Der Ansatz *Parkplatz selbst erfassen und abspeichern* weist bei der Erfassung von freien Parkplätzen dabei folgende Unterschiede auf:

- Es werden noch nicht bekannte freie Parkplätze mit Uhrzeit, GPS-Position, Länge, Breite und Orientierung erfasst
- Die Informationen der Länge, Breite und Orientierung des Parkplatzes müssen zusätzlich erfasst werden
- Die noch nicht bekannten Parkplätze werden mit den Daten an die Datenbank gesendet

Der Ansatz *Parkplatz aus Datenbank überprüfen und Status aktualisieren* weist bei der Erfassung von freien Parkplätzen folgende Unterschiede auf:

- Es werden keine neue freien Parkplätze in die Datenbank eingetragen
- Es werden nur die vorhandenen Parkplätze auf „Frei“ oder „Belegt“ geprüft

3.2.4 Vor- und Nachteile des Ansatzes Parkplatz selbst erfassen und abspeichern

Dieser Unterpunkt beschreibt die Vor- und Nachteile des Ansatzes *Parkplatz selbst erfassen und abspeichern*.
Die Vorteile dieses Ansatzes sind:

- Reale Abmessungen des Parkplatzes wird in die Datenbank der *Local Dynamic Map* gespeichert
- Eigenständiges Update der Datenbank in Bezug auf neue Parkplätze möglich

Die Nachteile dieses Ansatzes sind:

- Die realen Abmessungen zu bestimmen bedeutet mehr Aufwand für das Programm, welches die Auswertung umfangreicher ausfallen lässt. Je nach Umsetzung können auch mehr Sensoren nötig sein
- Bei mehreren theoretischen Parkplätzen, die nebeneinander frei sind, wird dieser nur als ein großer erkannt
- Bei der Erfassung neuer freier Parkplätze müssen Fehlinterpretationen ausgeschlossen werden. Die können als freier Parkplatz erkannte Ausfahrten, dynamische Parkverbote wie Baustellen und Umzüge, sowie dauerhafte Parkverbote sein

3.2.5 Vor- und Nachteile des Ansatzes Parkplatz aus Datenbank überprüfen und Status aktualisieren

Dieser Unterpunkt beschreibt die Vor- und Nachteile des Ansatzes *Parkplatz aus Datenbank überprüfen und Status aktualisieren*.
Die Vorteile dieses Ansatzes sind:

- Lediglich eine signifikante Überschneidung aus den Flächen des Parkplatzes und der freien Fläche muss ermittelt werden um „Frei“oder „Belegt“ klassifizieren zu können
- Parkplätze können in der Datenbank ergänzt werden, ohne dass diese vom Auto vermessen werden müssen. Andere Quellen können herangezogen werden
- Umgeht das Problem bei dem mehrere Parkplätze als einer erfasst wird

Die Nachteile dieses Ansatzes sind:

- Keine reale Abmessung der freien Fläche des Parkplatzes
- Kann zu Problemen führen, wenn die Autos zu ungenau auf den Parkplatz parken, wenn diese z.B. zwei zur Hälfte belegen
- Arbeitsaufwand bei der Eintragung von neuen Parkplätzen in die Datenbank

3.3 Anforderung und Bewertung an die beiden Konzepte

In der Tabelle 3.1 sind die Anforderungen an die beiden Ansätze, die sie als Umsetzung als System erfüllen müssen, aufgeführt. Die True-Positive-Parkplätze geben dabei die Parkplätze an, die frei sind und vom System auch als freie Parkplätze erkannt werden.

Anforderungen	Beschreibung	Gewichtung
Anteil an True-Positive -Parkplätzen	Beschreibt, wie viele True-Positive-Parkplätze Einträge an die Datenbank übertragen werden	5
Genauigkeit der Parkplatzerfassung	Beschreibt die Wahrscheinlichkeit, mit der ein freier Parkplatz frei ist. Zusätzlich hat der Ansatz *Parkplatz selbst erfassen und abspeichern* noch die Anforderungen an die Genauigkeit der Erfassung der Parkplatzmaße	4
Latenz der Auswertung	Wie lange soll die maximale Zeit der Auswertung betragen	3
Maximale Geschwindigkeit für die Auswertung	Wie schnell darf das Fahrzeug während der Auswertung fahren	2

Tabelle 3.1: Die Anforderungen an die Ansätze mit der Gewichtung der Anforderungen

In den Tabellen 3.2 und 3.3 sind zu den jeweiligen Anforderungen für jedes Konzept eine Gewichtung und eine Bewertung vorgenommen worden.

Anforderungen	Gewichtung	Bewertung Ansatz I	Punkte Ansatz I
Anteil an True-Positive-Parkplätzen	5	2	10
Genauigkeit der Parkplatzerfassung	4	3	12
Latenz der Auswertung	3	3	9
Maximale Geschwindigkeit für die Auswertung	2	3	6

Tabelle 3.2: Bewertung mit Gewichtung für das Konzept *Parkplatz selbst erfassen und abspeichern*

Anforderungen	Gewichtung	Bewertung Ansatz II	Punkte Ansatz II
Anteil an True-Positive-Parkplätzen	5	4	20
Genauigkeit der Parkplatzerfassung	4	4	16
Latenz der Auswertung	3	4	12
Maximale Geschwindigkeit für die Auswertung	2	3	6

Tabelle 3.3: Bewertung mit Gewichtung für das Konzept *Parkplatz aus Datenbank überprüfen und Status aktualisieren*

Nachfolgend ist die Erklärung für die Bewertung aufgelistet.

- Anteil an True-Positive-Parkplätzen: Beim Ansatz *Parkplatz aus Datenbank überprüfen und Status aktualisieren* sind die möglichen Parkplätze schon mit den richtigen Abmaßen in der Datenbank vorhanden. Deswegen muss nur die Informationen ob diese „Frei“ sind richtig erfasst werden. Beim Ansatz *Parkplatz selbst erfassen und abspeichern* allerdings muss zusätzlich noch erst mal erfasst werden, ob die freie Fläche überhaupt ein Parkplatz ist, da müssen Faktoren wie Feuerwehr- und Garagenausfahrten, Grundstücksauffahrten, sowie als Parkbuchten missverstandene Bushaltestellen und temporäre wie dauerhafte Parkverbote beachtet werden. Daher wird der Ansatz *Parkplatz aus Datenbank überprüfen und Status aktualisieren* prozentual mehr True-Positive Einträge haben als der Ansatz *Parkplatz selbst erfassen und abspeichern*

- Die beiden Ansätze unterscheiden unterscheiden sich nicht in der Erfassung der Wahrscheinlichkeit der freien Parkplätze, wie viel der Fläche des freien Parkplatzes frei ist. Der Ansatz *Parkplatz aus Datenbank überprüfen und Status aktualisieren* hat die zusätzliche Anforderung, die Parkplatzmaße richtig zu erfassen. Dies ist eine zusätzliche Fehlerquelle, weswegen der Ansatz *Parkplatz aus Datenbank überprüfen und Status aktualisieren* schlechter abschneidet

- Da beim Ansatz *Parkplatz aus Datenbank überprüfen und Status aktualisieren* mehr Daten ausgewertet werden müssen als beim Ansatz *Parkplatz aus Datenbank überprüfen und Status aktualisieren*, wird der Ansatz *Parkplatz aus Datenbank überprüfen und Status aktualisieren* mehr Zeit für die Auswertung brauchen und damit eine höhere Latenz aufweisen

- Erst bei sehr hohen Latenzen wird die Geschwindigkeit oder bei sehr hohen Geschwindigkeiten die Latenz einen Einfluss haben

Die Auswertung der Bewertung der beiden Ansätze ergibt folgende durchschnittliche Punkte:

- Der Ansatz *Parkplatz selbst erfassen und abspeichern* hat durchschnittlich 9,25 Punkte

- Der Ansatz *Parkplatz aus Datenbank überprüfen und Status aktualisieren* hat durchschnittlich 13,5 Punkte

Damit wird in dieser Arbeit der Ansatz *Parkplatz aus Datenbank überprüfen und Status aktualisieren* umgesetzt.
Die Tablle 3.4 umfasst die Anforderungen an das System für die Auswertung und das Abspeichern der freien Parkplätze. Hierbei ist der Anteil der True-Positive-Parkplätze, die Genauigkeit der Parkplatzerfassung, die Latenz der Auswertung und die maximale Geschwindigkeit, die das Fahrzeug während der Auswertung fahren darf, aufgelistet.

Anforderungen	Beschreibung	Quantifizierung
Anteil an True-Positive-Parkplätzen	Wie hoch soll der Anteil der True-Positive erkannten Parkplätze sein	70%
Genauigkeit der Parkplatzerfassung	Wie sicher ist das System, dass die freie Fläche frei ist	Durchschnittlich 70% frei bei freien Parkplätzen
Latenz der Auswertung	Wie groß soll die Latenz bei der Auswertung sein	500 ms
Maximale Geschwindigkeit für die Auswertung	Wie schnell darf das Fahrzeug maximale während der Auswertung fahren	Muss Anforderung: - 30 km/h Kann Anforderung: - 50 km/h

Tabelle 3.4: Anforderungen an den Systementwurf

4 Konzeptentwurf

Dieses Kapitel befasst sich mit dem Konzeptentwurf, wie der Use Case und die Anforderungen des vorherigen Kapitels, erfüllt werden können . Dies umfasst die Systeme auf denen die Umsetzungen laufen soll, den Entwurf der Komponenten die für die *Local Dynamic Map* benötigt werden und der Entwurf zur Erfassung des Status der Parkplätze.

4.1 Systemübersicht

Die Umsetzung des Use Cases dieses Projekts kann in zwei verschiedene Bestandteile aufgeteilt. Der eine Bestandteil ist, die Umsetzung der *Local Dynamic Map*, in der der Endbenutzer die Parkplätze abrufen kann. Der andere Bestandteil ist die Erfassung und Auswertung der Parkplätze. Beide Bestandteile werden in einer eigenen virtuellen Maschine ausgeführt, welche eine Verbindung untereinander haben sollen, über welche die beiden kommunizieren können. In der Abbildung 4.1 ist der Systemaufbau der beiden virtuellen Maschinen, sowie deren Datenaustausch dargestellt.

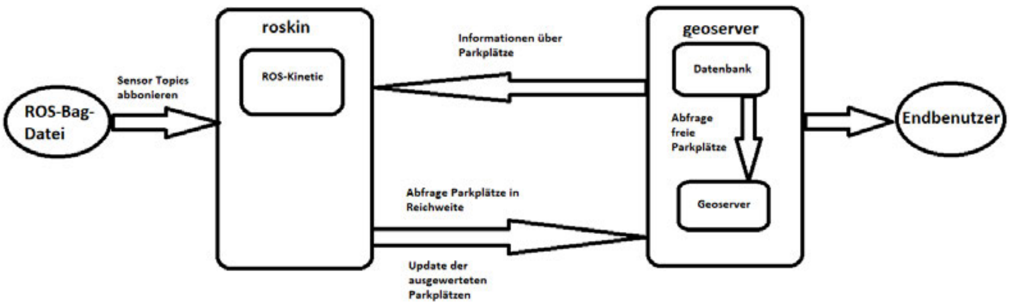

Abbildung 4.1: Systemaufbau und Datenaustausch der beiden virtuellen Maschinen

4.2 Entwurf der Local Dynamic Map Komponenten

In diesem Unterkapitel wird darauf eingegangen, wie die *Local Dynamic Map* in ihre vier unterschiedlichen Punkten, der Datenbank, des Datenbank Management System, der Schnittstelle zur Datenbank und der Anwendung, wofür die Daten der Datenbank verwendet werden, umgestzt werden soll.

4.2.1 Datenbank

Der wichtigste Aspekt einer *Local Dynamic Map* ist die Datenbank mit den enthaltenen Daten. In dieser Arbeit sollen die gemessen Parkplätze manuell in die Datenbank eingetragen werden, sodass die Abmaße und die Position des Parkplatzes jedem Parkplatz zugeordnet werden können. Zudem soll die Datenbank die Uhrzeit mit den Status und einer Wahrscheinlichkeit, die angibt wie wahrscheinlich dieser Parkplatz frei ist, von dem Messsystem automatisiert übertragen bekommen.
Daraus ergibt sich dass die Datenbank folgende Einträge, in der Tabelle 4.1 aufgeführt, enthalten.

Einträge in der Datenbank	Datentyp
Eine eindeutige Parkplatzidentifikation	char[20]
Die GPS-Position des Parkplatzes/ Die GPS-Positionen der Ecken des Parkplatzes	float
Die Länge Parkplatzes	float
Die Breite Parkplatzes	float
Die Uhrzeit der Statusänderung des Parkplatzes	time
Den Status „Frei“ oder „Belegt“ des Parkplatzes	char[20]
Die Konfidenz des richtigen Eintrags des Status	float
Geographischer Ort des Parkplatzes	point

Tabelle 4.1: Datenbankeinträge mit Datentypen

Folgende dieser Einträge werden im Voraus oder bei weiteren Eintragung der gemessen Parkplätze in die Datenbank eingetragen, dies ist in der Tabelle 4.2 dargestellt.
Die folgenden Einträge werden durch die Vermessung des Status der Parkplätze dynamisch angepasst, dies ist in der Tabelle 4.3 dargestellt.

Einträge in der Datenbank	Datentyp
Eine eindeutige Parkplatzidentifikation	char[20]
Die GPS-Position des Parkplatzes/ Die GPS-Positionen der Ecken des Parkplatzes	float
Die Länge Parkplatzes	float
Die Breite Parkplatzes	float
Geographischer Ort des Parkplatzes	point

Tabelle 4.2: Datenbankeinträge mit Datentypen, die für das manuelle Einträgen benötigt werden

Einträge in der Datenbank	Datentyp
Die Uhrzeit der Statusänderung des Parkplatzes	time
Den Status „Frei“ oder „Belegt“ des Parkplatzes	char[20]
Die Konfidenz des richtigen Eintrags des Status	float

Tabelle 4.3: Datenbankeinträge mit Datentypen, die für das Auswerten benötigt werden

4.2.2 Struktur der Datenbank

Die Struktur der Datenbank leitet sich aus den benötigten Daten, welche für die *Local Dynamic Map* vorhanden sein soll, ab. Dafür hat jede Zeile in der Datenbank diese Daten als einzelne Spalteneinträge. Jede neue Spalte die angelegt wird, stellt dabei einen neuen Parkplatz für die Datenbank dar, in welcher alle benötigten Daten spezifisch für den neu angelegten Parkplatz eingetragen werden müssen.
Die Konsistenz der Struktur der Datenbank ist hierbei wichtig, um bei der Verarbeitung und Nutzung der Daten zu gewährleisten, dass die richtigen Dateneinträge genutzt werden.

4.2.3 Schnittstelle zur Datenbank

Um die Daten aus der virtuelle Maschine, welche für die Auswertung der Parkplätze zuständig ist, zu der virtuellen Maschine, welche die Datenbank beinhaltet und zur Datenbank direkt zu senden und umgekehrt, ist es nötig, eine Verbindung zwischen den beiden virtuellen Maschinen herzustellen und eine Remote-Verbindung zur Datenbank umzusetzen. Diese Verbindung sorgt dafür, dass das Programm, welches die Parkplätze auswerten soll, die benötigten Informationen von der Datenbank bekommt und die Daten der Auswertung wieder an ebenjene zurückgeben kann.

4.2.4 Darstellung der Parkplätze auf einer Webkarte

Aus den Use Case *Parkplätze anzeigen* leitet sich der Anwendungsfall der *Local Dynamic Map* für dieses Projekt her. Diese soll es dem Endbenutzer ermöglichen, dass dieser aus den folgenden beiden Fällen, der nächstgelegene freien Parkplatz und alle freien Parkplätze innerhalb eines Radius, durch eine Eingabe entscheiden kann. Dafür muss der Endbenutzer die Möglichkeit haben, den gewünschten Standort, welche der eigene Standort oder der Zielstandort des Endbenutzers sein kann, sowie den Radius, falls die Option von Endbenutzer ausgewählt wird, bei dem alle freien Parkplätze innerhalb eines Radius angezeigt werden, eingeben zu können.
Nachdem der Endbenutzer die benötigten Daten eingetragen hat, die Auswahl getroffen und eingegeben hat, soll der Server die Daten aus der Datenbank abfragen und die gewünschten freien Parkplätze auf der Webkarte anzeigen lassen.

4.3 Entwurf der Komponenten zur Erfassung und Auswertung von Parkplätzen

Dieses Unterkapitel befasst sich mit dem Entwurf für die Erfassung und Auswertung von Parkplätze. Dabei wird zuerst die Erfassung der Parkplätze in der Datenbank betrachtet, danach werden die benötigten Daten zur Auswertung des Status der Parkplätze aufgeführt. Unter dem Punkt *Auswertung der Parkplätze mittels LiDAR* wird auf den Messradius, in welcher die Auswertung stattfinden soll, die Abfrage der Datenbank bezüglich der im Messradius befindlichen Parkplätze, das Speichern und Synchronisieren der Messdaten und das Vorgehen zur Auswertung, welcher Parkplatz „Frei“ ist, eingegangen.

4.3.1 Erfassung der Parkplätze

Durch die, für dieses Projekt getroffene Auswahl des Konzeptes, ist es notwendig die Parkplätze, dessen Status überprüft werden soll und den Endbenutzer als „Frei“ auf der

Webkarte angezeigt werden soll, vorher manuell zu vermessen und dann in die Datenbank einzutragen. Folgende Daten sollen dabei erfasst und manuell in die Datenbank eingetragen werden:

- GNSS-Position des Zentrums des Parkplatzes
- Länge des Parkplatzes
- Breite des Parkplatzes
- Orientierung des Parkplatzes

Bei der Eintragung der Parkplätze in die Datenbank, muss jeder der manuell eingetragenen Parkplätze eine eindeutige Identifikation bekommen. Die restlichen Informationen werden automatisch während der Auswertung für den jeweiligen Parkplatz ergänzt.

4.3.2 Benötigte Daten zur Auswertung

Die Auswertung benötigt sowohl Daten, welche wie im vorherigen Punkt erläutert, manuell eingetragene Informationen für den Parkplatz sind, als auch die von den Sensoren gemessenen Daten. Die Daten die aus der Datenbank abgerufen werden sollen, sind:

- Eindeutige Parkplatzidentifikation
- GNSS-Position des Zentrums des Parkplatzes

Die eindeutige Parkplatzidentifiaktion wird benötigt, um während der Auswertung die Informationen den jeweiligen Parkplatz zuordnen und diese dann für den jeweiligen Parkplatz in der Datenbank überschreiben lassen zu können.
Von den Sensoren werden für die Auswertung folgende Daten, wie in der Tabelle 4.4 angegeben, benötigt. Der System Apogee-D veröffentlicht die Daten der GNSS-Positionsbestimmung mit einer Frequenz von 5 Hz, der Velodyne 32-HDL-E mit einer Frequenz von 10 Hz. Es ist festzustellen, dass im Programmablauf eine größere Differenz als 100 ms auftritt. Deswegen sind die Zeitstempel der LiDAR PointCloud2 Datenmessung und der GNSS-Positionsbestimmung sind im weiteren Verlauf wichtig, um eine möglichst kleine zeitliche Differenz zu gewährleisten, damit sich die GNSS bestimmte Position des Fahrzeuges nicht zu sehr von der Position des Fahrzeuges während der LiDAR PointCloud2 Datenmessung unterscheidet.

Sensordaten zur Auswertung	ROS-Topics Name
LiDAR PointCloud2 Daten	velodyne_points
Zeitstempel der LiDAR PointCloud2 Datenmessung	velodyne_points
GNSS-Position des Fahrzeugs	gps_pos
Zeitstempel der GNSS-Positionsmessung des Fahrzeugs	gps_pos
Orientierung des Fahrzeuges	gps_hdt

Tabelle 4.4: Die ROS-Topics-Namen zu den dazugehörigen Sensordaten, welche zur Auswertung benötigt werden

4.3.3 Programmablauf

Nachfolgend wird der Programmablauf des Programms zur Auswertung der Parkplätze erläutert. Die Daten zur Auswertung der Parkplätze werden in einer Klasse gespeichert, die bei jedem Durchlauf neu instanziiert wird, um dann neue Messdaten zu speichern. Nach der Speicherung der Messung und Abfrage an die Datenbank erfolgt die Überprüfung der zeitlichen Differenz, falls Parkplätze im Messradius vorhanden sind. Wenn kein Parkplatz sich innerhalb des Messradius befindet, dann wird die Zeit 2,448 Sekunden gewartet, damit das Fahrzeug die Möglichkeit hat sich fort zubewegen, um außerhalb des vorherigen Messradius beim Start einer neuen Auswertung zu beginnen. Für den Fall, dass sich Parkplätze innerhalb des Messradius befinden, aber die zeitliche Differenz überschritten wird, wird sofort eine neue Messung begonnen.
Das Auslassen der Wartezeit zwischen den Auswertungen wird dazu führen, dass viele Parkplätze in sehr kurzer Zeit öfters ausgewertet werden. Diese Auswertungen enthalten durch die öfteren Auswertungen in sehr kleinen zeitlichen Abständen viele redundante Informationen, da sich der Status des Parkplatzes in den aller meisten Fällen nicht innerhalb von Millisekunden und Sekunden verändert.
Wenn sich Parkplätze innerhalb des Messradius befinden und die zeitliche Differenz nicht überschritten wird, wird die kartesische Position des Mittelpunkt des Parkplatzes relativ zum Fahrzeug berechnet und der Konfidenzwert ermittelt, für jeden Parkplatz, welchen die Datenbank zurück gibt ermittelt. Diese Daten werden im Anschluss an die Datenbank zurück übergeben und am Ende der Auswertung wird die Zeit 2,448 Sekunden bis zum Starten einer neuen Messung gewartet.

4.3.4 Auswertung der Parkplätze mittels LiDAR

Für die Auswertung der Parkplätze wird zuerst ein Messbereich definiert, in welchen die Parkplätze ausgewertet werden. Die Datenbank wird auf Parkplätze innerhalb dieses Messradius abgefragt, welche die benötigten Daten der Parkplätze zum Auswerten des Status, falls Parkplätze im Messradius vorhanden sind, an das Programm sendet. Im gleichen Programmablauf werden die Sensordaten gespeichert und für die Parkplätze im Messradius, zur jeweiligen Auswertung genutzt. Die Informationen der Auswertung wird nach der Auswertung an die Datenbank übergeben und das Programm wartet, bis der Messradius verlassen wurde, auf eine neue Messung.
Dazu wird in den Unterabschnitten zuerst auf den Messradius und die zu wartende Zeit, um den Messradius zu verlassen eingegangen, nachfolgend auf die Abfrage der Parkplätze in Reichweite an die Datenbank, das Speichern und Synchronisieren der gemessenen Daten, welche zur Auswertung benötigt werden, sowie die Auswertung des Status der Parkplätze und die Informationsübermittlung dieser Auswertung an die Datenbank

Messbereich

Für die Erfassung des Status der Parkplätze, ob diese den Status „Frei" oder „Belegt" aufweisen, werden verschiedene Aspekte betrachtet. Der erste dieser Aspekte ist, dass die Parkplätze aus der Datenbank zur Überprüfung abgerufen werden, welche in einem sinnvollen Messbereich der Sensoren liegt, zum Zeitpunkt der Messung liegen. Der sinnvolle Messbereich ist der Messbereich, bei welchen die Umgebung durch LiDAR PointCLoud2 Punkte noch erkennbar ist. Dadurch sind genügend LiDAR PointCloud2 Punkte für die Auswertung in der Punktwolke vorhanden. Der sinnvolle Messbereich, stellt den Messbereich dar, bei den der LiDAR-Sensor PointCloud2-Daten aufnimmt, bei der noch die Objekte der Umgebung erkennbar sind. Da mit dem Abstand zum Sensor, die Ringe um Fahrzeug herum, einen immer größer werdenden Abstand haben, sind weiter entfernten Objekten nicht mehr erkennbar. Aus dem Bildern 4.2 und 4.3 geht hervor, dass der Abstand zwischen dem Ring vor dem Fahrzeug und dem Ring hinter dem Fahrzeug, bei welchen die Fahrzeuge am Straßenrand noch gut als solche erkennbar sind, bei ca 34 Metern liegt. Weil dies den Durchmesser darstellt, wird er Radius zur Abfrage auf 17 Metern gelegt.
Aus dieser Distanz, in welchem Radius gemessen wird, kann eine Zeit bestimmt werden, die das Fahrzeug weiterfahren soll, um eine neue Auswertung zu starten und dabei nicht

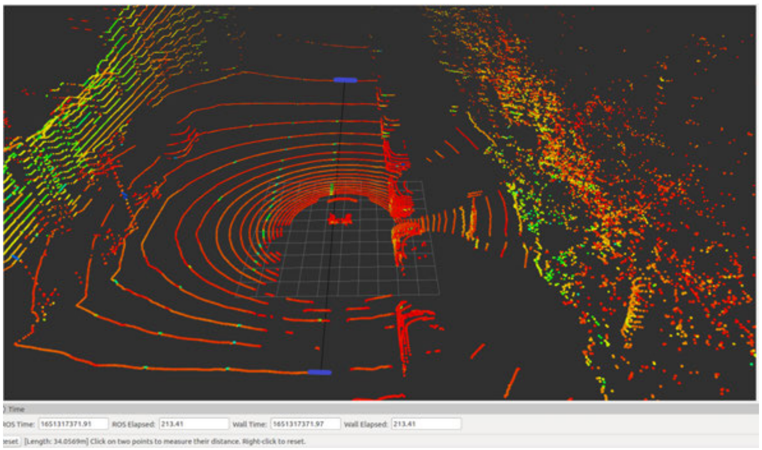

Abbildung 4.2: Der Abstand, in welchen in den LiDAR-Punkten, die Fahrzeuge am Straßenrand noch gut erkennbar ist

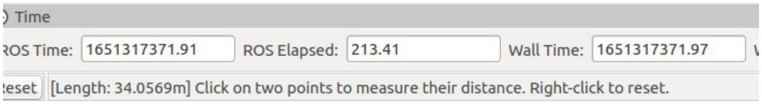

Abbildung 4.3: Der Abstand, in welchen in den LiDAR-Punkten, die Fahrzeuge am Straßenrand noch gut erkennbar ist, vergrößert dargestellt

nochmal die gleichen Parkplätze zu vermessen. Dafür werden die typischen Straßenverkehrsgeschwindigkeiten 30 km/h und 50 km/h in m/s umgerechnet und dann ausgerechnet, wie lange gebraucht wird um einen Messdurchmesser zurückzulegen. Für 50 km/h folgt

$$\frac{50000}{3600}\frac{m}{s} = \frac{125}{9}\frac{m}{s} = 13,9\ \frac{m}{s} \tag{4.1}$$

und für 30 km/h folgt

$$\frac{30000m}{3600s} = \frac{25}{3}\frac{m}{s} = 8,3\ \frac{m}{s}. \tag{4.2}$$

Die beiden Ergebnisse werden zur Zeitberechnung benutzt. Für 50 km/h folgt

$$\frac{34}{13,\overline{8}}\frac{m}{s} = 2,448\ s \tag{4.3}$$

und für 30 km/h folgt

$$\frac{34}{8,\overline{3}}\frac{m}{s} = 4,08\ s. \tag{4.4}$$

Es wird die Zeit für die Geschwindigkeit von 50 km/h als Referenzwert genommen, da bei größeren Zeiten eine Lücke zwischen dem Vermessen der Parkplätze entsteht, in welcher die Parkplätze nicht auf ihren Status überprüft werden, da diese dann nicht von der Datenbank abgefragt werden können, weil sie sich außerhalb des Messradius befinden. Die zeitliche Differenz von 2,448 Sekunden wartet das Programm nach der erfolgten Auswertung der Parkplätze bis zum Start der nächsten Auswertung.

Abfage an die Datenbank

Zum Start des Messvorgangs sollen alle in der Datenbank vorhandenen Parkplätze, welche sich innerhalb des Radius befinden, abgefragt werden. Falls keine Parkplätze innerhalb dieses Radius um das Fahrzeug herum befinden, braucht keine Auswertung der Messdaten vorgenommen zu werden. Es wir die Zeit 2,448 s von der Software-Seite aus gewartet, bis die nächste Abfrage an die Datenbank stattfindet.
Wenn die Datenbank vorhandene Parkplätze innerhalb des Messradius zurückgibt, werden folgende Daten zu jeden Parkplatz der innerhalb des Messradius liegt, zusätzlich von der Datenbank zurückgegeben:

- Die eindeutige Parkplatzidentifikation
- Die GNSS-Position des Zentrum des Parkplatzes

Speichern und synchronisieren der benötigten Messdaten

Bei dem Speichern der Daten, die durch die Messung der Sensoren vorhanden sind, ist es wichtig, die Messungen aufeinander abzustimmen, um die Differenz der Zeit und somit die Differenz der Positionen klein zu halten. Da diese nicht zeitgleich messen, ist es wichtig zu betrachten, welche zeitliche Distanz zwischen den beiden Messungen akzeptabel ist.
Hierfür werden die maximal möglichen Fehler der Abstände der GNSS-Positionen in Metern zwischen den beiden Messungen von der GNSS-Position und der nächstgelegenen LiDAR-Messung für die Geschwindigkeiten 30 km/h und 50 km/h berechnet, welche zunächst in das Äquivalent in Meter pro Sekunde umgerechnet wird. Diese Ergebnisse werden aus der Formel 4.2 und 4.1 übernommen. Für eine Abweichung von einem Meter lässt sich die maximale zeitliche Differenz ermitteln, die zwischen den beiden Messungen auftauchen darf, um unterhalb der Abweichung liegen zu können. Für die Geschwindigkeit von 30 km/h ergibt sich

$$1\ m = 8,3\ \frac{m}{s} \cdot Y_1 \tag{4.5}$$

umgeformt zu

$$Y_1 = \frac{1}{8,3}\ s = 0,12\ s \tag{4.6}$$

und für die Geschwindigkeit von 50 km/h ergibt sich

$$1\ m = 13,9\ \frac{m}{s} \cdot Y_2 \tag{4.7}$$

umgeformt zu

$$Y_2 = \frac{1}{13,9}\ s = 0,072\ s. \tag{4.8}$$

Diese zeitliche Differenz, muss bei der Programmierung zur Sicherstellung dieser maximalen Abweichung der Messung beachtet werden.

Konfidenzwert

Der Konfidenzwert gibt den Wert der Sicherheit an, mit welchen das System den Status des Parkplatzes erfasst hat. Dieser bildet sich aus dem Quotient der Punkte, dessen Z-Wert innerhalb des Offsetsintervalls liegen, zu allen Punkten, die innerhalb des Radius liegen und überprüft werden:

$$Konfidenzwert = \frac{\text{Punkte innerhalb des Offsetwerte Intervalls}}{\text{Alle überprüften Punkte innerhalb des Radius}} \tag{4.9}$$

Wenn dieser Faktor ausgerechnet wurde, wird anhand der Werte entschieden, ob der Status des Parkplatzes „Frei“ oder „Belegt“ ist.

- Konfidenzwert $\geq$ 0,5: Der Parkplatz ist frei
- Konfindenzwert $<$ 0,5: Der Parkplatz ist belegt

Diese Berechnung wird für jeden Parkplatz innerhalb des Messradius durchgeführt.

Auswertung welcher Parkplatz „Frei‘ ist

Wenn der Fall eintritt, dass die zeitliche Differenz unter der akzeptablen zeitliche Differenz aus den Formeln 4.6 und 4.8 zwischen den Erfassung der Sensordaten vorliegt, diese Daten zwischengespeichert sind, die Abfrage an die Datenbank Parkplätze innerhalb des Messradius zurück gibt, kann die Auswertung der Parkplätze innerhalb des Messradius begonnen werden.

Die aufgenommen LiDAR-Daten sollen auf der Fläche des Parkplatzes, in einem Kreis um den Mittelpunkt, die Höhe, also den Z-Wert, der Punkte überprüfen und daraus eine Wahrscheinlichkeit, den Konfidenzwert, ableiten zu können, ob dieser Parkplatz „Frei' oder „Belegt' ist. Aus der Abfrage der Datenbank sind die Longitude und Latitude Werte der Parkplätze bekannt, die LiDAR PointCloud2 Daten sind allerdings in einem kartesischen Koordinatensystem geordnet. Daher ist es notwendig, den Mittelpunkt des Parkplatzes in eine kartesische Koordinate, relativ zum Fahrzeug umzuwandeln. Dies lässt sich wie folgt umsetzen, der Mittelpunkt der Lidarmessung, ist der Ausgangspunkt (0,0) in X- und Y-Koordinaten des kartesischen Koordinatensystems. Diese Position ist durch die GNSS-Messung auch als Longitude und Latitude Wert vorhanden. Aus diesem geographischen Punkt des Fahrzeuges und dem geographischen Punkt des Mittelpunkt des Parkplatzes, kann die Distanz und die Peilung ermittelt werden, mit welcher von dem geographischen Punkt des Fahrzeuges zum geographischen Mittelpunktes des Parkplatzes gelangt werden kann. Die Peilung gibt dabei die Richtung in Winkel an und die Distanz die Strecke in die sich bewegt muss, um vom Ausgangspunkt zum Zielpunkt zu gelangen.

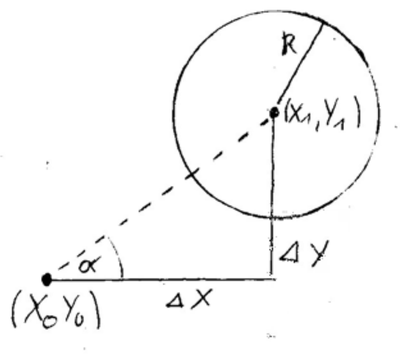

Abbildung 4.4: Skizze für die Bestimmung der Parkplatzposition und dem Messradius

Dies in bildlich in der Abbildung 4.4 dargestellt. Der Punkt (X_0,Y_0) ist die Position des Fahrzeuges, der Punkt ist die Position des Parkplatzes. Die beiden Werte Δx und Δy ist die Distanz von der Position des Fahrzeuges zu der Position des Parkplatzes im kartesischen Koordinatensystem. Der Winkel α ist der Peilungswinkel. Der Wert R stellt den Radius für den Messradius dar. Für alle diese Punkte innerhalb des Radius von einem

Meter, wird jeder dieser Punkte überprüft ob sich die Höhe, der Z-Wert des Messpunktes, in dem Intervall befindet, welcher die Bodenhöhe mit einem zusätzlichen Fehler ist. Der Radius von einem Meter wird gewählt, da die Parkplätze in der Breite mindestens 2 Meter betragen, welches dem Durchmesser des Kreises bei diesem Radius entspricht.
Um die Distanz zweier geographischer Punkte zu ermitteln, kann die Haversine-Formel genutzt werden. Die Distanz zweier Punkte auf einem Kreis ist dabei

$$D = R \cdot \theta \Leftrightarrow \theta = \frac{D}{R}, \tag{4.10}$$

hierbei ist R der Radius der Kugel und θ der Winkel zwischen A und B im Bogenmaß. Die Haversine-Formel kann auch als

$$\mathrm{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right) \tag{4.11}$$

ausgedrückt werden. Hierbei steht hav (θ) für die Haversine-Formel mit den Winkel zwischen den beiden Punkten A und B. Durch Umformungen lässt sich folgender Ausdruck für die Haversine-Formel erhalten:

$$\mathrm{hav}\left(\theta\right) = \mathrm{hav}\left(\phi_2 - \phi_1\right) + \cos\left(\phi_1\right)\cos\left(\phi_2\right)\mathrm{hav}\left(\lambda_2 - \lambda_1\right) \tag{4.12}$$

Hierbei sind ϕ die Latitude Werte und λ die Longitude Werte beider Punkte A und B. die Mit Hilfe der Formel 4.11 kann die Haversine-Formel [4] ohne die eigene Abhängigkeit aufgestellt werden:

$$\mathrm{hav}\left(\theta\right) = \left(\sqrt{sin^2\left(\frac{\phi_1 - \phi_2}{2}\right) + cos(\phi_1) \cdot cos(\phi_2) \cdot sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{4.13}$$

Nach der Umstellung der Formel 4.10 nach D folgt

$$D = R\,\mathrm{archav}(\mathrm{hav}\left(\theta\right)) = 2R\arcsin\left(\sqrt{\mathrm{hav}\left(\theta\right)}\right). \tag{4.14}$$

Aus den beiden Ergebnissen aus den Formeln 4.13 und 4.14 kann die Distanz zwischen zweier Punkte A und B wie folgt errechnet werden:

$$D = 2 \cdot R \cdot \sqrt{arcsin(\mathrm{hav}\left(\theta\right))}. \tag{4.15}$$

Für die Berechnung der Peilung β wird der Arcustanges benötigt, welcher 2 reelle Zahlen als Argumente nehmen kann. Diese beiden Werte sind die Punkte (X_0, Y_0) und (X_1, Y_1).

Daraus folgt für den Winkel β

$$\beta = arctan2((X_0, Y_0), (X_1, Y_2)). \tag{4.16}$$

Im den Nachfolgenden zwei Gleichungen sind die Latitude Werte der beiden Punkte, zwischen denen die Peilung ermittelt werden soll, mit ϕ und die Longitude Werte mit λ gekennzeichnet.

$$X = cos(\phi_2) \cdot sin(\lambda_2 - \lambda_1) \tag{4.17}$$

$$Y = cos(\phi_1) \cdot sin(\phi_2) \check{} sin(\phi_2) \cdot cos(\phi_2) \cdot cos(\lambda_2 - \lambda_1) \tag{4.18}$$

Mit diesen beiden Werten, die Distanz und die Peilung, für jeden Parkplatz, lässt sich die kartesische Position des Mittelpunktes des Parkplatzes,relativ zur Position des Fahrzeuges, berechnen:

$$X_1 = Distanz \ \cdot sin(\beta) \tag{4.19}$$

$$Y_1 = Distanz \ \cdot cos(\beta) \tag{4.20}$$

Aus dieser kartesischen Position, kann überprüft werden, ob sich die Messpunkte, die durch die LiDAR PointCloud2 Daten, erfasst wurden innerhalb des Radius um den Mittelpunkt des Parkplatzes befindet.
Da sich der Velodyne-Sensor auf einer Höhe von 1,88 Meter, siehe Abbildung 2.9 befindet, wird ein Offset-Wert von dem Z-Wert abgezogen, wodurch dieser Punkt auf den Boden hinunter gesetzt wird. Zu dem wird eine Unsicherheit von 10% dem Messwert gebilligt, da sich die Straße und der Parkplatz nicht immer auf der exakt gleicher Höhe befinden müssen, sowie die Straßen auch eine gewisse Neigung haben, wodurch Parkplätze, die weiter entfernt sind, nicht mehr auf der gleichen Höhe wie die Straße, auf dem das Fahrzeug fährt, befinden müssen. In der Abbildung 4.5 ist dies exemplarisch skizziert. Die schwarze Linie stellt hiebrei die Straße dar, die blaue Linie den Boden, bei der Messung durch den Velodyne-Sensor. Es ist zu erkennen, dass die tatsächliche Straße, je weiter weg vom Fahrzeug, desto mehr Abweichung von der gemessenen Bodenlinie um das Fahrzeug herum aufweist. Die Höhe des angebrachten Sensor wird mit einer Abweichung von $+10\%$ und -10% versehen. Daraus folgt wenn der Z-Wert der Punkte innerhalb dieses Intervall liegt, gilt

$$-2,068 \ Meter \leq Z - Wert \ des \ Datenpunktes \ in \ Meter \leq -1,692 \ Meter \tag{4.21}$$

Daraus folgt ein Offset-Intervall von

$$Offsetwerte\ Intervall = [-1.692, -2.068]. \tag{4.22}$$

wird dieser Punkt zu der Menge der Punkte die sich auf dem Boden befinden, hinzugefügt.

Abbildung 4.5: Skizze zur Erklärung des Offsetwerts. Die schwarze Linie ist die Straße, die blaue der Boden der Messung

Mit der Anzahl der Punkte die sich innenrhalb des Radius um den Mittelpunkt des Parkplatzes und alle Punkte deren Z-Wert sich innerhalb des Offsetintervalls befindet, kann der Konfidenzwert ermittelt werden.

Update der Datenbank

Nach der Berechnung des Konfidenzwert werden folgende Information für die dazugehörige eindeutige Parkplatzidentifikation an die Datenbank übergeben und überschrieben:

- Zeitpunkt der Messung
- Status des Parkplatzes
- Konfidenzwert der Messung

4.3.5 Abstand Fahrzeug zum ersten gemessenen Ring

Während der Entwurfsphase, ist aufgefallen, dass sich durch die Montage des Velodyne-Sensors auf dem Autodach, das Problem ergibt, dass der Sensor nicht alles rund ums Auto erfassen kann. Hiervon ist die Umgebung direkt um das Auto herum betroffen, durch den Winkel mit dem der Velodyne-Sensor die Umgebung vermisst. Dadurch, dass der Sensor nicht steil genug misst um von der Höhe auf dem Dach, welche 1,88 Meter

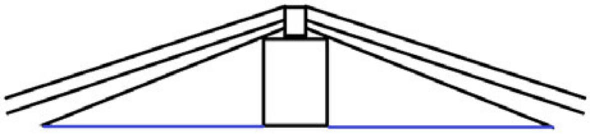

Abbildung 4.6: Skizze des Messabstand vom Fahrzeug zum ersten gemessenen LiDAR-Ring

beträgt, direkt den Boden neben dem Auto zu treffen, beziehungsweise bei einem steileren Winkel das Autodach des Teslas vermisst, anstatt den Boden um Auto herum, muss der Abstand zwischen den ersten gemessene Ring und dem Rand des Teslas mit betrachtet werden. Die Abbildung 4.6 mit der Skizze zeigt die Ansicht von vorne. Dabei sind die beiden blauen Linien die Abstände des ersten Rings zum Fahrzeug. Die Breite des Tesla Model S beträgt 1,964 Meter. Mit den aus den beiden Bildern 4.7 und 4.8 gemessenen Abstand innerhalb der Ringe lässt sich herleiten, ab welcher Distanz zum Auto der Boden vermessen wird:

$$\frac{6,14\ m - 1,964\ m}{2} = 2,088m \tag{4.23}$$

$$\frac{6,60\ m - 1,964\ m}{2} = 2,318m \tag{4.24}$$

Die beiden Bilder wurden aus einer der aufgenommenen ROS-Bag Dateien mittels RVIZ, ein Tool, welches die ROS-Topics einer ROS-Bag Datei darstellen kann, abgebildet und mit dem mitgelieferten Messtool ausgemessen. Es ist festzuhalten, dass der Veldonye-

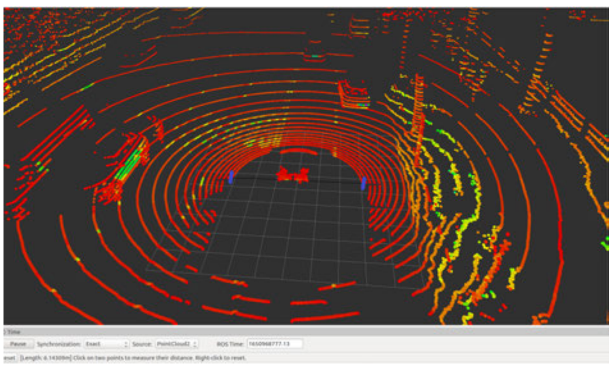

Abbildung 4.7: Gerade als Abstandmessung der LiDAR-Punkte im vorderen Fahrzeugbereich

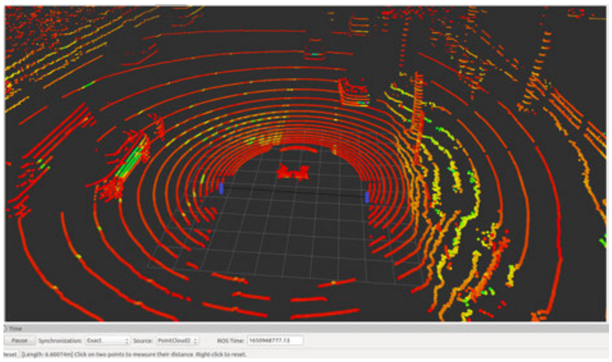

Abbildung 4.8: Gerade als Abstandmessung der LiDAR-Punkte im mittleren Fahrzeugbereich

Sensor mit der Montage auf dem Dach des Teslas, erst ab einen Abstand von ca. 2 Metern um den vorderen Bereich, auf der Höhe des Fahrers, den Boden vermisst und bei der ungefähren Mitte des Fahrzeug erst ab einem Abstand von ca. 2,30 Metern den Boden vermisst.

5 Implementierung

In diesem Kapitel werden die Entwürfe aus dem vorangegangenen Kapitel realisiert. Hier wird zuerst die Umsetzung der virtuellen Maschinen, danach die Umsetzung der *Local Dynamic Map* und zum Schluss die Auswertung der Parkplätze erläutert.

5.1 Virtuelle Maschinen

Für die beiden Bestandteile des Systementwurf wird jeweils eine virtuelle Maschine aufgesetzt.
Auf einer der beiden virtuellen Maschinen, auf welcher die Realisierung des Messvorgangs aus den aufgenommen Sensordaten und die Verarbeitung dieser Daten für die Kommunikation und Eintragung der Datenbank stattfindet, heißt *roskin*. Das Betriebssystem dieser virtuellen Maschine ist Ubuntu 16.04 [21].
Die zweite virtuelle Maschine, welche die Datenbank und den Geoserver zum Anzeigen der Parkplätze mit den dazugehörigen Informationen auf einer Webkarte umfasst heißt *geoserver*. Das Betriebssystem dieser virtuellen Maschine ist Ubuntu 18.04 [20]. Die IP-Adresse zu welcher kommuniziert werden kann, ist:

- IP-Adresse: 10.0.2.15

5.2 Realisierung der Local Dynamic Map Kompnenten

Zur Realisierung der Local Dynamic Map Komponenten wird im Nachfolgenden auf die Umsetzung der Datenbank, die Umsetzung der Datenbankstruktur und auf welche Art und Weise die Daten gespeichert werden, eingegangen. Zudem wird die Schnittstelle zur Datenbank, sowie die Darstellung der Parkplätze auf einer Webkarte, welche den Geoserver nutzt und die Daten aus der Datenbank anzeigt, erklärt.

5.2.1 Datenbank

Als Datenbank wird die PostgreSQL [12] mit der Erweiterung PostGIS [31] verwendet. Die Versionsnummer der PostgreSQL ist 14.2. Die Versionsnummer der PostGIS-Erweiterung ist 3.2. Die Datenbank wird auf der virtuellen Maschine mit dem Namen *geoserver* installiert.
Die Installation der benötigten Datenbank PostgreSQL und der Erweiterung PostGIS sind im Anhang erklärt [13]. Für diese Arbeit wird eine Datenbank mit dem Namen *parkinglotsDatabase* erstellt, mit der PostGIS Erweiterung versehen. In dieser Datenbank wird ein Table mit dem Namen *parkinglots* erstellt [32]. Die Anleitung der Erstellung ist im Anhang aufgeführt.

5.2.2 Datenbankstruktur

Um die Struktur der Datenbank umzusetzen, werden die benötigten Einträge, welche im Kapitel 4.2.1 aufgelistet sind, in Varibalennamen mit ihren entsprechenden Datentypen in die Datenbank eingetragen. Die Variablennamen mit ihren Datentypen sind in der nachfolgenden Tabelle 5.1 eingetragen.

Beschreibung	Name des Datenbankeintrages	Datentyp
Parkplatz-ID	ID-Nummer	VARCAHR(20)
Longitude des Parkplatzes	longitude	FLOAT
Latitude des Parkplatzes	latitude	FLOAT
Länge	height	FLOAT
Breite	width	FLOAT
Uhrzeit der Statusänderung	time_of_status	TIME
Status des Parkplatzes	Status	VARCHAR(20)
Konfidenz	Confidence	FLOAT
GNSS-Position des Parkplatzes als ein geographischer Punkt	location_geography	GEOGRAPHY(Point)

Tabelle 5.1: Die Struktur der Datenbankeintragun mit Namen und Datentyp

Zum Hinzufügen der Variablennamen mit ihren entsprechenden Datentypen als Spalten für die Einträge wird folgender Befehl benötigt:
ALTER TABLE parkinglots ADD [variablename] [datentyp];

Nach der Erstellung und ohne Einträge ist die Struktur der Datenbank wie in der Abbildung 5.1 dargestellt ist.

Abbildung 5.1: Struktur der Datenbank ohne Einträge

5.2.3 Schnittstelle zur Datenbank

Für die Kommunikation untereinander wird bei den beiden in den Netzwerkeinstellungen der Adapter 1 aktiviert und NAT-NETWORK ausgewählt. Die virtuelle Maschine, welche die Datenbank beinhaltet, die virtuelle Maschine mit dem Namen *geoserver*, bekommt zusätzlich den Adapter 2 aktiviert und wird auf den Host-Only Adapter eingestellt.
Um die Kommunikation zur Datenbank, von der Datenbank mit dem Namen *roskin*, welche nicht die Datenbank beinhaltet, zu ermöglichen, muss dies für die virtuelle Maschine erst eingestellt werden. Diese Verbindung läuft über das TCP, Transmission Control Protocol. Diese ist jedoch nicht standardmäßig aktiviert für die PostgreSQL. Aus diesem Grund müssen die beiden Dateien *postgresql.conf* und *pg_hba.conf* der PostgreSQL angepasst werden.
In der Datei *postgresql.conf* wird in der Zeile 59 die folgende Zeile einkommentiert und angepasst auf:
listen_addresses = ''*
Dadurch hat die PostgreSQL nun die Möglichkeit auf alle möglichen IP-Adressen, welche Anfragen schicken, zu reagieren. Mit der folgenden Zeile wird allen Nutzern ermöglicht, durch die Verbindung zur Datenbank und der Eingabe des Passworts, Veränderungen vorzunehmen [19]:
host all all 0.0.0.0/0 password

Eintragung von Daten

Die erfassten Parkplätze werden in die Datenbank durch den Befehl
INSERT INTO parkinglots VALUES([variablenname] = [variablenwert])
eingetragen. Dies wird vor der Auswertung für jeden Parkplatz, welcher in der Datenbank vorhanden sein soll, durchgeführt.

5.2.4 Darstellung der Parkplätze auf einer Webkarte

Dieses Unterkapitel umfasst den Geoserver, die Darstellung der freien Parkplätze auf einer Webkarte und die Abfrage an den Endbenutzer.

Geoserver

Der Geoserver wird auf der virtuellen Maschine mit dem Namen *geoserver* installiert. Für die Installation vom Geoserver wird zuerst die Java Version 8 oder 11 Envoirment (JRE) benötigt. Nach der Installation von JRE-8, welche von OpenJDK installiert werden kann, wird die aktuellste stabile Version von dem Geoserver gedownloaded und in den Ordner */usr/share/geoserver*, wie sie auch vom Geoserver empfohlen wird, verschoben. Um die Umgebungsvariablen für den Geoserver anzulegen wird
echo “export GEOSERVER_HOME=/usr/share/geoserver“ >> /.profile . /.profile
ausgeführt. Es sollte auch noch mit
sudo chown -R USER_NAME /usr/share/geoserver/
sichergestellt werden, dass der Nutzer Account der virtuellen Maschine auch die Admin-Rechte für den Gesoerver-Ordner besitzt. Innerhalb des Geosserver-Ordner befindet sich ein Ordner mit dem Namen *bin*, in welcher die Startup-Routine mit den Namen *startup.sh* hinterlegt ist. Diese muss gestartet werden für den Geoserver, danach kann mittels der Eingabe folgender URL
http://localhost:8080/geoserver
das Webadmin-Interface vom Geoserver aufgerufen werden [10].

Webkarte

Die Webkarte wird in der Datei *webmap.html* realisiert. Die Datei beinhaltet die Karte, die die Parkplätze anzeigen soll. Dies wird über OpenLayers von OpenStreetMap realisiert. Dieser Layer zeigt das Straßenverzeichnis an. Unterhalb der Webkarte sind zwei Buttons, der für den *nächstgelegenden Parkplatz* und den für die *Parkplätze im Radius*. Wenn der Endbenutzer seine Angaben getroffen hat und die Auswahl über die Buttons erfolgt ist, wird die jeweilige Funktion aufgerufen. Die Funktionen rufen die SQL-Viewparameter-Layer des Geoservers auf. Die SQL-Viewparameter-Layer sind Layer, die eine SQL-Anfrage an die Datenbank, welche im Storage angeben ist, senden. Dabei lassen

sich Parameter bei der Erstellung der SQL-Viewparameter festlegen, die bei der Ausführung eines Programms dynamisch erfasst und als Parameter für die Datenbank Abfrage genutzt werden können. Die SQL-Viewparameter lassen sich unter dem Punkt Storage zu den ausgewählten Storage, mit der verknüpften Datenbank, erstellen. Das Ergebnis dieser Abfrage wird über den jeweiligen Layer angezeigt. Diese Layer werden über den Straßenverzeichnis Layer gelegt. Für den *nächstgelegenden Parkplatz* wird folgende SQL-Anfrage ausgeführt
SELECT time_of_status, location_geography, height, width FROM parkinglots WHERE status='Frei' ORDER BY location_geography <-> ST_GeogFromText('POINT(%long% %lat%') LIMIT 1
Für den *Parkplätze im Radius* wird folgende SQL-Anfrage ausgeführt
SELECT time_of_status, location_geography, height, width FROM parkinglots WHERE status='Frei' AND ST_DistanceSphere(ST_MakePoint(longitude, latitude), ST_MakePoint(%longPos%, %latPos%)) <= %radius%
Die Variable *%long%* und die Variable *%longPos%* ist der Longitude Wert, die Variable *%lat%* und die Variable *%latPos%* ist der Latitude Wert und die Variable *%radius%* ist der Radius.
In der nachfolgenden Abbildungen 5.2 ist die Realisierung der eigenen Karte mit der den Eingabefeldern und Buttons dargestellt.

Abbildung 5.2: Realisierung der eigenen Karte

Abfrage des Use durch den Endbenutzer

Die Abfrage des Use Cases durch den Endbenutzer ist über die HTML-Datei *webmap.html*, welche die Anweisungen zur Darstellung der Webkarte beinhaltet, umgesetzt. Über zwei Buttons, mit dem Text *nächstgelegender Parkplatz* und *Parkplätze im Radius* beschriftet, kann der Endbenutzer auswählen, welchen der beiden Fälle angezeigt werden soll. Nach der Eingabe der benötigten Daten und dem Betätigen des Buttons werden die Parkplätze auf der Karte angezeigt. Der Ablauf ist in der Abbildung 5.3 dargestellt.

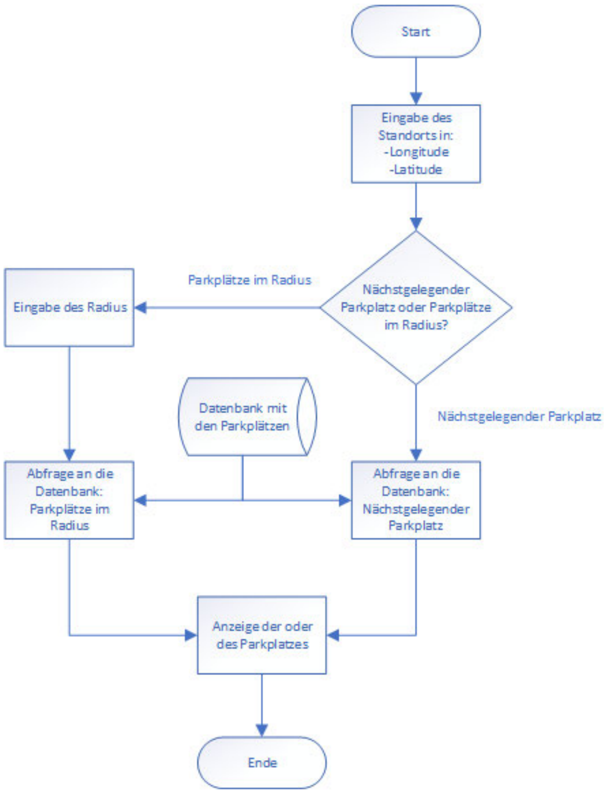

Abbildung 5.3: Flussdiagramm zur Endbenutzer Abfrage der freien Parkplätze

Es gibt drei Eingabefelder. Ein Eingabefeld fragt den vom Endbenutzer gewünschten Radius, um dem gewünschten Standort herum, ab. Die beiden anderen Eingabefelder fragen die vom Endbenutzer gewünschten Standort ab, hierbei soll der Endbenutzer in ein Eingabefeld den Longitude-Wert und in dem anderen den Latitude-Wert des gewünschten Standort angeben. Die vom Programm gewünschte Eingabe ist mit zwei Textfelder markiert, damit der Endbenutzer die richtigen Eingaben tätigen kann.

5.3 Realisierung der Parkplatzstatus Auswertung

Für die Auswertung des Status der Parkplätze wird zum Auslesen der Bag-Dateien ROS benötigt. Im Nachfolgenden wird die Installation und Einrichtung ebenjener erklärt. Daraufhin folgt die, für die Umsetzung ROS-spezifischen Eigenschaften und dann der Programmablaufplan, welcher beschreibt wie das Vorgehen des Programms bei der Abfrage, Auswertung und Übertragung der Auswertung an die Datenbank vorgeht.

5.3.1 ROS

Im weiteren Verlauf wird erklärt, wie das System mit ROS aufgesetzt wird, wie Packages und Nodes für die Umsetzung der Systemanforderungen erstellt werden.

5.3.2 Installation

Wie im Kapitel 2.5.2 beschreiben, wird die ROS-Kinetic Version genutzt. Diese unterstützt nur die Versionen *Willy* (Ubuntu 15.10), *Xenial* (Ubuntu 16.04) sowie *Jessie* (Debian 8). Hierbei sollte aber auch beachtet werden, dass das End-of-Life Datum für ROS-Kinetic bereits im April 2021 war, da zu diesem Zeitpunkt auch das End-of-Life Datum von *Xenial* gewesen ist. Die Installation in im Nachfolgendem erklärt wird, lässt sich auf der Wiki-Seite von ROS finden [40]:

Kinetic

Im ersten Schritt muss die Ubuntu-Respository so angepasst werden, dass *restricted*, *universe* und *multiverse* Software akzeptiert werden. Dies kann in der GUI des „Software & Updates"-Programm über manuelles Häkchen setzen erledigt werden.
Vor der eigentlichen Installationen werden noch zwei Setup-Optionen vorgenommen, mit denen die Software-Pakete von *packages.ros.org* heruntergeladen werden können und der *Key* eingestellt wird.
Es wird die Desktop-Full Version installiert. Nachdem die Installation abgeschlossen ist, wird die Umgebung aufgesetzt. Hierbei können alle Umgebungsvariablen hinzugefügt werden, so dass diese bei jeder neuen Bash-Session verfügbar sind.
Zum Abschluss des Aufsetzen des ROS-Systems wird um die Nutzbarkeit von ROS zu erreichen, noch die Abhängigkeiten von verschiedenen ROS-Paketen installiert und zudem *rosdep* initialisiert, damit diese Pakete auch genutzt werden können.

Workspace

Zur Erstellung eines Catkin-Workspaces, muss *Catkin* nicht zusätzlich installiert werden, da dieses mit der Installation von ROS mitgeliefert wird. Bevor der Workspace erstellt werden kann, muss erst mal die Umgebung für die Variablen erstellt werden. Dann wird ein Ordner, in diesem Projekt hat der Ordner den gleichen Namen *catkin_ws* wie im Tutorial angegeben, mit einem Unterordner *src* angelegt werden. Im Workspace Ordner *catkin_ws* wird mittels dem Befehl
catkin_make
automatisch die Unterornder *build* und *devel*, sowie die dazugehörigen Unterornder und Dateien erstellt. Nach der Erstellung dieser Ornder und Dateien muss der Befehl
source devel/setup.bash
ausgeführt werden, damit der Catkin-Workspace auch auf die Umgebung gelegt und somit die Umgebungsvariablen an den Catkin-Workspace angepasst werden können.
Um zu überpüfen ob die Umgebungsvariable auch richtig angelegt wurde kann folgender Befehl genutzt werden:
echo $ROS_PACKAGE_PATH

5.4 ROS-spezifische Eigenschaften

In diesem Unterkapitel werden auf die ROS-spezifischen Eigenschaften eingegangen. Hierbei werden nur die Eigenschaften Packages, Parameter und die Launch-Datei betrachtet, da diese Eigenschaften in dieser Arbeit genutzt werden.

5.4.1 Packages

Für dieses Projekt werden folgende Packages erstellt:

- Package für das Auslesen der LiDAR-Dateien und das Auswerten der Parkplätze: *parking_lot_information*
- Package für das Auslesen der GNSS-Daten aus dem Apogee-D: *sbg_rosmsg*

Das Package *sbg_rosmsg* hat nur die Aufgabe die GNSS-Daten, von dem SBG-System Apogee-D, auszulesen und dieser Werte dann als Parameter zu speichern. Die Datei, mit welcher ausgelesen wird, ist in C++ geschrieben. Hierfür wurden Teile der Umsetzung

der Auslesung aus der Arbeit [51] übernommen.
Das Package *parking_ lot_ information* liest mit der Datei *readVelodynePcl.py* die Daten des Velodyne-HDL-32-E ein und wandelt diese in ein PointCloud2-Array. Zudem wird die Zeit der Velodyne-Sensor Messung erfasst und gespeichert. Die Dateien *getParkinglotsInRange.py* und *updateParkinglotsStatus.py* sind für die Verbindung, Abfrage und Updaten der Datenbank zuständig. Die Datei *checkParkinglots.py* ist dafür zuständig alle Parkplätze, die von der Datenbank innerhalb des Messradius zurückgegeben werden, anhand der PointCloud2-Daten auswerten, ob sich dort ein Fahrzeug auf dem Parkplatz befindet oder nicht und diese Erkenntnisse zurückzugeben.
Die ganzen Daten werden in der Datei *dataClass.py* in der Klasse *dataClass* gespichert. Diese wird in der Datei *startRoutine.py* für jede Messung eines Messradius neu instanziert. Solange das Programm zur Messung am Laufen ist, wird eine neue Klasse instanziert und die einzelnen Messungen und Abfragen vorgenommen, zur Speicherung der benötigten Daten, diese werden ausgewertet und die Auswertung in die Datenbank übertragen, dann wartet das Programm 2,448 Sekunden ob eine neue Klasse für neue Messungen zu instanzieren.

5.4.2 Parameter

ROS verfügt über die Möglichkeit einen Parameter-Server zu benutzen, in welchen bestimmte Daten zwischen den Nodes ausgetauscht werden können. Der Parameter-Server hat gegenüber den Topics, welche über Subscriber und Publisher ausgetauscht werden, dass die Daten auch erhalten bleiben, in dem Fall, dass der Publisher ausfällt und somit keine Daten mehr sendet, den Vorteil, dass auf die zuletzt gespeicherten Daten zugegriffen werden kann. In diesem Projekt werden fünf Parameter verwendet. Diese sind in der Tabelle 5.2 aufgelistet. Diese werden mit einem Default-Wert in der Launch-Datei *launch-file.launch* angegeben. Das Package *sbg_ rosmsg* speichert dort die Werte und die Datei *parking_ lot_ information* im Package *parking_ lot_ information* liest diese Werte wieder aus.

5.4.3 Lauch-Datei

ROS bietet mit der Möglichkeit Launch-Dateien zu erstellen, die Möglichkeit, dass die Packages und die ROS-Bag-Dateien gleichzeitig gestartet werden können. Dies erspart das einzelne Aufrufen der jeweiligen Packages und Bag-Dateien einzeln im Terminal.

ROS-Parameter	Datentyp	Name
GNSS-Zeit in Sekunden	int	gps_time_sec
GNSS-Zeit in Nanoekunden	int	gps_time_nsec
Longitude-Wert der Position des Fahrzeugs aus der GNSS-Messung	double	longitude
Latitude-Wert der Position des Fahrzeugs aus der GNSS-Messung	double	latitude
Die Orientierung des Fahrzeuges aus der GNSS-Messung	double	orientation

Tabelle 5.2: Auflistung der ROS-Paramter des ROS-Parameter-Servers

Zudem wird hierbei auch noch sichergestellt, dass die Funtkionen der Packages, welche gleichzeitig starten sollen, auch gleichzeitig gestartet werden.
In diesem Projekt hat die Launch-Datei den Namen *launch-file.launch* und startet die gewünschte Bag-Datei, welche ausgewertet werden soll und die beiden Packages *parking_lot_information* und *sbg_rosmsg*. Zudem werden hier auch alle genutzten ROS-Parameter mit den entsprechenden Datentypen und den Defaultwerten angegeben.

5.5 Programmablauf

Dieses Unterkapitel umfasst den Programmablauf und beschreibt dabei das Auslesen der GNSS- und LiDAR PointCloud2 Daten, die Kommunikation zur Datenbank und das Auswerten der Parkplätze.

5.5.1 Ablauf des Programms

Mit dem Aufrufen der Datei *startRoutine.py* wird zuerst geprüft, ob schon Parameter von der GNSS-Messung vorhanden sind. Danach werden die Messungen gespeichert, die Abfrage, ob sich Parkplätze im Messradius befinden, abgefragt und welche vorhanden sind wird die Auswertung gestartet. Nach der Auswertung werden die Ergebnisse der Auswertung zurück an die Datenbank übersendet. In der Abbildung 5.4 ist der Programmablauf dargestellt.
Die Abbildung 5.5 zeigt den Programmblauf mit den dazugehörigen Dateien als Flussdiagramm.

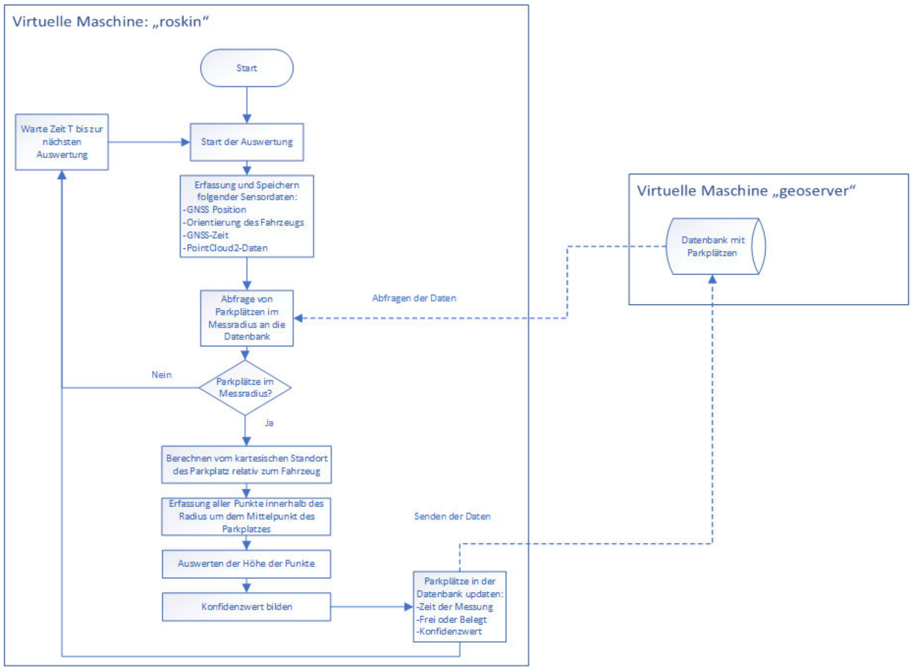

Abbildung 5.4: Flussdiagramm des Ablaufs der Auswertung mit der Kommunikation zur Datenbank

5.5.2 Auslesen der GNSS Daten

Zum Auslesen der GNSS-Daten aus dem SBG-System Apogee-D wird der *sbg-driver* benötigt, welcher auf Github bereitgestellt ist. Dort sind die zum Apogee-D dazugehörigen ROS-Messages, mit welchen die Daten gespeichert werden und abgerufen werden können, vorhanden um diese zum Speichern und Lesen zu benutzen. Diese Messages sind keine Standard-ROS-Messages, sondern es sind Custom-ROS-Messages, welche selbst erstellt von dem Hersteller erstellt sind. Folgende dieser Messages werden für dieses Projekt ausgelesen: [37]

- SbgGpsHdt Message
- SbgGpsPos Message

Aus den zwei Custom-Messages werden nur Teile der vorhandenen Daten in diesen Messages genutzt.

- Aus der SbgGpsHdt Message werden folgende Daten ausgelesen:
  - float32 true_heading

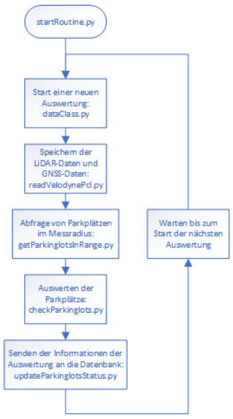

Abbildung 5.5: Programmablauf mit den dazugehörigen Dateien

  - time stamp aus dem Header

- Aus der SbgGpsHdt Message werden folgende Daten ausgelesen:
  - float64 longitude
  - float64 latitude
  - time stamp aus dem Header

Diese Daten werden in als Parameter gespeichert, damit diese durch das Package *parking_lot_information* wieder abgerufen werden können.

5.5.3 Auslesen der LiDAR PointCloud2 Daten

Die LiDAR PointCloud2 Daten die der Velodyne Sensor aufnimmt, werden in der ROS-Message *veldoyne_packets* gespeichert, welche wiederum in die Sensor-ROS-Message *velodyne_points* umgewandelt wird. Diese wird im ROS-Packages *parking_lot_information* mittels einer Python Datei *readVelodynePcl.py* ausgelesen. Hierbei wird die UTC-Zeit der Erfassung der Messung des Velodyne-Sensors zu den dazugehörigen Daten gespeichert, sowie die X-, Y- und Z-Koordinaten der einzelnen Punkte der gesamten Sequenz in einem Array gespeichert.

5.5.4 Abfragen und Übertragung von Daten der Auswertung an die Datenbank

Das Abfragen und Übertragen wird von den beiden Python-Dateien

- getParkinglotsInRange.py
- updateParkinglotsStatus.py

übernommen. Für die Verbindung zur Datenbank wird das Python-Modul *psycopg2* genutzt. Mit dem Datenbank- und Usernamen, sowie der Hostadresse und dem Passwort, wird ein Cursor ertsellt, welche die Verbindung zur Datenbank aufbaut. Über diesen Cursor können Anweisungen an die Datenbank übermittelt und Ergebnisse abgerufen werden.
Die Datei *getParkinglotsInRange.py* sendet eine Anfrage an die Datenbank ob innerhalb des Radius, 17 Meter (aus der Abbildung 4.2 zu entnehmen), sich Parkplätze in der Datenbank befinden, bei welchen ihr Zustand, „Frei“ oder „Belegt“ erfasst und ausgewertet werden soll. Die Datenbank liefert dann die eindeutige Identifikation sowie die Longitude und Latitude Werte zurück, welche an die Python-Datei *dataclass.py* zurückgegeben wird. Für diese Abfrage wurde die *ST_ DistanceSphere* genutzt, sowie die Funktion *ST_-MakePoint* benutzt, um die Einträge innerhalb des Radius um die derzeitige Messung, aus der Datenbank auszulesen
Die Datei *updateParkinglotsStatus.py* bekommt die Listen der eindeutigen Parkplatzidentifikation, die der Status der Parkplätze und die der Konfidenzwerte. Zudem wird auch noch die Uhrzeit der Messung übergeben. Der Status, der Konfidenzwert aus den beiden Listen, zusätzlich zu der Messzeit werden entsprechend der eindeutigen Parkplatzidentifikation an die Datenbank gesendet, um ebenjene Einträge der jeweiligen Parkplätze in der Datenbank mit den neuen Informationen zu überschreiben.

5.5.5 Auswerten der Parkplätze

Zum Auswerten der Parkplätze ist die Datei *checkParkinglots.py* zuständig. Sie benutzt die Haversine Formel und die Peilunsgwinkel Formel um die Position des Parkplatzes relativ zu der Position des Fahrzeuges im kartesisch Koordinatensystem zu ermitteln. Daraufhin werden die Z-Werte aller Punkte innerhalb des Messradius in eine Liste gespeichert. Diese Werteliste wird auf den Offsetintervall geprüft. Jeder Punk, dessen Z-Wert innerhalb dieses Intervalls liegt, erhöht einen Zähler um eins. Wenn alle Punkte überprüft wurden, wird, um den Konfidenz-Wert zu bilden, der Wert des Zählers durch die Länge der Liste geteilt. Dieser Wert wird geprüft, ist dieser größer als 0,5 ist der Parkplatz „Frei“ ansonsten ist er „Belegt“. Der Konfidenzwert und der Status werden in die vorgesehenen Listen hinzugefügt. Dies wird für jeden Parkplatz die von der Datenbank zurückgeliefert worden, durchgeführt. Die Listen werden bei der Update der Datenbank aufgerufen um den Parkplätzen die Ergebnisse ihrer Auswertung zuordnen zu können.

6 Testen des Sytsems

6.1 Testaufbau

Während der Fahrt, bei dem die Messdaten aufgenommen werden, wird die Aufnahme der Daten in die ROS-Bag Datei von einem Laptop aus überwacht, welcher über das Programm VNC-Viewer auf den Computer zugreift.

6.2 Testumgebung

Für die Testfahrt wird die Strecke der TAVF Hamburg (Teststrecke für automatisiertes und vernetztes Fahren Hamburg) ausgewählt. Die vermessenen Parkplätze und das Testen des System beschränkt sich aus diesem Grund auf die Teststrecke oder die unmittelbare Umgebung der Teststrecke. Der Verlauf dieser Teststrecke und die Orte der ausgewerteten Parkplätze sind in der Abbildung 6.1 [46] zu sehen. Die Wetterbedingungen während der Fahrt fahren optimal zum Einsatz der Sensoren, es ist ein sonniger Tag gewesen.
In der Tabelle 6.1 sind die Anzahl der in der Messung beachteten Parkplätze mit der dazugehörigen Orientierung vermerkt.

Orientierung der Parkplätze	Anzahl der Parkplätze
Längs zur Straße	5
Quer zur Straße	5

Tabelle 6.1: Übersicht über die in der Messung beachteten Parkplätze

Abbildung 6.1: Der Verlauf der TAVF Hamburg mit Markierungen an den Orten, an welchen eine Auswertung der Parkplätze stattgefunden hat

6.3 Erfassung und Eintragung der Parkplätze

Die Erfassung und Eintragung der Parkplätze findet vor dem Start des Programms zur Auswertung statt, da dieses Programm nur in der Datenbank eingetragene Parkplätze und Informationen über deren Status, der schon eingetragenen und im Messradius vorhandenen Parkplätzen, auswertet. Die Vorgang der der Erfassung der Parkplätzen hat folgenden Ablauf:

1. Ausmessen der Länge und Breite des Parkplatzes vor Ort
2. Ermitteln der GPS-Position des Parkplatzes
3. Anlegen des Parkplatzes und Eintragen der Daten in die Datenbank

Das Ausmessen der Maße der Parkplätze hat vor Ort mit einem Zollstock stattgefunden. Für die Ermittlung der GPS-Position des Parkplatzes, wird auf die GPS-Position des Fahrzeuges zurückgegriffen. Da das Fahrzeug an jedem der hier erfassten Parkplätze vorbeigefahren ist, kann aus der Orientierung und der Distanz bis zum Mittelpunkt des Parkplatzes, sowie die GPS-Position des Fahrzeuges, die GPS-Position des Parkplatzes ermittelt werden. Die Messung der Distanz erfolgte über das ROS-Tool RVIZ in den LiDAR PointCloud2 Daten. Die Messung ist beispielhaft in der Abbildung 6.2 dargestellt. Die Peilung hat bei diesem Vorgehen den Wert von 90°. Für die GPS-Position, werden

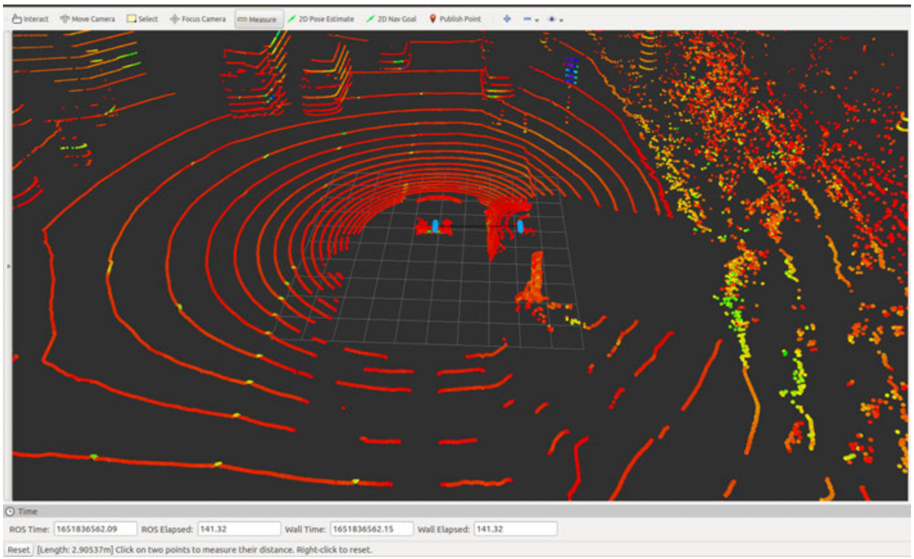

Abbildung 6.2: Erfassen der Position eines Parkplatzes mit Hilfe der LiDAR PointCloud2 Daten, der Distanz, der Position des Fahrzeuges und der Orientierung

die Werte genommen, die so zeitgleich wie möglich mit der LiDAR Distanzmessung aufgenommen wurden. Dafür wird in der Launch-Datei die Durchlauflänge auf die Länge gestellt, bei der die Distanz gemessen wird und der letzte gemessene GNSS-Wert als Position des Fahrzeuges angenommen. Mit diesen Werten und der Formel [49]

$$\phi_2 = asin(sin(\phi_1) \cdot cos(\gamma) + cos(\phi_1) \cdot sin(\gamma) \cdot cos(\theta) \tag{6.1}$$

und

$$\lambda_2 = \lambda_1 + arcustan2(sin(\theta) \cdot sin(\gamma) \cdot cos(\phi_1), cos(\gamma) - sin(\phi_1) \cdot sind(\phi_2)) \tag{6.2}$$

kann dann die GPS-Position des Parkplatzes errechnet werden. Hiebrei ist ϕ die Latitude Werte, λ die Longitude Werte, θ der Winkelabstand und γ die Peilung.
Während der Auswertung ist aufgefallen, dass die Orientierung des Fahrzeuges falsch gemessen wurde. Die Orientierung ist um 180° gedreht, was bei dieser Methode mitbrachtet werden muss.
Bei diesem Vorgehen treten folgende Probleme auf, die eine Ungenauigkeit in die Erfassung der Parkplätze zur Folge haben. Zum einem ist die Abmessung der Länge und Breite der Parkplätze ohne eine klare Abgrenzung der Parkplätze zueinander nicht vollumfänglich sicher ob die tatsächliche Länge und Breite des Parkplatzes, wie ihn andere

Verkehrsteilnehmer wahrnehmen, die in die Datenbank eingetragene ist. Dies ist der Fall bei mehreren aneinander gereihten Parkplätzen, bei dem andere parkende Fahrzeuge auch mittig auf zwei Parkplätzen stehen können.
Zudem ist die Erfassung der GPS-Position des Parkplatzes fehlerbehaftet. Dies ist zum einem dadurch begründet, dass RVIZ nur den Abstand zwischen gemessenen LiDAR-Punkten feststellen kann, welche nicht unmittelbar auf dem exakten Mittelpunkt des Fahrzeuges liegen müssen, zum anderen wird der Fehler der GNSS-Messung der Position des Fahrzeuges bei der Ermittlung der GPS-Position des Parkplätze mit übernommen. Für eine erste Umsetzung der Problemstellung erscheint dieser Fehler durch die Überprüfung der Auswertung vernachlässigbar, sollte aber in weiterführenden Umsetzungen bei Problemstellung dieser Art umfangreicher angegangen werden.
Die Parkplätze welche ausgewertet werden sollen werden nach der Vermessung mit den dazugehörigen Daten manuell in die Datenbank eingetragen.

6.4 Ablauf der Auswertung

Die Auswertung der Parkplätze findet Offline statt, dass bedeutet, es ist nicht während der Fahrt ausgewertet worden, sondern danach mit dem entsprechenden ROS-Bag Dateien. Da diese die Daten in Echtzeit wiedergeben, kann dies auch während einer Fahrt umgesetzt werden. Aus diesem Grund werden die ROS-Bag Dateien, welche ausgewertet werden sollen, in der Launch-Datei angegeben. Diese startet dann die Packages zur Auswertung zeitgleich mit dem Abspielen der Bag Datei. Durch dieses Vorgehen ist eine Auswertung Zeit und Standort unabhängig von Messfahrten und eine Anpassung der Auswertung ist im Nachhinein möglich, solange die Daten noch vorhanden sind, welche in diesem Projekt ausgewertet werden. Die folgende Liste ist eine Auflistung der ROS-Bag Dateien, die im diesem Projekt ausgewertet werden.

- 2022-04-26-12-34-38_19.bag
- 2022-04-26-13-01-07_77.bag
- 2022-04-26-13-11-09_98.bag

6.5 Auswertung der LiDAR Messung

In diesem Unterkapitel der Auswertung werden zuerst die Längs-Parkplätze ausgewertet, gefolgt von der Auswertung der Quer-Parkplätze und abschließend den Vergleich dieser zwei Auswertungen.

6.5.1 Längs-Parkplätze

Für die Auswertung der Parkplätze ist es interessant zu wissen, ob die Erfassung der Standorte der Parkplätze in einem akzeptablen Rahmen ist, denn ansonsten werden Orte ausgewertet, welche nicht am Standort des Parkplatzes sind. Dafür ist mit dem Python Modul *pptk* die LiDAR PointCLoud2 Daten visualisiert worden. Die Punkte innerhalb des Radius sind farblich markiert, sie stellen also den Ort da, der vermessen wird. Die Abbildung 6.4 zeigt den ausgewerteten Standort mit der, im Unterpunkt 6.3, aufgeführten Methode. Die Abbildung 6.3 zeigt den ausgewerteten Standort, dessen GNSS-Position mit

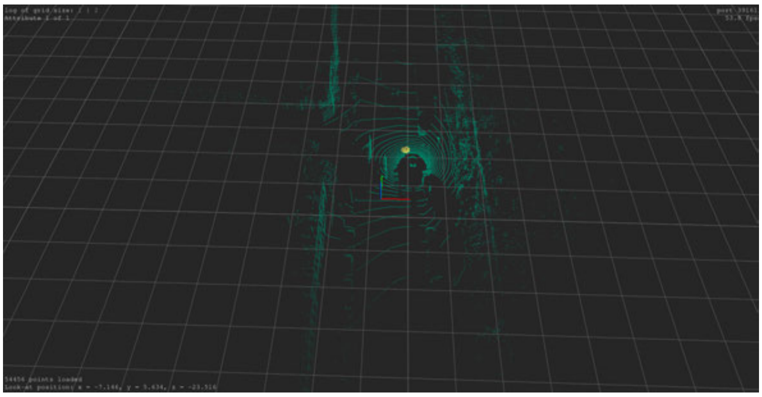

Abbildung 6.3: Parkplatz-Standort gemessen mit dem parkenden Fahrzeug und dem Apogee-D

dem Apooge-D während des Parken des Fahrzeuges auf dem Parkplatz erfasst worden ist. Es ist erkennbar, dass die Methode aus dem Unterkapitel 6.3 näher am Standort des Parkplatzes liegt, als diesen mit dem Apogee-D zu vermessen. Daraus lässt sich schließen, dass für eine genaue Erfassung des Standortes der Parkplätze, eine sehr genaue Messung durchgeführt werden muss.
Es sind die 5fünf Längs-Parkplätze ausgewertet worden. Einmal in der Bag-Datei 2022-04-26-12-34-38_19.bag und in der Bag-Datei 2022-04-26-13-01-07_77.bag. Beide sind

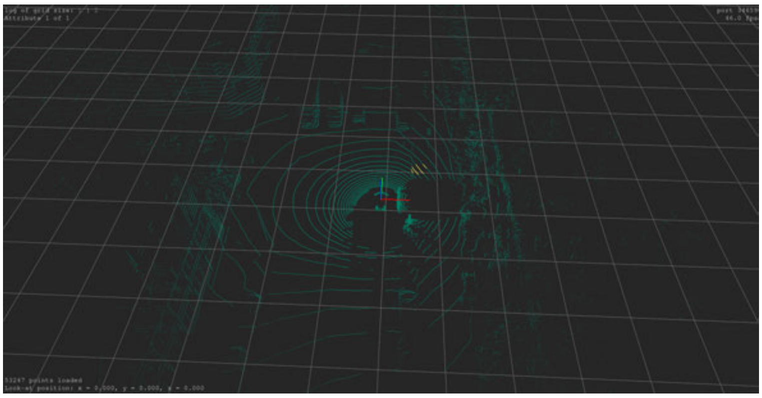

Abbildung 6.4: Parkplatz-Standort gemessen mit der Position des Fahrzeuges, der Distanz und dem Peilungswinkel

zehn mal durchlaufen gelassen. Zudem sind die Messung einmal mit den 10% Offsetintervall, siehe Unterpunkt 4.3.4, und einmal mit einem 15% Offsetintervall durchgeführt worden. Dabei sind in der Bag-Datei 2022-04-26-12-34-38_19.bag bei dem 10% Offsetintervall 80 Auswertungen und bei 15% Offsetintervall 74 Auswertungen vorhanden. Bei der Bag-Datei 2022-04-26-13-01-07_77.bag sind bei dem 10% Offsetintervall 223 Auswertungen und bei 15% Offsetintervall 229 Auswertungen vorhanden. Insgesamt ergibt das 606 Auswertungen.
Bei den Auswertungen ist die Wartezeit zwischen erfolgreichen Messungen auf 0.1 Sekunden abgesenkt worden, um mehr Auswertungen in den zehn Durchläufen zu erfassen. Die Tabelle 6.2 ist die Auswertung der True- und False-Positiv, sowie die Auswertung der True- und False-Negativ Status der Parkplätze, der Bag-Datei 2022-04-26-12-34-38_19 aufgeführt. Es gibt bei der Bag-Datei 2022-04-26-12-34-38_19 keine signifikante Ände-

	10% Offsetintevrall	15% Offsetintervall
True Positiv	9 (36%)	12(35,3%)
False Positiv	16 (64%)	22 (64,7%)
True Negativ	53 (96,3%)	37 (92,5%)
False Negativ	2 (3,7%)	3 (7,5%)

Tabelle 6.2: Auswertung der Bag-Datei 2022-04-26-12-34-38_19.bag

rungen der True- und False-Positiv Auswertungen. Bei den False-Negativ Werten hat sich der Anteil von 3,7% auf 7,5% mehr als verdoppelt.
Die Tabelle 6.3 ist die Auswertung der True- und False-Positiv, sowie die Auswertung der True- und False-Negativ Status der Parkplätze, der Bag-Datei 2022-04-26-13-01-07_77

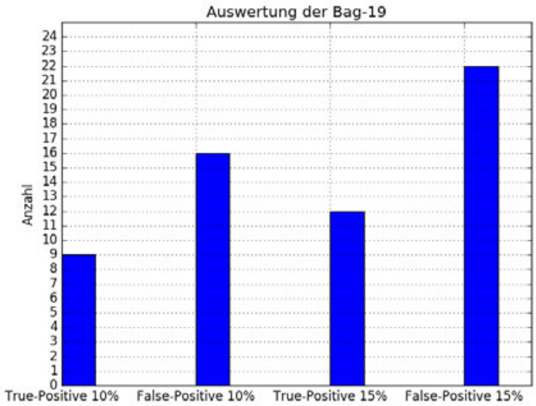

Abbildung 6.5: Auswertung der Parkplätze nach True Positive und False Positive für 10% und 15% Offsetwert der Bag-Datei 2022-04-26-12-34-38_19.bag

aufgeführt. Es gibt bei der Bag-Datei 2022-04-26-13-01-07_77 eine signifikante Ände-

	10% Offsetintevrall	15% Offsetintervall
True Positiv	29(42,0%)	38 (56,7%)
False Positiv	40 (58,0%)	29 (43,3%)
True Negativ	144 (90,0%)	134 (85,9%)
False Negativ	16 (10,0%)	22 (14,1%)

Tabelle 6.3: Auswertung der Bag-Datei 2022-04-26-13-01-07_77.bag

rungen der True- und False-Positiv Auswertungen, von einer Steigerung um 31,03%. Die False-Negativ Werte hat sich um 28% verschlechtert.
Keine dieser Auswertungen erfüllt das Kriterium von mindestens 70% True-Positive Parkplätzen.
Eine Steigerung der möglichen Abweichung im Bezug der Höhe der Punkte kann zu eine Verbesserung der Erkennung von True-Positive Parkplätzen führen, gleichzeitig muss dadurch eine Verschlechterung der True ´-Negativ Parkplätze in Kauf genommen werden.
In der Abbildung 6.5 ist ein Balkendiagramm der Auswertung der Bag-Datei 2022-04-26-12-34-38_19.bag abgebildet. Es sind die True-Positive und False-Positive Parkplätze für einen Offsetwert von 10% und 15% aufgeführt.
In der Abbildung 6.6 ist ein Balkendiagramm der Auswertung der Bag-Datei 2022-04-26-13-01-07_77.bag abgebildet. Es sind die True-Positive und False-Positive Parkplätze für einen Offsetwert von 10% und 15% aufgeführt. Während der Auswertung ist aufgefallen, dass sich eine genaue Wiederholung der gleichen Auswertung schwer fällt, da es

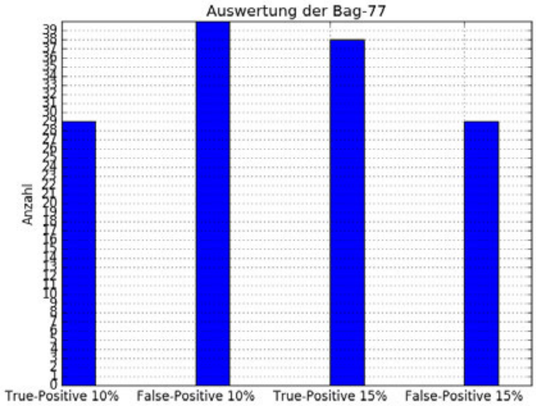

Abbildung 6.6: Auswertung der Parkplätze nach True Positive und False Positive für 10% und 15% Offsetwert der Bag-Datei 2022-04-26-13-01-07_77.bag

auf den Prozessmanager ankommt, wie groß die zeitliche Differenz zwischen den Messungen ist. Dies kann zustande kommen, wenn der eine Prozess gegenüber den anderen stark bevorzugt wird und dadurch mehrere Auslesungen der Messungen des einen Sensor vorgenommen wird, bevor der andere Sensor ausgelesen wird. Dennoch ist bei der Auswertung aufgefallen, dass ich auch gleiche Ergebnisse bei der Auswertung einstellen. Dies lässt sich darauf zurückführen, wenn der Prozessmanager das Auslesen der Messungen der Sensoren so gleichwertig behandelt, dass die zeitliche Differenz kleiner als der vorgegeben Wert ist. Die Zeit der Aufzeichnung der Messung bleibt gleich, so dass beim wiederholten Abspielen der Bag-Dateien zur Auswertung der gleiche Zeitpunkt der Messungen getroffen werden kann und der Prozessmanager das Auslesen innerhalb der zeitlichen Differenz zustande bekommt.
Auch ist aufgefallen, dass die Verbildlichung der Auswertung ergeben hat, dass in einigen Fällen die Parkplätze welche vor dem Fahrzeug sind besser ausgewertet werden, als die Parkplätze hinter dem Fahrzeug. In der Abbildung 6.7 ist zusehen, dass der Parkplatz 3, der dritte von oben, nicht als frei erkannt wird. Es ist zu erkennen, dass dieser versetzt gemessen wird und ein parkendes Fahrzeug im nahem Blickfeld zwischen dem Fahrzeug und dem freien Parkplatz sich befindet. In der Abbildung ist hingegen zu sehen, dass der Parkplatz 3, welcher der erste von oben ist, besser erfasst wird und mehr in die Richtung des Standorts des Parkplatzes liegt, als in der Abbildung 6.7.

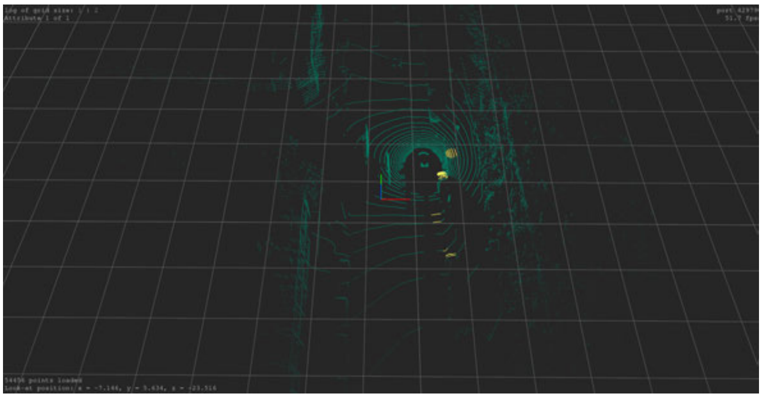

Abbildung 6.7: Ein freier Parkplatz 3 wird nicht erkannt, durch das parkende Fahrzeug in Sichtreichweite

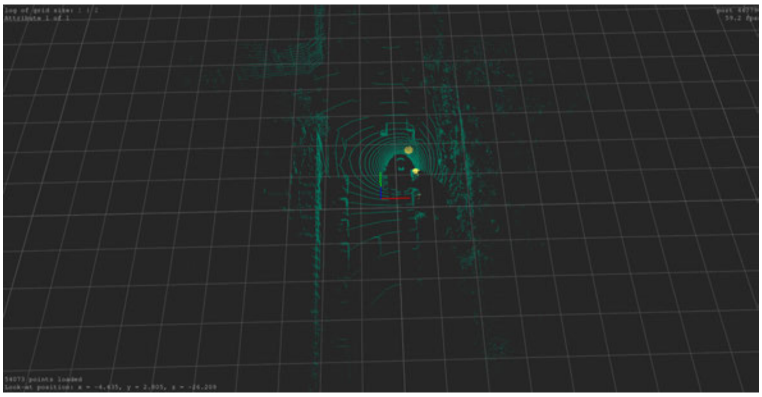

Abbildung 6.8: Der freie Parkplatz 3, der diesmal besser erkannt wird, wenn dieser vor dem Fahrzeug liegt

6.5.2 Quer-Parkplätze

Die Auswertung der Quer-Parkplätze ist mit der Bag-Datei 2022-04-26-13-11-09_98.bag durchgeführt worden. Hierbei wurden bei einem Offsetwerte Intervall von 10% 756 Auswertungen und bei einem Offsetwerte Intervall von 15% 800 Auswertungen vorgenommen. In beiden Fällen wurde die Bag-Datei fünf abgespielt und der Wartezeit raus genommen, um mehr Ergebnisse zu bekommen. Aus der Tabelle 6.4 geht hervor, dass nur bei einen Offsetwerte Intervall von 15% das Kriterium von mindestens 70% True-Positive Parkplätzen erfüllt ist. Interessant ist zudem, dass sich durch die Umstellung des Offsetwerte Intervalls sich nicht auf die Erkennung von belegten Parkplätzen ausgewirkt hat, welche zuverlässig mit 100% erkannt worden sind.
In der Abbildung 6.9 ist ein Balkendiagramm der Auswertung der Bag-Datei-98 abge-

	10% Offsetintevrall	15% Offsetintervall
True Positiv	291(48,2%)	424 (72.4%)
False Positiv	260 (52,8%)	162 (28,6%)
True Negativ	205 (100,0%)	214 (100,0%)
False Negativ	0 (0,0%)	0 (0,0%)

Tabelle 6.4: Auswertung der Bag-Datei 2022-04-26-13-11-09_98.bag

bildet. Es sind die True-Positive und False-Positive Parkplätze für einen Offsetwert von 10% und 15% aufgeführt.

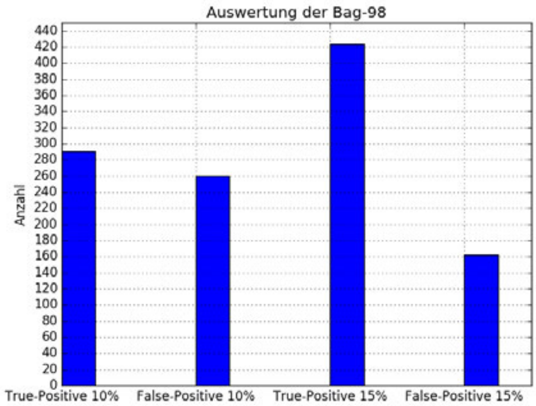

Abbildung 6.9: Auswertung der Parkplätze nach True Positive und False Positive für 10% und 15% der 2022-04-26-13-11-09_98.bag

6.5.3 Vergleich zwischen Längs- und Quer-Parkplätzen

Es ist zu erkennen, dass die Auswertung der Quer-Parkplätze deutlich bessere Ergebnisse erzielt hat als die Auswertung der Längs-Parkplätze. Bei der Auswertung der Quer-Parkplätzen ist bei dem erweiterten Offsetwerte Intervall von 15% die Anforderung der True-Positive Parkplätze erfüllt.

6.6 Auswertung der Konfidenz

Für die Auswertung der Anforderung des Konfidenzwert der freien Parkplätze werden in der Tabelle 6.5 die Anzahl der freien Parkplätzen, der Anzahl der freien Parkplätzen mit einem Konfidenzwert von mindestens 70% gegenübergestellt. Dies ist für jede Auswertung der Bag-Dateien mit den entsprechenden Offsetwerten abgebildet.

Bag-Datei und Offsetwerte	Freie Parkplätze	Freie Parkplätze mit einem Konfidenzwert von mindestens 70%
Bag-19 mit 10%	28	9 (32,14%)
Bag-19 mit 15%	34	12 (35,29%%)
Bag-77 mit 10%	69	29 (42,03%)
Bag-77 mit 15%	67	38 (55,07%)
Bag-98 mit 10%	551	291 (52,81%)
Bag-98 mit 15%	586	424 (72,35%)

Tabelle 6.5: Gegenüberstellung der freien Parkplätze mit den freien Parkplätzen mit einem Konfidenzwert von mindestens 70%

Bei drei der sechs Auswertungen werden mehr als die Hälfte der Parkplätze mit einem Konfidenzwert von mindestens 70% erreicht. Der höchste Wert liegt bei 72,35% aller ausgewerteten Parkplätze dieser Auswertung vor. Dies ist bei der Auswertung der Bag-Datei 2022-04-26-13-11-09_98.bag mit einem Offsetwerte Intervall von 15% gegeben.

6.7 Auswertung der Latenz

Um die Latenz auszuwerten, wird die Zeit beim Start einer Auswertung und am Ende der Auswertung gemessen. Es sind insgesamt 147 Latenzwerte aufgenommen worden. Die Latenzwerte sind in der Datei *latenz.txt* abgespeichert. Bei der maximalen Latenz von 500 ms sind 146 der 147 unterhalb dieser Grenze. Dies ist ein Anteil von 99,3% der Auswertungen welche die Anforderungen der maximalen Latenz erfüllen. In der nachfolgenden Tabelle 6.6 sind noch weitere Latenzen und die dazugehörigen Anzahl der Auswertungen, die innerhalb dieser Zeit durchgeführt worden sind.
Die Anforderung der Latenz von 500 ms ist bis auf eine Ausnahme erfüllt. Auch eine Latenz von 400 ms kann sich noch in einem akzeptablen Rahmen bewegen, da hier nur jede zwanzigste Auswertungen dann die Anforderung an die Latenz von 400 ms überschreiten würde.

Latenz	Anzahl an Auswertungen die innerhalb dieser Zeit durchgeführt sind	Prozentualer Anteil
100 ms	31	21,1%
200 ms	79	53,7%
300 ms	106	72,1%
400 ms	140	95,2%
500 ms	146	99,3%

Tabelle 6.6: Latenzen mit deren Anzahl der Auswertung, die diese Latenz erfüllen

6.8 Auswertung der maximalen Geschwindigkeit

Während der Realisierung des Konzeptes, ist das Problem aufgetaucht, dass die zeitliche Differenz zwischen der Messung der LiDAR-Daten und die der GNSS-Daten, bedingt durch die Umsetzung des Programms, teilweise eine Differenz annimmt, die für eine sinnvolle Auswertung zu hoch ist. Dies tritt trotz der geringen zeitlichen Differenz der Messungen, 5 Hz bei den GNSS-Daten und 10 Hz bei den LiDAR-Daten, auf. Daher ist dieses Problem auf das Programm, welches die Auswertung durchführt, zurückzuführen und unabhängig von den Sensoren zu betrachten. Diese zeitlichen Differenz der Messung kann bei der Auswertung zu Problemen führen, da diese zeitliche Differenz, die Differenz der Messung der LiDAR-Daten und der GNSS-Daten darstellt. Eine große zeitliche Differenz bedeutet, dass das Fahrzeug eine höhere Abweichung als Grundlage des eigenen Ausgangspunktes zur Auswertung annimmt, als es sich zu diesem Zeitpunkt tatsächlich befunden hat. Mit höherer Abweichung wird die Wahrscheinlichkeit dass bei der Abfrage der Parkplätze, Parkplätze außerhalb des Messradius für die Auswertung von der Datenbank zurückgegeben werden können, höher. Zudem ist der Fehler, der bei der Berechnung der Distanz der X-Achse und Y-Achse, um innerhalb der LiDAR PointCLoud2 Daten die Überprüfung der Fläche vorzunehmen, bei größeren Abweichung der tatsächlichen zur gemessenen Position größer. Dies kann dazu führen, dass eine falsche Position, möglicherweise der Parkplatz neben den zum Auswerten gewünschten Parkplatz, ausgewertet wird.
Aus diesem Grund werden zwei verschiedene Arten der Datenübertragung in ROS auf die zeitliche Differenz und die Distanz, die dadurch im Straßenverkehr zurückgelegt werden kann. Die beiden Arten der Datenübertragung die verglichen werden, sind die Übergabe über die Parameter mit dem Parameter-Server und die Übergabe über die Topics, welche von Publisher gesendet und Subscriber empfangen werden.
Für die nachfolgenden Tabelle 6.7 sind die zeitliche Differenzen, bei der Datenübergabe

mittels ROS-Parametern, die bei der Auswertung entstehen, in der Textdatei *deltatime-1.txt* gespeichert worden. Diese Textdatei enthält 93 Werte der Zeitdifferenz, die in Nanosekunden angegeben sind. Die Tabelle gibt die minimale, die maximale und die durchschnittliche Zeitdifferenz an. Zusätzlich wird die jeweilige Entfernung in Metern für die Geschwindigkeiten 30 km/h und 50 km/h ausgerechnet und in der Tabelle angeführt. Die

	Minimale Zeitdifferenz	Maximale Zeitdifferenz	Durchschnittliche Zeitdifferenz
Zeit in ms	8,41	6034,77	967,87
30 km/h: Distanz in Meter	0,0701	50,2897	8,0656
50 km/h: Distanz in Meter	0,1168	83,8162	13,4426

Tabelle 6.7: Zeitliche Differenzen bei der Übertragung der Daten über Parameter

Abbildung 6.10 zeigt die dazugehörige Verteilung der Zeiten als Histogramm, die Zeiten sind in Sekunden angegeben. Die nachfolgende Tabelle 6.8 stellt die gleiche Auswertung,

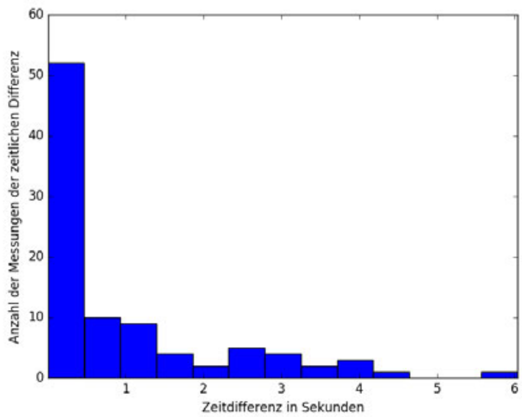

Abbildung 6.10: Zeitliche Differenzen bei der Übertragung der Daten über Parameter

wie im vorgehenden Punkt, auf, nur dass hierbei die Daten mittels Topics über Subscriber und Publisher übermittelt werden. Es sind 95 Werte in die Textdatei *deltatime-2.tx* zur Auswertung aufgenommen worden. Die Abbildung 6.11 zeigt die dazugehörige Verteilung der Zeiten als Histogramm, die Zeiten sind in Sekunden angegeben. Die Auswertung ergibt, dass der Ansatz, die Daten über die Parameter zu übergeben, eine deutliche geringe Zeitdifferenz aufweist. Die Faktoren, um die der Ansatz die Parameter zu übergeben besser ist, gegenüber den Ansatz die Daten über Topics zu übergeben ist, ist durch-

	Minimale Zeitdifferenz	Maximale Zeitdifferenz	Durchschnittliche Zeitdifferenz
Zeit in ns	458528636	11060437328	2253461046,16
30 km/h: Distanz in Meter	3,8211	92,1703	18,7788
50 km/h: Distanz in Meter	6,3685	153,6172	31,2981

Tabelle 6.8: Zeitliche Differenzen bei der Übertragung der Daten über Topics

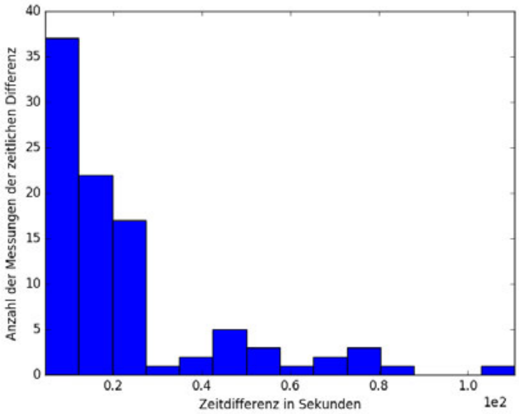

Abbildung 6.11: Zeitliche Differenzen bei der Übertragung der Daten über Topics

schnittlich:

$$\frac{92,1703\ m}{50,2897} = 1,8328. \tag{6.3}$$

Obwohl der Ansatz, die Daten über Parameter zu übertragen besser ist als die Daten über Topics zu übertragen, ist die durchschnittliche zeitliche Differenz mit der dazugehörigen Distanz 8,0656 Metern bei 30 km/h und 13,4426 Metern bei 50 km/h eine Abweichung, die als zu groß betrachtet werden kann. Aus diesem Grund wird bei der Realisierung eine Abfrage abgebaut, welche sicherstellt, die Daten für die Parkplätze nur dann auszuwerten, wenn eine zeitliche Differenz von weniger als 0,1 Sekunden besteht.
Es lässt sich festhalten, dass wenn dieses Problem der Systemumsetzung beachtet wird und die maximale Abweichung der GNSS-Postion aus der Tabelle 2.4 von einem Meter, sowie die Abweichung für 30 km/h bei 0,1 Sekunden von 0,83 Metern genommen wird, dass bei einer Fahrt eine Abweichung von 1,83 Metern entstehen kann. Der beste Fall mit der minimalen Abweichung von 0,06 Metern aus der Tabelle 2.4 ergibt eine Abweichung von 0.89 Metern.

Entsprechend ist für die die Geschwindigkeit von 50 km/h bei 0,1 Sekunden eine zusätzliche Abweichung von 1,39 Metern. Dies ergibt eine maximale Abweichung von 2,39 Metern und eine minimale Abweichung von 1,45 Metern.
Nur bei einer Geschwindigkeit von 30 km/h ist es bei einer genauen Positionbestimmung möglich, eine Abweichung von weniger als einem Meter zu erreichen. Aus diesem Grund ist die Muss-Anforderung von 30 km/h Geschwindigkeit während der Auswertung erfüllt und die Kann-Anfoderung von 50 km/h nicht erfüllt.

6.9 Zusammenfassung der Auswertung

Die True-Positive Anforderung von 70% ist nicht zufriedenstellend vom System erfüllt worden. Nur in einem der sechs Auswertungen ist dieses Kriterium erfüllt worden. Dies hat mehrere Gründe. Ein Grund ist die Bestimmung der eigenen Position des Fahrzeugs, die während der Fahrt von der Umgebung und der Anzahl der Satelliten beeinflusst werden kann. Zudem ist für eine erfolgreiche Auswertung eine sehr genaue Positionsbestimmung des Mittelpunkts des Parkplatzes notwendig, welches in dieser Arbeit nicht erreicht worden ist.
Zudem besteht das Problem der Sichtfeldverdeckung, wodurch im Weg stehende Objekte wie Fahrzeuge oder Büsche die Erfassung der LiDAR-Daten dahin beeinträchtigen können, dass nur Daten von diesen Objekte aufgenommen werden, nicht aber von den dahinter liegenden freien Parkplätzen.
Die Anforderung an die Latenz von 500 ms ist bis auf einen Fall erfüllt. Insgesamt haben 99,3% der Auswertungen diese Anforderung erfüllt.
Die Anforderung des Konfidenzwert von 70% ist größtenteils erfüllt. Für die Hälfte der Auswertungen haben über die Hälfte der erkannten freien Parkplätze einen Konfidenzwert von mehr als 70%.
Bei der Anforderung an die Geschwindigkeit während der Auswertung ist die Muss-Anforderung von 30 km/ erfüllt und die Kann-Anforderung von 50 km/h nicht erfüllt.

6.10 Anzeigen der Parkplätze

Das Anzeigen der Parkplätze konnte nicht in der Form umgesetzt werden, dass der Endbenutzer die Parkplätze angezeigt bekommt. Die Eingabe der benötigten Informationen

und die Abfrage ob der *nächstgelegende Parkplatz* oder alle *Parkplätze im Radius* angezeigt werden sollen, funktioniert.
Das Problem taucht beim Anzeigen der Parkplätze an. Um zu überprüfen, ob die Abfrage der Layer an die Datenbank ein gewünschtes Ergebnis liefert, wurde die Layer-Preview Funktion des Geoservers genutzt. Mit der Preview Funktion für die Layer können die zum Layer gehörigen Daten auf der Preview Webseite angezeigt werden. Diese hat den Nachteil keine Hintergrund Karte zu haben, sondern einen weißen Hintergrund. Wenn auf der Preview Webseite die URL dahingehend angepasst wird, dass die Parameter für die Datenbankabfrage manuell eingegeben werden, sie also die Variablen ersetzen, erscheinen die gewünschten Parkplätze. Mit Hilfe des Mauszeigers können die Koordinaten der angezeigten Parkplätze geprüft werden. Diese entsprechen denen die aus der Datenbank abgefragt wurden. Dies ist in der Abbildung 6.12 dargestellt.

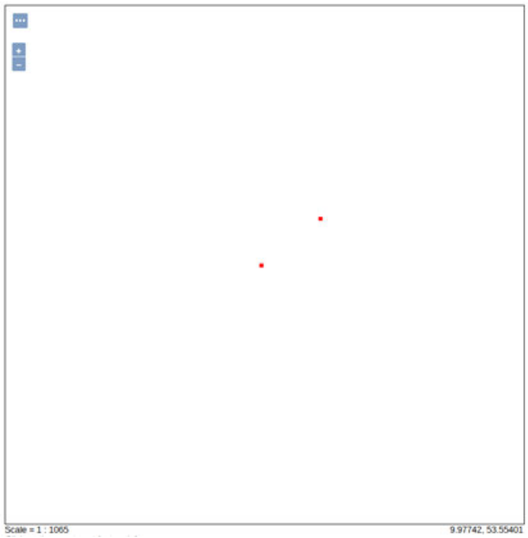

Abbildung 6.12: Abfrage der Parkplätze im Radius, mit der Anzeige der beiden freien Parkplätze im Radius

7 Fazit und Ausblick

Das letzte Kapitel befasst sich mit dem Fazit der Arbeit und den Ausblick, wie dieses Projekt verbessert und erweitert werden kann.

7.1 Fazit

Diese Arbeit und die Umsetzung des Projekts hat einen Einblick geliefert, wie die Parkplatzsituation innerhalb der Stadt in, für das automatisierte und vernetzte Fahren taugliche, aussehen kann. Hierbei wird für die Teilnehmer des automatisierten und vernetzten Fahren eine *Local Dynamic Map* erstellt, auf welche die Teilnehmern, welche die Daten bereitstellen und updaten, sowie die Endbenutzer dieser Daten auf diese zugreifen können.
Auch wenn der Aufbau der *Local Dynamic Map* zufriedenstellend gelungen ist und der Kommunikation zu dieser, hat die Erkennung Parkplätze noch Verbesserungsbedarf. Die größte Schwachstelle ist die richtige Erfassung der Standorte der Parkplätze und des Fahrzeuges im Straßenverkehr.
Die Anforderung der Latenz und der Kann-Anforderung der Geschwindigkeit wurden in dieser Arbeit zufriedenstellend gelöst. Die Anforderung an den Konfidenzwert, mit welcher Zuversicht die freien Parkplätze erkannt werden, ist überwiegend zufriedenstellend gelöst worden.
Die Anforderung an den Anteil der True-Positive erkannten Parkplätze wurde nicht zufriedenstellend gelöst. Hier besteht für zukünftige Projekte und Arbeiten, die an diese Arbeit anknüpfen, Verbesserungsbedarf.

7.2 Ausblick

Wie im Fazit erwähnt besteht der notwendigste Verbesserungsbedarf darin, die Position der Parkplätze und die Position des Fahrzeuges im Straßenverkehr zu erfassen, damit die Umsetzung des System auch den richtigen Ort auswertet. Wenn die Standorte der Parkplätze falsch in der Datenbank erfasst sind, aber der Standort des Fahrzeuges richtig erfasste wird, oder umgekehrt, können die Auswertungen einen Fehler in Abhängigkeit der Fehler der Standortbestimmung aufweisen.

7.2.1 Position der Parkplätze

Die Position der Parkplätze, welche in der Datenbank erfasst werden, sind in dieser Arbeit manuell erfasst worden. Eine Alternative, die zwar mehr Aufwand bedeutet, aber den Eintragungsprozess automatisieren könnte, ist der nicht ausgewählte Ansatz *Parkplatz selbst erfassen und abspeichern*. Dadurch könnten mehr Parkplätze eintragen werden. Allerdings muss hierbei aufgepasst werden, dass der Fehler der Positionsbestimmung der Parkplätze möglichst klein gehalten wird und dass das System eine Überprüfung des richtigen Standortes des Parkplatzes durchführen kann, wenn dieser Parkplatz nach der ersten Eintragung in die Datenbank, weiterhin abgefahren, vom System vermessen und als potentieller Parkplatz erkannt wird.
Wenn weiterhin auf dem Ansatz *Parkplatz aus Datenbank überprüfen und Status aktualisieren* aufgebaut werden soll, ist es notwendig die Standortbestimmung bei dem Vermessen zu verbessern. Dies kann dadurch erreicht werden, mehrere längere Messungen mit dem Apooge-D oder vergleichbaren Messgeräten durchzuführen. Die systematische und genaue Erfassung der Parkplätze ist dadurch allerdings mit bürokratischen Aufwand verbunden. Bei der Erfassung der Parkplätze ist es sinnvoll ganze Straßen, oder Abschnitte längerer Straßen zu erfassen. Dafür müssten die Parkplätze eine längere Zeit, für die Zeit der Erfassung aller dieser Parkplätze frei sein. Von daher ist eine temporäre Sperrung der Parkplätze, für die Dauer der Erfassung nötig.

7.2.2 Position des Fahrzeuges

Für die Bestimmung des Standortes des Fahrzeuges im Straßenverkehr gibt es verschiedene Ansätze welche als Lösung verwendet und ausgebaut werden können. Dies könnte wie in der Arbeit [51] eine durchgehende Standortbestimmung des Fahrzeuges anhand der Erkennung der Verkehrschilder durchgeführt werden.

7.2.3 Erkennung von Objekten im Sichtfeld vor den Parkplätzen

Um eine bessere Erkennung von True-Positive Parkplätzen zu erreichen, ist eine Erkennung von Objekten innerhalb des Sichtfeldes, wie zum Beispiel Büsche und Fahrzeuge, wichtig. Dadurch kann die fehlerhafte Auswertung der LiDAR-Daten, welche den Parkplatz verdecken, vermieden werden.

Literaturverzeichnis

[1] CRF, G. V.: LDM API and Usage Reference. (2010), S. 11–12

[2] CRF, G. V.: LDM API and Usage Reference. (2010), S. 13

[3] (CRF-CSST), Luisa Andreone & Roberto Brignolo & Sergio Damiani & Fulvio Sommariva & Giulio Vivo (CRF) & Stefano M.: SAFESPOT Final Report – Public version. (2010)

[4] Dauni, P ; Firdaus, M ; Asfariani, R ; Saputra, M ; Hidayat, A ; Zulfikar, W: Implementation of Haversine formula for school location tracking. In: *Journal of Physics: Conference Series* 1402 (2019), 12, S. 077028

[5] Esri: *ArcGIS API for JavaScript: Key features.* 2022. – URL https://developers.arcgis.com/javascript/latest/key-features/. – Zugriffsdatum: 2022-04-11

[6] Esri: *ArcGIS API for JavaScript: Layers and data.* 2022. – URL https://developers.arcgis.com/javascript/latest/layers-and-data/. – Zugriffsdatum: 2022-04-11

[7] Esri: *ArcGIS REST APIs: Overview.* 2022. – URL https://developers.arcgis.com/rest/enterprise-administration/server/overview.htm. – Zugriffsdatum: 2022-04-11

[8] Esri: *Mapping APIs and location services.* 2022. – URL https://developers.arcgis.com/documentation/mapping-apis-and-services/data-hosting/tutorials/tools/import-data-as-a-feature-layer/. – Zugriffsdatum: 2022-04-11

[9] e.V., FOSSGIS: *Open Street Map Deutschland.* 2022. – URL https://www.openstreetmap.de/. – Zugriffsdatum: 2022-05-18

[10] Foundation, Open Source G.: *Linux binary.* 2022. – URL https://docs.geoserver.org/latest/en/user/installation/linux.html. – Zugriffsdatum: 2022-04-23

[11] Foundation, Open Source G.: *What is Geoserver?* 2022. – URL https://geoserver.org/about/. – Zugriffsdatum: 2022-04-11

[12] Group, The PostgreSQL Global D.: *PostgreSQL: The World's Most Advanced Open Source Relational Database).* 2018. – URL https://www.postgresql.org/. – Zugriffsdatum: 2022-05-18

[13] Group, The PostgreSQL Global D.: *Linux downloads (Ubuntu).* 2022. – URL https://www.postgresql.org/download/linux/ubuntu/. – Zugriffsdatum: 2022-04-23

[14] Hamburg, Landesrecht: *Verordnung über den Bau und Betrieb von Garagen und offenen Stellplätzen: Garagenverordnung (GarVO).* 2012. – URL https://epub.sub.uni-hamburg.de/epub/volltexte/2014/34508/pdf/garagenverordnung_garvo.pdf. – Zugriffsdatum: 2022-04-23

[15] Inc, Velodyne L.: *Velodyne LiDAR HDL-32-E High Resolution Real-Tiem 3D LiDAR Sensor.* 2017. – URL https://pdf.aeroexpo.online/pdf/velodyne/hdl-32e/176220-7835.html. – Zugriffsdatum: 2022-04-23

[16] Infrastruktur, Bundesministerium für Verkehr und digitale: *Autobranche in "ganz schwerem Fahrwasser".* 2018. – URL https://web.archive.org/web/20180730202532/https://www.bmvi.de/SharedDocs/DE/RedenUndInterviews/2017/VerkehrUndMobilitaet/dobrindt-interview-straubinger-tagblatt-09092017.html. – Zugriffsdatum: 2022-04-18

[17] Institute, European Telecommunications S.: Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM), Rationale for and guidance on standardization. (2011)

[18] Kords, Martin: *Anzahl zugelassener Pkw in Deutschland von 1960 bis 2022.* 2022. – URL https://de.statista.com/statistik/daten/studie/12131/umfrage/pkw-bestand-in-deutschland/. – Zugriffsdatum: 2022-04-18

[19] Labor Geoinformatik, HTW D.: *VirtualBox, PostgreSQL und PostGIS Installation.* 2019. – URL https://geoinformatik.htw-dresden.

de/anleitungen/Master/DBT/Arbeitsanleitungen/VirtualBox_PostgreSQL.html#start. – Zugriffsdatum: 2022-04-26

[20] Ltd, Canonical: *Ubuntu 18.04.6 LTS (Bionic Beaver)*. 2018. – URL https://ubuntu.com/18.04. – Zugriffsdatum: 2022-05-18

[21] Ltd, Canonical: *Ubuntu 16.04 LTS (Xenial Xerus)*. 2022. – URL https://ubuntu.com/16-04. – Zugriffsdatum: 2022-05-18

[22] OpenStreetMap-Wiki: *DE:PBF Format*. 2017. – URL https://wiki.openstreetmap.org/wiki/DE:PBF_Format. – Zugriffsdatum: 2022-04-10

[23] OpenStreetMap-Wiki: *DE:OSM XML*. 2018. – URL https://wiki.openstreetmap.org/wiki/DE:OSM_XML. – Zugriffsdatum: 2022-04-10

[24] OpenStreetMap-Wiki: *OSM file formats*. 2020. – URL https://wiki.openstreetmap.org/wiki/OSM_file_formats. – Zugriffsdatum: 2022-04-10

[25] OpenStreetMap-Wiki: *DE:GIS software*. 2021. – URL https://wiki.openstreetmap.org/wiki/DE:GIS_software. – Zugriffsdatum: 2022-04-10

[26] OpenStreetMap-Wiki: *DE:OpenStreetMap benutzen*. 2021. – URL https://wiki.openstreetmap.org/wiki/DE:OpenStreetMap_benutzen. – Zugriffsdatum: 2022-04-10

[27] OpenStreetMap-Wiki: *OSM JSON*. 2021. – URL https://wiki.openstreetmap.org/wiki/OSM_JSON. – Zugriffsdatum: 2022-04-10

[28] OpenStreetMap-Wiki: *FAQs: Fragen und Antworten*. 2022. – URL https://www.openstreetmap.de/faq/#was-ist-openstreetmap. – Zugriffsdatum: 2022-04-10

[29] OSM, Learn: *Getting OSM Data*. 2016. – URL https://learnosm.org/en/osm-data/getting-data/. – Zugriffsdatum: 2022-04-10

[30] POSTGIS: *About PostGIS*. 2022. – URL https://postgis.net/. – Zugriffsdatum: 2022-04-11

[31] POSTGIS: *About PostGIS*. 2022. – URL https://postgis.net/. – Zugriffsdatum: 2022-05-18

[32] POSTGIS: *Installation*. 2022. – URL https://postgis.net/install/#binary-installers. – Zugriffsdatum: 2022-04-23

[33] Reif, Konrad: *Fahrstabilisierungssysteme und Fahrerassistenzsysteme.* Robert Bosch GmbH / Vieweg + Teubner Verlag / Springer, 2010. – 156 S

[34] Reif, Konrad: *Fahrstabilisierungssysteme und Fahrerassistenzsysteme.* Robert Bosch GmbH / Vieweg + Teubner Verlag / Springer, 2010. – 152 – 156 S

[35] Reisdorf, P. ; Auerswald, A. ; Wanielik, G.: Local Dynamic Map als modulares Software Framework für Fahrerassistenzsysteme. In: *Advances in Radio Science* 13 (2015), 11, S. 205

[36] Reisdorf, P. ; Auerswald, A. ; Wanielik, G.: Local Dynamic Map als modulares Software Framework für Fahrerassistenzsysteme. In: *Advances in Radio Science* 13 (2015), 11, S. 203–207

[37] Robotics, Open: *Package Summary: sbg_driver.* 2021. – URL http://wiki.ros.org/sbg_driver. – Zugriffsdatum: 2022-04-23

[38] Robotics, Open: *ROS- Robot Operating System.* 2021. – URL https://www.ros.org/. – Zugriffsdatum: 2022-05-18

[39] Robotics, Open: *ROS Kinetic Kame.* 2021. – URL http://wiki.ros.org/kinetic. – Zugriffsdatum: 2022-05-18

[40] Robotics, Open: *Ubuntu install of ROS Kinetic.* 2021. – URL http://wiki.ros.org/kinetic/Installation/Ubuntu. – Zugriffsdatum: 2022-05-03

[41] Shimada, Hideki ; Yamaguchi, Akihiro ; Takada, Hiroaki ; Sato, Kenya: Implementation and Evaluation of Local Dynamic Map in Safety Driving Systems. In: *Journal of Transportation Technologies* 5 (2015), S. 102–112

[42] Standardization, International O. for: Intelligente Verkehrssysteme – Kooperative ITS – Lokale dynamische Karten (ISO 18750:2018); Englische Fassung EN ISO 18750:2018. (2018)

[43] Standardization, International O. for: Intelligente Verkehrssysteme – Kooperative ITS – Lokale dynamische Karten (ISO 18750:2018); Englische Fassung EN ISO 18750:2018. (2018), S. 4

[44] Systems, SBG: *Apogee Series.* 2020. – URL https://www.sbg-systems.com/products/apogee-series-high-accuracy-ins-gnss/. – Zugriffsdatum: 2022-05-18

[45] SYSTEMS, SBG: *ULTIMATE ACCURACY MEMS Inertial Navigation System*. 2021. – URL https://www.sbg-systems.com/wp-content/uploads/Apogee_Marine_Series_Leaflet.pdf. – Zugriffsdatum: 2022-04-23

[46] V., Geschäftsstelle der Teststrecke für automatisiertes und vernetztes Fahren Hamburg c/o ITS mobility e.: *Bild der Teststrecke*. 2022. – URL https://tavf.hamburg/fileadmin/templates/images/ani_karte.gif. – Zugriffsdatum: 2022-04-23

[47] VATER, Tobias: *Inline-Auswertungen und -Korrektur multipler, komplexer, fusionierter, bildgebender Sensoren für das automatisierte Fahren*. Bachelor Arbeit. 04 2018

[48] VELODYNE LIDAR, Inc.: *HDL-32E*. 2021. – URL https://velodynelidar.com/products/hdl-32e/. – Zugriffsdatum: 2022-05-18

[49] VENESS, Chris: *Calculate distance, bearing and more between Latitude/Longitude points*. 2022. – URL http://www.movable-type.co.uk/scripts/latlong.html. – Zugriffsdatum: 2022-05-18

[50] VERKEHRSWESEN ARBEITSGRUPPE STRASSENENTWURF, Forschungsgesellschaft für Strassen-und: *Empfehlungen für Anlagen des ruhenden Verkehrs EAR 05*. Forschungsgesellschaft für Straßen- und Verkehrswesen e.V., 2005. – 28 S

[51] WENZEL, Niklas F.: *Automatisierte Erstellung von dynamischen Karten für das Autonome Fahren in komplexen, urbanen Umgebungen*. Bachelor Arbeit. 11 2019

A Anhang

A.1 Postgres und PostGIS Installation

Zuerst muss die PostgreSQL installiert und angelegt werden, damit die PostGIS-Erweiterung von der Datenbank genutzt werden kann. Zur Installation von PostgreSQL wird zuerst die Konfiguration des Dateienverzeichnisses angelegt,
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main» /etc/apt/sources.list.d/pgdg.list'
Dann der Signatur-Schlüssel importiert,
wget –quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
die Paketenliste geupdatet
sudo apt-get update
und anschließend PostgreSQL installiert: [13]
sudo apt-get -y install postgresql Zur Installation der PostGIS-Erweiterung muss die vorhandene PostgreSQL- und die gewünschte PostGIS-Version bekannt sein, damit für den folgenden Befehl die Versionsnummern genutzt werden können. In diesem Fall ist es folgender Befehl:
sudo apt install postgis postgresql-14-postgis-3

A.2 Quellcode vom Package *sbg_rosmsg*

A.2.1 Datei: sbg_rosmsg.cpp

```
// File header
#include "sbg_rosmsg.h"

ros::Publisher pub_lat;
```

```
ros::Publisher pub_long;
ros::Publisher pub_time;
ros::Publisher pub_orientation;
std_msgs::UInt32 time_stamp;
std_msgs::Time utc_time_stamp_pub;

// Compare gps_pos and gps_hdt tiemstamps,
// to ensure to get the same measurement
void compare_time_stamps(uint32_t gps_hdt_time, float orientation){
        if(time_stamp.data == gps_hdt_time){
                //ROS_INFO("TIME");
                ros::param::set("/orientation", orientation);
                ros::param::set("/latitude", gps_pos_x);
                ros::param::set("/longitude", gps_pos_y);
                ros::param::set("/gps_time_sec", utc_time_sec);
                ros::param::set("/gps_time_nsec", utc_time_nsec);
        }

}

void gps_pos_cb(const sbg_driver::SbgGpsPos data_gps){
        utc_time_sec = data_gps.header.stamp.sec;
        utc_time_nsec = data_gps.header.stamp.nsec;
        utc_time_stamp_pub.data = data_gps.header.stamp;
        gps_pos_x = data_gps.position.x;
        gps_pos_y = data_gps.position.y;
        time_stamp.data = data_gps.time_stamp;
}

void gps_hdt_cb(const sbg_driver::SbgGpsHdt data){
        float orientation = data.true_heading;
        uint32_t gps_hdt_time = data.time_stamp;
        compare_time_stamps(gps_hdt_time, orientation);
}

void gps_utc_cb(const sbg_driver::SbgUtcTime utc_time){
```

```
        utc_time_sec = utc_time.header.stamp.sec;
        utc_time_nsec = utc_time.header.stamp.nsec;
        utc_time_stamp_pub.data = utc_time.header.stamp;
}

int main(int argc, char *argv[]) {
        ros::init(argc, argv, "sbg_rosmsg");
        ros::NodeHandle n;
        ros::Subscriber sub_pos = n.subscribe("/gps_pos",
            1000, gps_pos_cb);
        ros::Subscriber sub_hdt = n.subscribe("/gps_hdt",
            1000, gps_hdt_cb);
        ros::spin();
}
```

A.2.2 Datei: sbg_rosmgs.h

```
#ifndef CATKIN_WS_SBG_ROSMSG_H
#define CATKIN_WS_SBG_ROSMSG_H

// ROS Includes
#include "ros/ros.h"
#include <std_msgs/Float32.h>
#include <std_msgs/UInt32.h>
#include <std_msgs/Time.h>
// SBG Includes
#include "sbg_driver/SbgGpsPos.h"
#include "sbg_driver/SbgGpsHdt.h"
#include "sbg_driver/SbgUtcTime.h"

double gps_pos_x, gps_pos_y;
uint32_t gps_pos_time;
int utc_time_sec, utc_time_nsec;

void compare_time_stamps(uint32_t gps_hdt_time, float orientation);
```

```
void gps_pos_cb(const sbg_driver::SbgGpsPos data);
void gps_hdt_cb(const sbg_driver::SbgGpsHdt data);
void gps_utc_cb(const sbg_driver::SbgUtcTime utc_time);

#endif // CATKIN_WS_SBG_ROSMSG_H
```

A.3 Quellcode vom Package *parking_lot_information*

A.3.1 Datei: startRoutine.py

```
#!/usr/bin/env python
from readVelodynePcl import startLidarMeasurment, checkGpsTime
from dataClass import dataClass
import rospy

def main():
        while True:
                while rospy.get_param("/gps_time_sec") == 0:
                        pass
                dataMeasurment = dataClass()
                startLidarMeasurment()
                dataMeasurment.setLidarTime()
                dataMeasurment.setGpsTime()
                dataMeasurment.setPointCloud2Array()
                dataMeasurment.setCarGpsPos()
                dataMeasurment.setOrientation()
                dataMeasurment.setDeltaTime()
                dataMeasurment.setTableResults()
                while dataMeasurment.done == False:
                        pass
                dataMeasurment.checkParkinglotsData()
                if dataMeasurment.checkDelta == True:
                        dataMeasurment.updateParkinglots()
                        rospy.sleep(2.448)
```

```
if __name__ == '__main__':
        main()
```

A.3.2 Datei: dataClass.py

```
from readVelodynePcl import getLidarTime, getGpsTime,
    getGpsPos, getLidarSequence
from getParkinglotsInRange import databaseRequest
from updateParkinglotsStatus import updateDatabase
from checkParkinglots import checkParkinglotStatus,
    getConfidence, getParkinglotsStatusArray
import rospy

class dataClass:
        def __init__(self):
                self.gpsTime = 0
                self.lidarTime = 0
                self.carGpsPosLong = 0.0
                self.carGpsPosLat = 0.0
                self.carOrientation= 0.0
                self.parkinglotsIDArray = []
                self.parkinglotsLongArray = []
                self.parkinglotsLatArray = []
                self.parkinglotsStatusArray = []
                self.parkinglotConfidenceArray = []
                self.pointCloud2Array = []
                self.deltaTime = 0
                self.tableResults = []
                self.done = False
                self.checkDelta = False

        def setLidarTime(self):
                self.lidarTime = getLidarTime()

        def setGpsTime(self):
```

```
        self.gpsTime = getGpsTime()

    def setPointCloud2Array(self):
        self.pointCloud2Array = getLidarSequence()

    def setCarGpsPos(self):
        self.carGpsPosLong, self.carGpsPosLat =
            getGpsPos()

    def setOrientation(self):
        self.carOrientation =
            rospy.get_param("/orientation")

    def setDeltaTime(self):
        self.deltaTime =
            abs(self.gpsTime - int(str(self.lidarTime)))
        self.done = True

    def setTableResults(self):
        self.tableResults =
            databaseRequest(self.carGpsPosLong,
                self.carGpsPosLat)
        for i in range(0, len(self.tableResults)):
            self.parkinglotsIDArray.append
                (self.tableResults[i][0])
            self.parkinglotsLongArray.append
                (self.tableResults[i][1])
            self.parkinglotsLatArray.append
                (self.tableResults[i][2])

    def checkParkinglotsData(self):
        if self.deltaTime < 100000000:
            self.checkDelta = True
            if not self.carOrientation == 0:
            checkParkinglotStatus
                (self.carGpsPosLong, self.carGpsPosLat,
```

```
                                    self.parkinglotsLongArray,
                                        self.parkinglotsLatArray,
                                            self.pointCloud2Array,
                                                self.carOrientation,
                                                    self.parkinglotsIDArray)
                            self.setConfidence()
                            self.setParkinglotsStatus()

        def setConfidence(self):
                print(getConfidence())
                self.parkinglotConfidenceArray = getConfidence()

        def setParkinglotsStatus(self):
                self.parkinglotsStatusArray =
                    getParkinglotsStatusArray()

        def updateParkinglots(self):
                self.done = updateDatabase
                    (self.parkinglotsIDArray,
                        self.parkinglotsStatusArray, self.gpsTime,
                            self.parkinglotConfidenceArray)
```

A.3.3 Datei: readVelodynePcl.py

```
#!/usr/bin/env python
import rospy
import rosbag
import roslib.message
import sys
import math
import ros_numpy
import numpy as np
from sensor_msgs.msg import PointCloud2

# Global variables
```

```
lidarTime = 0
points = []
carGpsPosLong = 0
carGpsPosLat = 0
gpsTime = 0

# Function to subscribe the /velodyne_points topic
def startLidarMeasurment():
        rospy.init_node('listener_lidar', anonymous = True)
        msg = rospy.wait_for_message('/velodyne_points',
            PointCloud2, timeout=None)
        setGpsPosAndTime()
        setPointCloud2Data(msg)

# Function to set the time stamp of the lidar measurment
def setLidarTime(ros_cloud):
        global lidarTime
        lidarTime = ros_cloud.header.stamp
        # To check if there is data
        if not lidarTime == None:
                return True

def setGpsPosAndTime():
        global carGpsPosLong, carGpsPosLat, gpsTime
        carGpsPosLong = rospy.get_param("/longitude")
        carGpsPosLat = rospy.get_param("/latitude")
        gpsTimeSec = rospy.get_param("/gps_time_sec")
        gpsTimeNsec = rospy.get_param("/gps_time_nsec")
        gpsTime = int('{}{}'.format(gpsTimeSec, gpsTimeNsec))
        checkGpsTime()

def checkGpsTime():
        global gpsTime
        digits = int(math.log10(gpsTime))+1
        if digits == 18:
                gpsTime = gpsTime * 10
```

```
def getGpsPos():
        global carGpsPosLong, carGpsPosLat
        return carGpsPosLong, carGpsPosLat

def getGpsTime():
        global gpsTime
        return gpsTime

# Function to get the time stamp of the lidar measurment
def getLidarTime():
        global lidarTime
        return lidarTime

# Function to set the point cloud data of the lidar measurment
def setPointCloud2Data(ros_cloud):
        global points
        if setLidarTime(ros_cloud) == False:
                print("Erro LiDAR Time")
        pc = ros_numpy.numpify(ros_cloud)
        points = np.zeros((pc.shape[0],3))
        points[:,0]=pc['x']
        points[:,1]=pc['y']
        points[:,2]=pc['z']
        if not (points == 0).sum() == len(points):
                return True

# Function to get the point cloud data of the lidar measurment
def getLidarSequence():
        global points
        return points
```

A.3.4 Datei: getParkinglotsInRange.py

```
from psycopg2 import connect, Error
```

```
tableName = "parkinglots"
radius = 17

def databaseRequest(gpsLon, gpsLat):
        tableResult = None

        sqlString = "SELECT parkinglot_id, longitude, latitude
            FROM {0} WHERE ST_DistanceSphere(ST_MakePoint
                (longitude,latitude), ST_MakePoint({1},{2}))
                    <= {3};".format(tableName, gpsLon, gpsLat,
                        radius)
        debugString = "SELECT * FROM parkinglots;"
        try:
                conn = connect(
                        dbname = "parkinglotsDatabase",
                        user = "postgres",
                        host = '10.0.2.15',
                        password = "123456",
                        connect_timeout = 3
                )
                cur = conn.cursor()
                try:
                        cur.execute(sqlString)
                        conn.commit()
                        tableResult = cur.fetchall()

                except Error as err:
                        print("Psycopg2 cursor error:", err)

        except (Exception, Error) as err:
                print ("Psycopg2 connect error:", err)
                conn = None
                cur = None

        return tableResult
```

A.3.5 Datei: updateParkinglotsStatus.py

```
from psycopg2 import connect, Error
from datetime import datetime, timedelta

tableName = "parkinglots"

def updateDatabase(parkinglotsIDArray, parkinglotsStatusArray,
        gpsTime, confidenceArray):
        datetimeGps = datetime.fromtimestamp(gpsTime // 1000000000)
        datetimeGps = datetimeGps.strftime('%H:%M:%S')
        for i in range(0, len(parkinglotsIDArray)):
                sqlString = "UPDATE {0} SET time_of_status = '{1}',
                    status = '{2}', confidence = {3} WHERE parkinglot_id
                        = '{4}';".format(tableName, datetimeGps,
                            parkinglotsStatusArray[i], confidenceArray[i],
                                parkinglotsIDArray[i])
                try:
                        conn = connect(
                                dbname = "parkinglotsDatabase",
                                user = "postgres",
                                host = '10.0.2.15',
                                password = "123456",
                                connect_timeout = 3
                        )
                        cur = conn.cursor()
                        try:
                                cur.execute(sqlString)
                                conn.commit()

                        except (Exception, Error) as err:
                                print("Update: Error: ", err)
                                exit()

                except (Exception, Error) as err:
                        print ("Psycopg2 connect error:", err)
```

```
                        conn = None
                        cur = None
                        exit()

        return True
```

A.3.6 Datei checkParkinglots.py

```
import numpy as np
from math import radians, cos, sin, asin, sqrt, pi

confidence = []
parkinglotsStatusArray = []

def checkParkinglotStatus(carGpsPosLong, carGpsPosLat,
    parkinglotsLongArray, parkinglotsLatArray,
        pointCloud2Array, carOrientation,
            parkinglotsIDArray):
        global confidence
        global parkinglotsStatusArray
        confidence = []
        parkinglotsStatusArray = []
        radius = 1
        debugPoint = []
        dc = []
        for i in range(0, len(parkinglotsLongArray)):
                print(parkinglotsIDArray[i])
                x, y = calcualteDistanceInXY(carGpsPosLong,
                    carGpsPosLat, parkinglotsLongArray[i],
                        parkinglotsLatArray[i], carOrientation)
                pointsInRadius = np.array([])
                pointsOnGround = 0
                offSetMin = 1.598 #1.692
                offSetMax = 2.162 #2.068
                # Check points within radius
```

```
                for j in range(0, np.size(pointCloud2Array,0)):
                        dx = x-pointCloud2Array[j,0]
                        dy = y-pointCloud2Array[j,1]
                        if ((dx**2)+(dy**2) < radius**2):
                                pointsInRadius = np.append
                                    (pointsInRadius,
                                        pointCloud2Array[j,2])

                # Check for points on ground
                for k in range(len(pointsInRadius)):
                        #print(pointsInRadius[i])
                        if (-pointsInRadius[k] > offSetMin and -
                            pointsInRadius[k] < offSetMax):
                                pointsOnGround = pointsOnGround + 1

                # Calculate confidence
                confidenceCal = 0
                if len(pointsInRadius) > 0:
                        confidenceCal = (float(pointsOnGround)/
                            float(len(pointsInRadius)))
                        confidence.append(confidenceCal)
                        # Update Status
                        if confidenceCal > 0.5:
                                parkinglotsStatusArray.
                                    append("Frei")
                        else:
                                parkinglotsStatusArray.
                                    append("Belegt")
                else:
                        confidence.append(-1)
                        parkinglotsStatusArray.append("NB")

def get_bearing(lat1, long1, lat2, long2):
        dLon = (long2 - long1)
        x = cos(radians(lat2)) * sin(radians(dLon))
        y = cos(radians(lat1)) * sin(radians(lat2)) -
```

```
            sin(radians(lat1)) * cos(radians(lat2)) *
                cos(radians(dLon))
        brng = np.arctan2(x,y)
        brng = np.degrees(brng)
        return brng

def calcualteDistanceInXY(carGpsPosLong, carGpsPosLat,
        parkinglotsLong, parkinglotsLat, carOrientation):
        distance = haversine((carGpsPosLat, carGpsPosLong),
            (parkinglotsLat, parkinglotsLong))
        bearing = get_bearing(carGpsPosLat, carGpsPosLong,
            parkinglotsLat, parkinglotsLong)
        parkinglotX = distance * sin(radians(bearing + 180
            - carOrientation))
        parkinglotY = distance * cos(radians(bearing + 180
            - carOrientation))
        return parkinglotX, parkinglotY

# To handle a syntax in a function, which will be imported
# with the haversine module
# Copy only that function, which is needed and doesn't throw
# a syntax error while importing that module
def haversine(point1, point2):
        # Earth Radius WGS84
        earthRadius = 6378.137
        lat1, lng1 = point1
        lat2, lng2 = point2
        lat1 = radians(lat1)
        lng1 = radians(lng1)
        lat2 = radians(lat2)
        lng2 = radians(lng2)
        lat = lat2 - lat1
        lng = lng2 - lng1
        d = sin(lat * 0.5) ** 2 + cos(lat1) * cos(lat2) *
            sin(lng * 0.5) ** 2
        return 2 * earthRadius * asin(sqrt(d)) * 1000
```

```
def getConfidence():
        global confidence
        return confidence

def getParkinglotsStatusArray():
        global parkinglotsStatusArray
        return parkinglotsStatusArray
```

A.4 Launch-Datei: launch-file.launch

In der Launch-Datei kann die ROS-Bag-Datei für die Auswertung beliebig geändert werden.

```
<launch>
        <param name="gps_time_sec" type="int" value="0"/>
        <param name="gps_time_nsec" type="int" value="0"/>
        <param name="longitude" type="double" value="0.0"/>
        <param name="latitude" type="double" value="0.0"/>
        <param name="orientation" type="double" value="0.0"/>

        <node pkg="rosbag" type="play" name="rosbagPlayer"
            output="screen" args="--clock /home/roskin/Desktop/
                Ros-Data/2022-04-26-13-11-09_98.bag"/>
        <node pkg="parking_lot_information" type="startRoutine.py"
            name="StartRoutine" output="screen"/>
        <node pkg="sbg_rosmsg" type="sbg_rosmsg"
            name="sbg_rosmsg_driver" output="screen"/>
</launch>
```

A.5 Datei: webmap.html

```
<!doctype html>
<html lang="en">
```

```
<head>
        <meta charset="utf-8">
        <link rel="stylesheet" href="https://cdn.jsdelivr.net/
            gh/openlayers/openlayers.github.io@master/en/
                v6.13.0/css/ol.css" type="text/css">
        <style>
                .map {
                        height: 700px;
                        width: 100%;
                        float: left;
                }
                .btn {
                        height: 100px;
                        width: 100%;
                        float: bottom;
                }
        </style>
        <script src="https://cdn.jsdelivr.net/gh/openlayers/
            openlayers.github.io@master/en/v6.13.0/build/ol.js">
                </script>
        <title>Test-Map</title>
</head>
<body>
        <h2>Parkplaetze</h2>
        <div id="map" class="map"></div>
        <footer>
                <label for="name", align="top">
                    Den gewuenchsten Standort eingeben
                        (Longitude, Latitude):</label>
                <input type="text" id="userInputLong" name=
                    "userInputLong", align="top">
                <input type="text" id="userInputLat" name=
                    "userInputLat", align="top">
                <button onClick="nearestParkinglot();", align=
                    "top">Naechstgelegnder Parkplatz</Button>
                <label for="name", align="top">Den gewuenchsten
```

```
            Radius eingeben(in Metern):</label>
        <input type="text" id="userInputRadius" name=
            "userInputRadius", align="top">
        <button onClick="parkinglotsInRange();", align=
            "top">Parkplaetze im Radius</Button>
</footer>
<script>
var untiled = new ol.layer.Image({
        source: new ol.source.ImageWMS({
                ratio: 1,
                url: 'http://localhost:8080/
                    geoserver/Parkplatz-Anzeige/wms',
                params: {
                        'VERSION': '1.1.1',
                        "STYLES": '',
                    "LAYERS": 'Parkplatz-Anzeige:
                                        parkinglots',
                        "exceptions": 'application/
                            vnd.ogc.se_inimage',
                }
        })
});
var tiled = new ol.layer.Tile({
        visible: false,
        source: new ol.source.TileWMS({
                url: 'http://localhost:8080/geoserver/
                    Parkplatz-Anzeige/wms',
        params: {//'FORMAT': format,
        'VERSION': '1.1.1',
        tiled: true,
        "STYLES": '',
        "LAYERS": 'Parkplatz-Anzeige:parkinglots',
        "exceptions": 'application/
            vnd.ogc.se_inimage',
        tilesOrigin: 9.965 + "," + 53.545
                }
```

```
            })
        });
    var parkinglots = new ol.layer.Image({
        source: new ol.source.ImageWMS({
                url: 'http://localhost:8080/geoserver/
                    Parkplatz-Anzeige/wms',
                params:{
                "LAYERS":'Parkplatz-Anzeige:
                                    parkinglots',
            },
                imageLoadFunction: function(image, src) {
                image.getImage().src = src;
            }
        })
    });
    var map = new ol.Map({
        target: 'map',
        layers: [
            tiled,
            untiled,
            new ol.layer.Tile({
                source: new ol.source.OSM()
            }),
        ],
        view: new ol.View({
            center: ol.proj.fromLonLat([10, 53.55]),
            zoom: 13
        })
    });
    map.render();
    function parkinglotsInRange(){
        var longitude = document.getElementById
            ("userInputLong").value;
        var latitude = document.getElementById
            ("userInputLat").value;
        var radius = document.getElementById
```

```
            ("userInputRadius").value;
        var parkinglotsInRange = new ol.layer.
            Image({
            source: new ol.source.ImageWMS({
                url: 'http://localhost:8080/
                    geoserver/Parkplatz-Anzeige/wms?
                    service=WMS&version=1.1.0&request=
                    GetMap&layers=Parkplatz-Anzeige%3
                    AparkinglotsInRange&bbox=9.5%2C53.
                    0%2C10.5%2C54.0&width=768&height=
                    768&srs=EPSG%3A404000&styles=&format
                    =application/openlayers&viewparams=
                    longPos:longitude;latPos:latitude;radius:radius',
                params:{
                    "VIEWPARAMS": 'longPos:longitude;
                        latPos:latitude; radius:radius',
                    "FORMAT": format,
                    "LAYERS": 'Parkplatz-Anzeige:
                        parkinglotsInRange',
                    "exceptions": 'application/
                        vnd.ogc.se_inimage'
                },
                serverType:'geoserver',
                imageLoadFunction: function(image, src) {
                    image.getImage().src = src;
                }
            })
        });
        map.addLayer(parkinglotsInRange);
        map.render();
    }
    function nearestParkinglot(){
        var longitude = document.getElementById
            ("userInputLong").value;
        var latitude = document.getElementById
            ("userInputLat").value;
```

```
                var nearestParkinglot = new ol.layer.Image({
                    source: new ol.source.ImageWMS({
                        url: 'http://localhost:8080/
                        geoserver/Parkplatz-Anzeige/wms?
                        service=WMS&version=1.1.0&request=
                        GetMap&layers=Parkplatz-Anzeige%
                        3AnearestParkinglot&bbox=9.5%2C53.
                        0%2C10.5%2C54.0&width=768&height=
                        768&srs=EPSG%3A4326&styles=&format
                        =application/openlayers&viewparams=
                        longPos:longitude;latPos:latitude5',
                        params:{
                            "VIEWPARAMS": 'longPos:longitude;
                                latPos:latitude',
                            "LAYERS": 'Parkplatz-
                                Anzeige:nearestParkinglot',
                            "exceptions": 'application
                                /vnd.ogc.se_inimage'
                        },
                        serverType:'geoserver',
                        imageLoadFunction: function(image, src) {
                            image.getImage().src = src;
                        }
                    })
                                });
                map.addLayer(nearestParkinglot);
                map.render();
            }
        </script>
    </body>
</html>
```

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

[redacted]

Ort	Datum	Unterschrift im Original