Projektart <Bachelorarbeit>

Studiengang <Fahrzeugtechnik>

an der Fakultät für <Fahrzeugsysteme und Produktion - 08>

der Technischen Hochschule Köln

Objektdetektion mit Datensegmentierung an einem LIDAR auf einem Single-Board-Computer mithilfe des Robot-Operating-System

vorgelegt von: Tim-Niklas Wennemann

Matrikel-Nr.: 11110993

eingereicht bei: Prof. Dr. Tom Tiltmann

Zweitgutachter/in: Prof. Dr. Chunrong Yuan

Köln, 31.07.2020

Technology
Arts Sciences
TH Köln

Inhaltsverzeichnis

Abkürzungs-/Symbolverzeichnis

α	Winkel zur Kreisklassifizierung zwischen $\overrightarrow{pg}$ und $\overrightarrow{pq}$
β	Winkel zur Kreisklassifizierung zwischen $\overrightarrow{pP}$ und $\overrightarrow{pq}$
c	Faktor zur Kreismittelpunktberechnung
d	Euklidischer Abstand
D	Abstand der Segmentzwischenpunkte zu der Gerade zwischen zwei Punkten
$Dmax$	Größter Abstand der Segmentzwischenpunkte zu der Gerade zwischen zwei Punkten
ε	Winkel zur Kreisklassifizierung zwischen $\overrightarrow{qg}$ und $\overrightarrow{qp}$
g	Segmentpunkt mit dem größten Abstand zu der Gerade zwischen zwei Punkten
g_s	Gespiegelter Punkt zu g im Rechteck
δ	Winkel zur Kreisklassifizierung zwischen $\overrightarrow{qP}$ und $\overrightarrow{qp}$
i	Jeweiliger Messpunkt
j	Jeweiliges Segment
l_x	Position Lidarsensor x-Koordinate Occupancy Grid
l_y	Position Lidarsensor y-Koordinate Occupancy Grid
L	Gerade zwischen zwei Punkten
λ	Parameterwert für Geradenbeschreibung zwischen zwei Punkten
m	Objektmittelpunkt
m_c	Objektmittelpunkt Kreis
m_L	Objektmittelpunkt Linie
m_r	Objektmittelpunkt Rechteck
$MinPts$	Minimale Anzahl Segmentpunkte DBSCAN
$\bar{\varphi}$	Gemittelter gemessener Winkel zur x-Achse
$\bar{R}$	Gemittelter gemessener Objektabstand
$\bar{a}$	Gemittelte gemessene Objektlänge
$\bar{b}$	Gemittelte gemessene Objektbreite
$\bar{D}$	Gemittelter gemessener Kreisradius
N	Anzahl der Blöcke
num	Anzahl der Datenpunkte zwischen Segment Start-, und Stopppunkt
O	Kreismittelpunkt ODAOA-Algorithmus
ODAOA-Algorithmus	Obstacel Detection and Obstacel Avoidance Algorithm
P	Messpunkt
p	Segment Startpunkt
φ	Messwinkel
φ_z	Objektorientierungswinkel um z-Achse
q	Segment Stopppunkt
R	Messpunkt Abstand Polarkoordinaten
r	Radius Kreis

RAM	Arbeitsspeicher
ROS	Robot Operating System
$\bar{R}_{qp}$	Durchschnittlicher Abstand des Start,- und Stopppunktes
σ_a	Standardabweichung gemessene Objektlänge
σ_b	Standardabweichung gemessene Objektbreite
σ_D	Standardabweichung gemessener Kreisradius
σ_φ	Standardabweichung gemessener Winkel zur x-Achse
σ_R	Standardabweichung gemessener Objektabstand
$\|S\|$	Strecke zwischen Start,- und Stopppunkt ODAOA-Algorithmus
T	Threshold
$\vec{L}$	Lotpunkt
$\vec{u}$	Richtungsvektor
x	Messpunkt x-Koordinate Kartesische Koordinaten
y	Messpunkt y-Koordinate Kartesische Koordinaten
θ	Messpunkt Winkel Polarkoordinaten
I	Erster Quadrant
II	Zweiter Quadrant
III	Dritter Quadrant
IV	Vierter Quadrant

Aufgabenstellung

Ziel dieser Bachelorarbeit ist es, die Daten einer durch einen 360° 2D Laserscanner erstellten Punktewolke auszuwerten. Dadurch sollen, durch die Wahl eines geeigneten Segmentierungs-/ und Klassifizierungsverfahren, Objekte in der Umgebung erkannt und entsprechend der Form klassifiziert werden.

Für die Implementierung sind dafür sowohl Softwarebibliotheken, sowie selbstgeschriebene Softwareprogramme in der Programmiersprache Python zugelassen. Die gesamte Implementierung soll dabei unter dem ROS Framework umgesetzt werden.

Als erster Teil der Arbeit werden dafür verschiedene Segmentierungsverfahren recherchiert und erläutert. Diese sollen dann anhand der unten folgenden Anforderungen untersucht werden. Aus der daraus resultierenden Bewertung soll ein für die Aufgabenstellung geeignetes Verfahren gewählt werden.

1. Dabei sollen mindestens die Objektgeometrien Rechteck und Kreis durch die Klassifizierung in der Umgebung erkannt und inklusive der Position ausgegeben werden. Außerdem ist die Information der jeweiligen Orientierung der Objekte erwünscht.

2. Ebenfalls gilt, dass eine unbekannte Anzahl von Objekten um den Sensor platziert und erfasst werden kann. Das gewählte Verfahren soll somit unabhängig von der Anzahl der Objekte arbeiten und erkennen, wie viele Objekte sich im Umfeld des Sensors befinden.

3. Die Umsetzung soll dabei auf einem Kleinstrechner (Raspberry Pi / Odroid XU4) realisiert werden. Aufgrund des stark limitierten Arbeitsspeichers soll die Rechenauslastung des gewählten Segmentierungsverfahren so gering wie möglich gehalten werden.

4. Das Verfahren eignet sich für die Auswertung einer 2D Punktewolke eines Laserscanners.

Nach der Auswahl eines geeigneten Verfahrens wird dieses im darauffolgenden Teil dieser Arbeit umgesetzt und ausgearbeitet. Über eine Versuchsreihe folgt anschließend noch eine Bewertung der Implementierung.

Ebenfalls soll der Aufbau von Sensor und Kleinstrechner robust miteinander verkabelt und verbaut werden.

Die Funktionsfähigkeit der selbstgeschriebenen Quelltexte ist mit Unit Tests sicherzustellen. Dafür werden jeweils drei Testfälle pro selbstgeschriebene Funktion angefertigt.

Einleitung

Das Thema des vollautonomen Fahrens ist heutzutage aus der Forschung zur modernen Fahrzeugtechnik nicht mehr wegzudenken. Während ein mittels Parkassistenzsystem selbstständig einparkendes Auto im Straßenverkehr keine Seltenheit mehr darstellt, werden sich einer aktuellen Studie zufolge vollständig „selbst fahrende“ Autos voraussichtlich frühestens ab 2040 endgültig durchsetzen. Doch gerade das enorme Potential der Technologie macht es für die Automobilunternehmen so reizvoll. Gesellschaftlich können ältere und leistungseingeschränkte Personen wieder besser eingebunden werden. Wirtschaftlich können Güter aufgrund von weniger Verkehrsstaus schneller und umweltschonender transportiert werden. Und auch der Aspekt der Sicherheit ist von entscheidender Bedeutung. Demnach können je nach Grad der Automatisierung die Unfallzahlen im Straßenverkehr deutlich reduziert werden, da ca. 90% aller Verkehrsunfälle auf menschliches Versagen zurückzuführen sind. Gerade deshalb steht die Entwicklung der bestmöglichen Technologie für die autonome Fortbewegung im Vordergrund. [27]

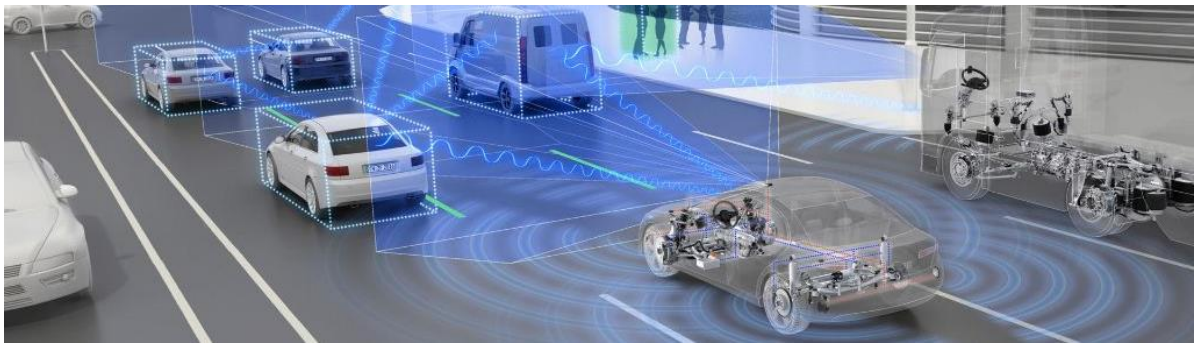

Abbildung 1: Autonomes Fahren im Straßenverkehr [29]

Damit sich das Fahrzeug ohne menschlichen Eingriff sicher im Straßenverkehr bewegen kann, steht logischerweise die Überwachung der Fahrzeugumgebung an vorderster Stelle. Um eine möglichst Fehlerfreie Analyse des Umfelds zu gewährleisten, ist die Auswahl der für die Anforderungen am besten geeignetsten Sensortechnik entscheidend. Neben fortschrittlicher Kameratechnik bieten moderne Abstandssensoren eine weitere Möglichkeit, Hindernisse in der Fahrzeugumgebung zu erkennen und dadurch mögliche Kollisionen zu vermeiden. Im Laufe der Jahre hat sich in der Fahrzeugtechnik besonders die Lidartechnik, bzw. der Einsatz von rotierenden Laserscannern zur Abstandsmessung in der Fahrzeugindustrie durchgesetzt. Die Idee der Weiterentwicklung von Systemen zur aktiven Sicherheit und den Bemühungen, das Fahren autonom zu gestalten, führte zu einer schnellen Entwicklung der Lidar-Technik und zu zahlreichen Neugründungen von Unternehmen. [28]

Neben dem Einsatz der richtigen Hardwaretechnik, ist auch die Auswahl eines geeigneten Algorithmus entscheidend. Denn nur durch die Entwickelung einer „künstlichen Intelligenz“, können die vom Sensor gemessenen Daten auch so weiterverarbeitet werden, dass potenzielle Gefahren in der Umgebung erkannt werden können und infolgedessen eine Entscheidung getroffen werden kann, wie sich das selbstständig fahrende Fahrzeug in der jeweiligen Situation fahrtechnisch zu verhalten hat. Aus diesem Grund wird sich im Umfang dieser Bachelorarbeit mit der Forschungsfrage auseinandergesetzt, wie aus den Abstandsmesswerten eines 360° Laserscanners potenzielle Objekte in der Umgebung erkannt werden können.

1 Stand der Technik

1.1 Robot Operating System

Das Robot Operating System (kurz ROS) ist ein Linux basiertes Open-Source Softwareframe zur Entwicklung von Robotersystemen. Verschiedene Tools wie eine 3D-Simulationsumgebung (rviz) oder Bibliotheken sollen bei der Erstellung von komplexen Robotersystemen als Unterstützung dienen. Das Robot Operating System ist Modular aufgebaut. Somit können die einzelnen Roboterkomponenten über sogenannte Themen (engl. *topics)* miteinander kommunizieren und Daten austauschen. Die Programmierung im ROS kann dabei in verschiedene Programmiersprachen wie Python, C++, Lisp, Java und Lua erfolgen.

Jegliche Programme und Funktionen, die in einem Robotersystem mit eingebunden werden, sind in sogenannten Workspace Ordner hinterlegt. Ein Workspace Ordner besteht dabei aus den beiden Ordnern *build* und *devel,* welche die Systemeigenschaften eines typischen ROS Workspace beinhalten und dem *src* Verzeichnis. In dem *src* Verzeichnis befinden sich die sogenannten Pakete, welche eine oder mehrere Softwarefunktionen beinhalteten. Pakete können dabei entweder selbst angelegt werden oder stehen in Form von beispielsweise Treiber- oder Filterpaketen online frei zur Verfügung. Über die einzelnen Pakete können dann verschiedene ROS-Prozesse ausgeführt werden. Innerhalb der Pakete befinden sich somit beliebig viele Programmcodes, welche als Knoten bezeichnet werden.

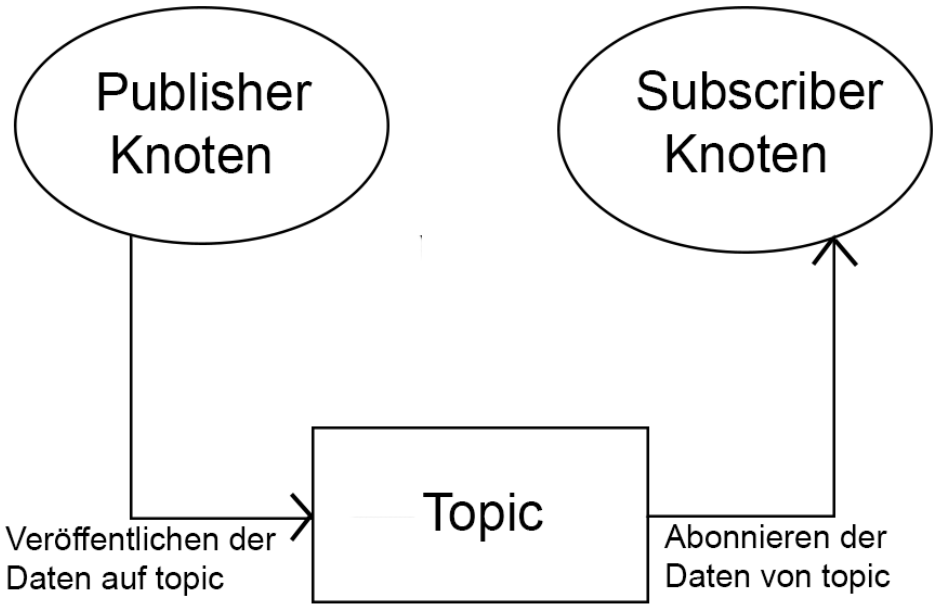

Abbildung 2: Knotenkommunikation über eine topic im ROS

Die Knoten haben dabei jeweils eine oder mehrere Funktionen und können innerhalb des ROS über topics miteinander kommunizieren. Ein Knoten kann dabei als *subsciber, publisher* oder beides fungieren. Wie in Abbildung 2 dargestellt, abonniert ein Knoten mit einer subsciber-Funktion die Daten zur Weiterverarbeitung von einer topic, auf die ein anderer Knoten mit der publisher-Funktion eben diese Daten veröffentlicht hat. Die gesamte Kommunikation unter den einzelnen Komponenten des ROS-Systems lässt sich über einen sogenannten *rqt-graph* visualisieren. [25]

1.2 RPLIDAR A1

Der in dieser Arbeit verwendete RPLIDAR A1 ist ein „low cost“ 360 Grad Laser Scanner (Lidar), hergestellt von SLAMTEC. Überwacht der Sensor einen 360 Grad Umfangswinkel, können Abstände mit einer Entfernung von bis zu 6 Metern gemessen werden. Bei einer Messfrequenz von 5,5 Hz kann dabei eine Punktewolke mit 360 Punkten erzeugt werden. Die maximale Messfrequenz des Sensors liegt bei 10 Hz. Die aus einer Messung resultierende Punktewolke kann zur Kartenerzeugung, Lokalisierung und zur Objekt- und Umgebungsmodellierung genutzt werden.

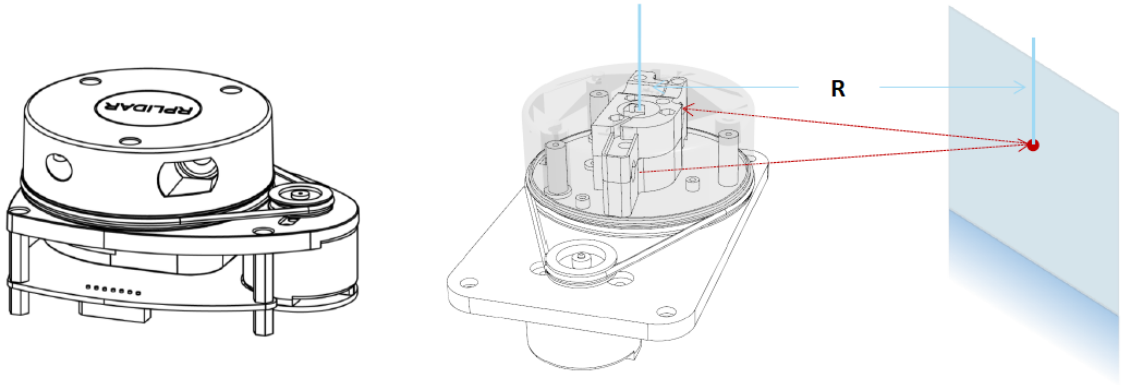

Abbildung 3: Darstellung des RPLIDAR (links); RPLIDAR Messprinzip (rechts) [32]

Die Messung des Sensors basiert auf dem Triangulations-Messprinzip, welches rechts in der Abbildung 3 dargestellt wird. Dafür nutzt der Sensor einen 3mW-Infrarotlaser mit einer Wellenlänge von 785 nm. Der ausgesendete Laserstrahl wird von der zu messenden Oberfläche reflektiert und vom Erfassungssystem des Sensors eingefangen. Über den Winkel zwischen dem ausgesendeten und dem reflektierten Laserstrahl kann dann mittels dem bekannten Abstand zwischen Sender- und Empfängerelektronik der Abstand d zum Sensormittelpunkt berechnet werden. Die gemessenen Abstände können laut Sensordatenblatt mit einer Abweichung von unter 1% der gemessenen Distanz bestimmt werden.

Als Rohdaten liefert der RPLIDAR neben der Abstandsmessung R und dem Winkel θ für jeden einzelnen Punkt außerdem einen Qualitätswert für jede Messung. Dieser gibt an, mit welcher Intensität das Lasersignal reflektiert wurde. Jeder der 360 Messpunkte kann mittels dem jeweiligen Abstand und Winkel im 2D Raum positioniert werden, wodurch die zu verarbeitende 2D Punktewolke entsteht. [32]

1.2.1 2D Punktewolke

Eine 2D Punktewolke besteht somit aus einer beliebigen Anzahl vom Messpunkten im zwei Dimensionalen Raum. Die Position der jeweiligen Punkte kann über Polarkoordinaten mit dem Abstand R und dem Winkel θ oder mittels kartesischen Koordinaten über die jeweiligen x- und y-Koordinaten der Punkte beschrieben werden. Die Punkte innerhalb der durch die Lidar-Messung erzeugten Punktewolke beschreiben somit im Falle eines gemessenen Objektes verschiedene Konturen. In Abbildung 4 ist eine skizzierte Darstellung einer Beispielspunktewolke gezeigt.

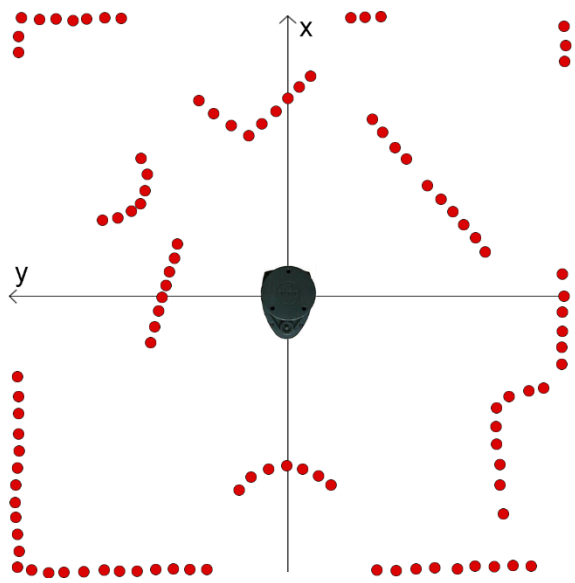

Abbildung 4: Skizzierte Darstellung einer Beispielpunktewolke des RPLIDARS

1.3 Raspberry Pi

Der Kleinstrechner, der in dieser Arbeit verwendet wurde ist der „Raspberry Pi 3 Model B“, welcher mit der Linux-Distribution Ubuntu läuft. Das Betriebssystem des Kleinstrechners wird über eine SD-Karte gebootet und steht im Internet frei zur Verfügung. Die in Kapitel 4 beschriebene Implementierung wurde auf dem Raspberry Pi umgesetzt.

Abbildung 5: Raspberry Pi 3 Model B [33]

Im Folgenden sind die wichtigsten technischen Daten [33] des Raspberry Pi 3 Model B aufgelistet.

GPU:	Videocore IV
Prozessor Geschwindigkeit:	QUAD Core @1250 MHz
Arbeitsspeicher:	1GB SDRAM @ 400 MHz
Speicher:	MicroSD
USB 2.0:	4x USB Ports
Stromstärke / Spannung:	2.5 Ampere / 5 Volt
GPIO:	40 Pins
Ethernet Port:	ja
Wi-Fi:	Verbaut
Bluetooth LE:	Verbaut

1.4 Segmentierung

Segmentierung ist nach der Erstellung der Punktewolke aus den Rohdaten des Laserscanners der erste Schritt der Verarbeitungskette zur Objektdetektion. Ziel der Segmentierung ist es, innerhalb der Punktewolke zusammenhängende homogene Regionen zu finden. Die Daten der Punktewolke werden dafür nach Ähnlichkeitsstrukturen beziehungsweise Punkte mit ähnlichen Charakteristiken untersucht und daraufhin in verschiedene Segmente aufgeteilt. Die vollständige Punktewolke besteht somit nach der Segmentierung aus einer bestimmten Anzahl von Segmenten, welche zur Weiterverarbeitung bezüglich der Objektdetektion verwendet werden kann. Nyguyen und Le [11] haben sich mit verschiedenen Segmentierungsmethoden auseinandergesetzt und diese grundsätzlich in die fünf verschiedenen Kategorien der kantenbasierten, regionsbasierten, attributbasierten, modellbasierten und graphenbasierten Segmentierung aufgeteilt. Diese werden im Folgenden kurz erläutert:

Kantenbasierte Segmentierung
Bei einer kantenbasierten Segmentierung wird davon ausgegangen, dass sich die Intensität entlang einer Linie in der Punktewolke stark ändern, wenn ein gemessenes Objekt durch eine Kante abgegrenzt wird. Die Kanten beschreiben dabei die charakteristische Form des Objekts. Nyguyen und Le [11] beschreiben die größte Schwäche der kantenbasierten Segmentierung, dass bei Rauschen oder ungleicher Punktedichte es schnell zu fehlerhaften Ergebnissen kommt.

Regionsbasierte Segmentierung
Bei der regionsbasierten Segmentierung werden sogenannte *neighborhood points*, welche räumlich nach beieinanderliegen und ähnliche Eigenschaften besitzen zu Segmenten gruppiert. Die *bottom-up*-Methode lässt die Segmente aufgrund bestimmter Bedingungen ab einem gewissen Datenpunkt anwachsen, bis eine andere Bedingung das Segmentwachstum beendet. Bei der *top-down*-Methode werden anfangs alle Daten innerhalb eines Segmentes zusammengefasst und dann aufgrund von bestimmten Bedingungen in mehrere kleinere Segmente geteilt. Laut Nyguyen und Le [11] liefern die regionsbasierten Methoden genauere Ergebnisse beim Auftreten von Rauschen, wobei die bottom-up-Methode diesbezüglich der top-down Methode noch etwas unterlegen ist. Außerdem neigt die regionsbasierte Segmentierung generell zur Über-, beziehungsweise Untersegmentierung.

Attributbasierte Segmentierung
Die attributbasierte Segmentierung ist ein robustes und flexibles Verfahren, welches die Segmente aufgrund von festgelegten Attributen gruppiert. Dabei werden als erstes die Attribute berechnet und daraufhin aufgrund von Bedingungen gruppiert. Das Ergebnis hängt dabei stark von der Auswahl der richtigen Attribute ab. Diese sollten optimal an die zu Aufgabenstellung angepasst werden. Die Attribute können dabei beispielsweise Distanzen oder Punktedichten wiederspiegeln. Das Problem der attributbasierten Segmentierung ist, dass besonders bei einer großen Anzahl von Datenpunkten im multidimensionalen Raum sehr großen Rechenzeiten resultieren. [11]

Modellbasierte Segmentierung

Modelbasierte Methoden erkennen und gruppieren einfache geometrische Formen wie Linien, Flächen oder Zylinder innerhalb der Punktewolke. Die gesamte Segmentierung findet somit nach einem rein mathematischen Prinzip statt. Generell ist die modellbasierte Segmentierung sehr robust gegenüber Ausreißern. Ungenauigkeiten entstehen allerdings, wenn die zu segmentierende Punktewolke aus verschiedenen Messquellen erzeugt wurde. [11]

Graphenbasierte Segmentierung

Bei der graphenbasierten Segmentierung wird die vorhandene Punktewolke in einen Graphen umgewandelt, der danach durch verschiedene mathematische Verfahren auf Ähnlichkeitsstrukturen untersucht werden kann. Die graphbasierten Methoden sind sehr genau und werden aufgrund ihrer Effizienz vermehrt bei Roboteranwendungen und im Bereich des maschinellen Lernens angewendet. Im Vergleich mit den anderen Methoden wird besonders bei komplexen Punktewolken im höherdimensionalen Raum, sowie bei Rauschen und ungleichmäßiger Dichte ein gutes Ergebnis erzielt. Dem entgegen sind graphenbasierte Methoden meist nicht Echtzeit fähig und erfordern oft speziell mitintegrierte Sensoren oder Kamerasysteme. [11]

1.4.1 Euklidische Abstände

Viele der in dieser Arbeit beschriebenen Segmentierungsmethoden beruhen auf dem Ansatz der Euklidischen Abstände. Aus diesem Grund wird dieser vor der spezifischen Erläuterung der einzelnen Segmentierungsmethoden hier kurz erklärt. Der Euklidische Abstand d beschreib die direkte Distanz zwischen zwei Messpunkten.

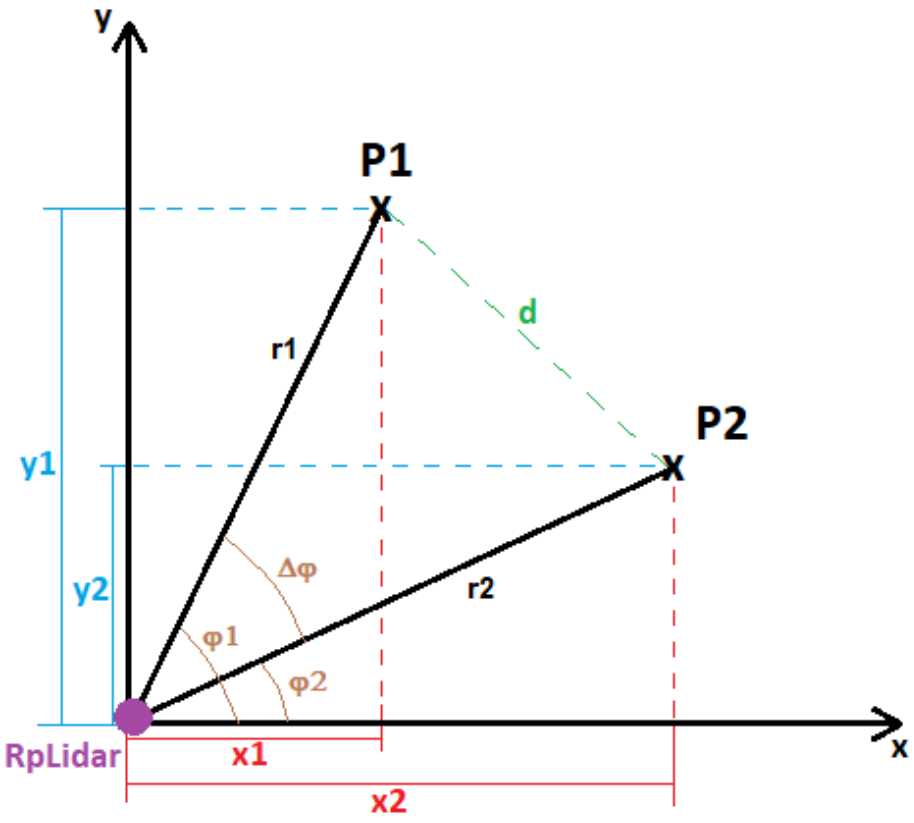

Abbildung 6: Euklidische Abstandsberechnung von zwei Messpunkten

Sind die verschiedenen Messpunkte als kartesischen Koordinaten $P_1\ (x_1, y_1)$ und $P_2\ (x_2, y_2)$ dargestellt, ergibt sich der Euklidische Abstand d nach dem Satz des Pythagoras über:

$$d_{P_1,P_2} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}. \tag{1.1}$$

Parallel dazu lässt sich der Abstand d bei Punkten in Polarkoordinaten mit der Eigenschaft $P\ (r, \varphi)$ für die beiden Punkte $P_1\ (r_1, \varphi_1)$ und $P_2\ (r_2, \varphi_2)$ wie folgt bestimmen:

$$d_{P_1,P_2} = \sqrt{{r_1}^2 + {r_2}^2 - 2r_1 r_2 \cos(\varphi_1 - \varphi_2)}. \tag{1.2}$$

Die Winkeldifferenz $\Delta\varphi = \varphi_1 - \varphi_2$ kann im Falle der 2D Lidarabstandsmessung bei zwei benachbarten Messpunkten mit den Winkelschritten zwischen den einzelnen Messungen gleichgesetzt werden. Somit ist $\Delta\varphi = \frac{360°}{n}$, wobei n der Anzahl der in einer Messung gemessenen Messpunkte entspricht. [1]

Im Folgenden werden verschiedene Segmentierungsmethoden erläutert.

1.4.2 Clusteranalyse

Die Clusteranalyse bezieht sich auf die Untersuchung von Ähnlichkeitsstrukturen innerhalb der aufgezeichneten Datensätze. Dadurch können die Daten in verschiedene Cluster, bzw. natürlichen Gruppen zusammengefasst werden. Aus diesem Zweck sind bereits unzählige Clusteralgorithmen entwickelt worden, welche sich durch unterschiedliche Vorgehensweisen und Strategien der Datenanalyse unterscheiden.

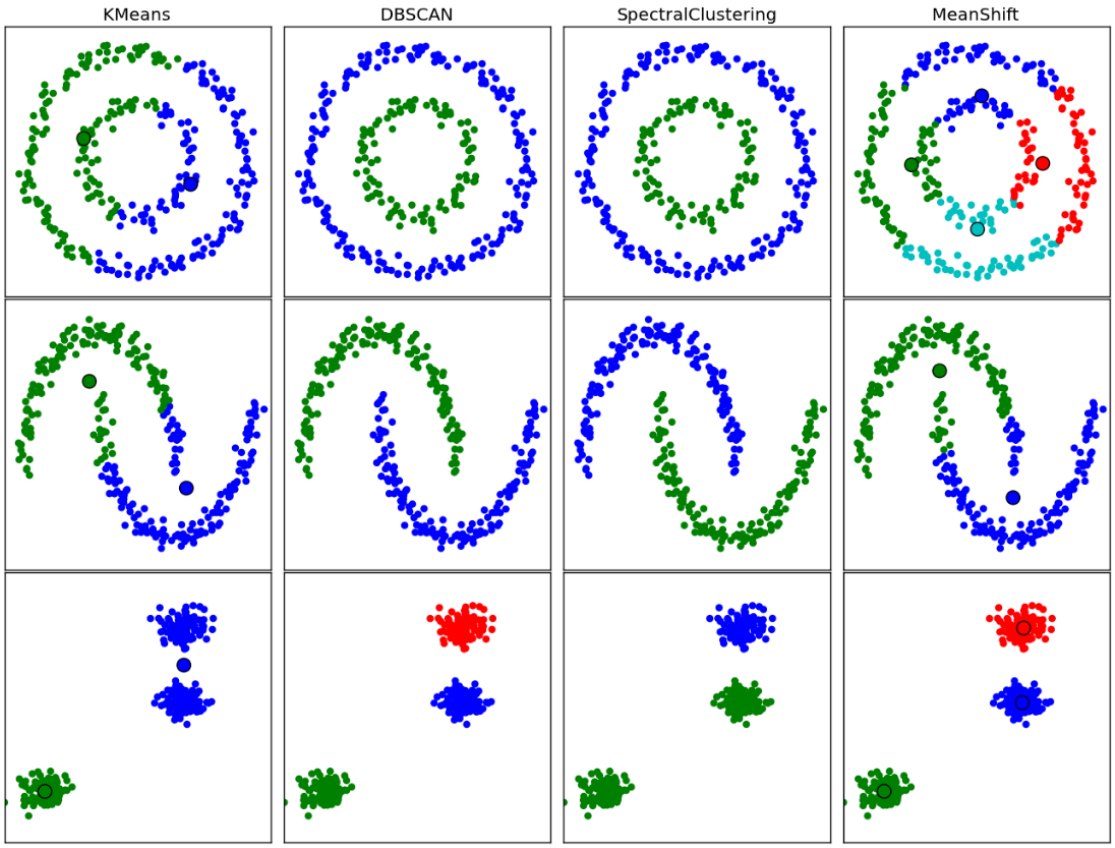

Abbildung 7: Charakteristik verschiedener Clusteralgorithmen bei Analyse von identischen Datensätzen. [2]

Verschiedene Clusteringverfahren unterscheiden sich nicht nur in ihrer Arbeitsweise, sondern auch dadurch, welche Arten von Clustern erzeugt werden. Diese werden nach [5] aufgeteilt in:

- **Iterativ:** Der Algorithmus beginnt mit einer Anfangsmenge von Clustern und verbessert daraufhin die anfängliche Einteilung.
- **Hierarchisch oder flach:** Ein hierarchischer Clusteralgorithmus generiert eine Hierarchie von Clustern, bei der die Cluster in verschiedene Ebenen eingeteilt werden. Bei flachen Algorithmen sind alle Cluster gleichwertig.
- **Disjunktiv:** Eine Instanz bzw. ein Datenpunkt kann mehreren Clustern gleichzeitig zugeordnet werden.
- **Hart oder weich:** Hartes Clustering beschreibt, dass eine Instanz auch nur genau einem Cluster zugeordnet wird. Bei weichem Clustering werden diese mit einer Wahrscheinlichkeit einem bestimmten Cluster zugeordnet.

Im Folgenden sind die in Abbildung 7 dargestellten Clusteralgorithmen kurz in ihren grundlegenden Eigenschaften und Funktionen erklärt.

K-Means

Der K-Means Algorithmus ist einer der am häufigsten angewandten und bekanntesten Verfahren zur Clusteranalyse. Er findet Anwendung auf Daten im n-dimensionalen Raum. Zur Clusterbildung muss bei K-Means Anfangs die Anzahl k der zu unterscheidenden Gruppen festgelegt werden. Entsprechend der Anzahl werden zufällig gewählte Mittelwert-Zentren im Raum verteilt. Jeder Datenpunkt sowie die einzelnen Zentren können als Vektor $\vec{P}_i = (x_i, y_i, \dots, n_i)$ mit einem Wert pro Attribut/Dimension beschrieben werden. Nun werden jegliche Datenpunkte dem Zentrum zugewiesen, welche zu dem jeweiligen Datenpunkt den geringsten Euklidischen Abstand (siehe Abschnitt 1.4.1) aufweist. Aus den dadurch entstehenden Clustern, kann über den Mittelwert mit der Anzahl der im Cluster enthaltenen Datenpunkten n_P, ein neues Cluster-Zentrum $\overrightarrow{ZP_{neu}}$ berechnet werden.

$$\overrightarrow{ZP_{neu}} = \frac{1}{n_P} \sum_{\vec{P}_i \in n_p} \vec{P}_i \tag{1.3}$$

Durch die neu entstandenen Zentren können dann aufgrund der Abstandswerte auch wieder neue Clustergruppen gebildet werden. Dieser Vorgang wird solange wiederholt, bis die Zentren sich nicht mehr verschieben und eine konstante Position eingenommen haben.

Die größte Problematik von K-Means ist, dass die Anzahl der zu bildenden Datengruppen schon vorher bekannt sein muss. Auch die zufällige Wahl der Mittelwert-Zentren kann eine sehr große Anzahl von Berechnungsschleifen zu Folge haben. „K-Means ++“ ist eine Erweiterung des ursprünglichen Algorithmus, wodurch die Clusterzentren nicht mehr zufällig, sondern nach einem bestimmten Verfahren gewählt werden, um somit die Anzahl der Schleifen zu minimieren. [3]

DBSCAN

DBSCAN ist ein dichtebasiertes Verfahren. Der Algorithmus erkennt Bereiche mit einer hohen Dichte von Datenpunkten. Gleichzeitig wird angenommen, dass die Dichte innerhalb eines Clusters höher ist als im Randbereich desselben Clusters. Um ein dichtebasiertes Clustering durchzufuhren, müssen die beiden Parameter $MinPts$ und ε festgelegt werden. Dabei ist $MinPts$ die minimale Anzahl von Datenpunkten, die sich in einem Gewissen ε – Umfeld befinden müssen, damit ein Cluster gebildet werden kann. Das heißt, dass der Algorithmus für jeden Datenpunkt P, welcher in diesem Fall als Kernpunkt beschrieben wird, in einem Umkreis von einem Radius ε prüft, ob sich die Mindestanzahl $MinPts$ von weiteren Datenpunkten im ε – Umfeld befindet. Ist dies der Fall, wird der Kernpunkt einem Cluster zugeordnet.

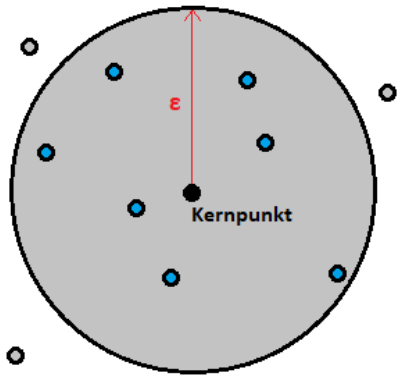

Abbildung 8: Dichtebasiertes DBSCAN Clustering

Ein Vorteil des DBSCAN Verfahrens ist, dass durch die Parameterbestimmung das Clustering auf einen bestimmten Fall angepasst werden kann. Ebenfalls filtert der Algorithmus Rauschen meist sehr gut und erkennt Cluster von beliebiger Form und Größe. Problematisch wird es für den Algorithmus bei Clustererkennung von unterschiedlichen Dichten. Wird der Wert für ε somit sehr groß angenommen, um auch Cluster mit einer niedrigeren Dichte zu erkennen, besteht die Gefahr, dass Rauschgebiete als Cluster erkannt werden. Um die Parameter sinnvoll zu bestimmen, wird auf die Methode des „k – Distanz-Graphen“ zugegriffen. Diese erlaubt die Bestimmung einer „Grenzdichte“, welche auf das Cluster mit der niedrigsten Dichte ausgelegt wird, um dieses zu erkennen, Rauschen aber dennoch effektiv zu entfernen. [4]

Spektrales Clustering

Der Spektrale Clustering Algorithmus verwendet ein auf Graphen basiertes Verfahren. Als erstes muss also aus den vorhandenen Daten ein Graph erzeugt werden. Dazu werden die einzelnen Datenpunkte als Knoten des Graphen und die Euklidischen Abstände zwischen den Datenpunkten als Kanten betrachtet.

Nach der Bildung des Graphen wird dieser in verschiedenen Schritten auf Ähnlichkeitsstrukturen untersucht und reduziert. Dafür werden zum einen Kanten mit zu großem Gewicht, auf im Fall der Lidardatensegmentierung bezogen, mit zu großen Abständen entfernt. Ähnlich wie bei dem DBSCAN wird dafür ein Grenzwert ε festgelegt. Danach wird der *k-nn Graph* gebildet, bei dem alle Kanten zu einem Knoten vorerst sortiert und dann je nach Kantengewicht ebenfalls vom Graphen entfernt werden. Dafür sollte k so gewählt werden, dass der

Graph nicht aus weniger Zusammenhangskomponenten besteht, als Cluster erwartet werden.

Über die Eigenvektoren der Laplace-Matrizen können die Datenpunkte in einem niedrigerdimensionalen Raum eingebettet werden, wodurch die Aufteilung der einzelnen Cluster ersichtlicher wird.

Dafür wird zuerst die Adjazenzmatrix $A_{n\times n}$ aus dem Graphen gebildet, wobei n die Anzahl der noch bestehenden Knoten im Graphen entspricht. $A_{n\times n}$ berechnet sich wie folgt:

$$A_{ij} = \begin{cases} \omega_{ij} & : \textit{Gewicht zwischen den Knoten (i,j)} \\ 0 & : \textit{Falls keine Kante zwischen den Knoten (i,j)} \end{cases} \tag{1.4}$$

Die Laplace Matrix L setzt sich dann wie folgt zusammen:

$$L_{ij} = \begin{cases} d_i & : \textit{Falls } i = j \\ -w_{ij} & : \textit{Gewicht zwischen den Knoten (i,j)} \\ 0 & : \textit{Falls keine Kante zwischen den Knoten (i,j)} \end{cases} \tag{1.5}$$

Wobei d_i der Summe einer Zeile aus $A_{n\times n}$ entspricht:

$$d_i = \sum_{\{j|(i,j)\in E\}} w_{ij} \tag{1.6}$$

Anschließend können über $L * v = \lambda * v$ von den k größten Eigenwerten die Eigenvektoren $\{v_1, v_2, \dots, v_k\}$ der Laplace Matrix bestimmt werden. Aus den Eigenvektoren kann anschließend eine Matrix erstellt werden, welche die einzelnen Cluster ersichtlicher darstellt. Durch die darauffolgende Anwendung eines weiteren Clusteralgorithmus wie zum Beispiel K-Means, können sehr gute Ergebnisse erzielt werden.

Ein großer Vorteil des Spektralen Clusterings ist, dass es insbesondere im höherdimensionalen Bereich auch komplexe Formen, wie zum Beispiel einer verdrehten Spirale, als ein Cluster erkennen kann. Ähnlich wie bei K-Means ist aber von großem Nachtteil, dass die Anzahl k der Cluster schon vorher festgelegt werden muss. [6], [7]

MeanShift

Das MeanShift-Clustering beschreibt ein Segmentierungsverfahren, welches Dichtezentren, sogenannte Modi, in multidimensionalen Merkmalsräumen ermittelt. Jeder Datenpunkt wird dafür Anfangs sich selbst als temporärer-Modus zugewiesen. Dieser befindet sich innerhalb eines n – dimensionalen Ellipsoiden, welches als Kernel bezeichnet wird. Über die innerhalb des Kernels liegenden Datenpunkte, kann ein neuer kernellokaler-Modus berechnet werden, um welchen erneut ein Kernel gelegt wird. Es wird davon ausgegangen, dass wenn die da-

rauffolgenden Modi für jeden Datenpunkt iterativ verschoben werden, sich alle Modi innerhalb eines Clusters an einem bestimmten Punkt sammeln, welcher dann als globaler Modus bezeichnet wird.

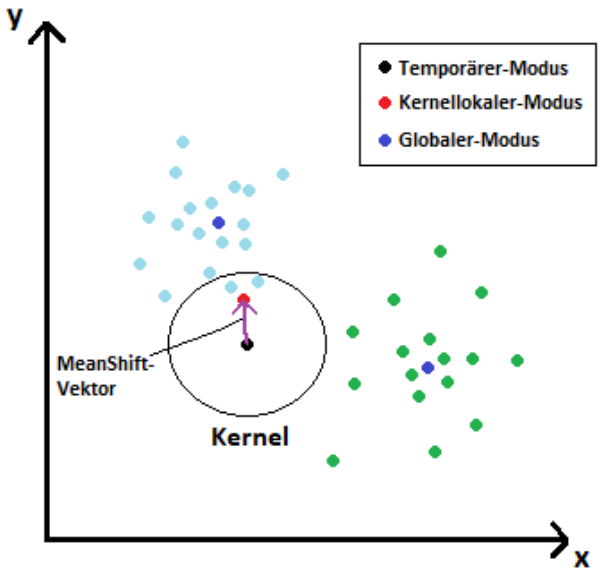

Abbildung 9: MeanShift Berechnung des Dichtezentrums (globaler Modus)

Innerhalb eines Kernels wird der kernellokale Modus als Durchschnitt aller im Kernel befindlichen Punkte ermittelt. Zwischen den temporären-Modus und den kernellokalen-Modus wird der MeanShift Vektor gelegt. Entlang dieses Vektors wird der temporäre-Modus, und somit auch das Kernel verschoben. Dieser Vorgang wird solange wiederholt, bis sich der temporäre-Modus nicht mehr von dem kernellokalen-Modus unterscheidet, welcher dann als globaler Modus bezeichnet wird.

Vorteilhaft von MeanShift ist, dass keine Angabe zu den zu bestimmenden Clustern benötigt wird. Ebenfalls arbeitet der Algorithmus Formunabhängig. Dafür sollte die Kernelgröße optimal für jeden Segmentierungsfall angepasst sein, da es ansonsten zu Fehlerhaften Ergebnissen kommen kann. [8], [9]

1.4.3 Occupancy Grid + CC-Algorithm

Eine weitere Möglichkeit der Datensegmentierung ist die Verwendung einer Gitterkarte, welche Fachspezifisch als Occupancy Grid bezeichnet wird. Dafür werden wie in [10] beschrieben die Sensormesspunkte als erstes von Polar- in Kartesische Koordinaten umgewandelt und daraufhin diskretisiert. Anschließend können die 2D Punkte auf den Occupancy Grid projiziert werden.

Die Größe des zu erzeugenden Gitters mit m^2 Kästchen entsteht wiederum durch:

$$m = \frac{range\_max}{s} \tag{2.3}$$

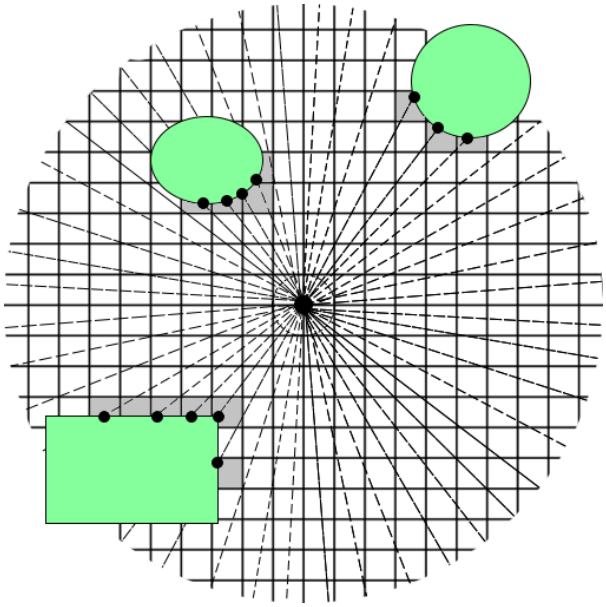

Abbildung 10: Objekterfassung mittels "Occupancy Grid"

Wobei $range\_max$ die maximale Messreichweite des Sensors, bzw. die maximale erwünschte Segmentierungsreiche beschreibt. Die Größe der einzelnen Gitter wird durch den Faktor s bestimmt. Das gesamte Gitter besteht somit aus $m * n$ vielen Kästchen, welche entweder mit 0 oder k Messpunkten gefüllt sind. Jedes Kästchen kann dabei wie bei einem Bild als ein einzelnes Pixel angesehen werden, welches entweder den Zustand 1 oder 0 besitzt. Befinden sich keine Punkte innerhalb eines Kästchen, ist der Zustand 0 und das „Pixel“ ist leer. Reziprok dazu besitz ein Kästchen mit Messpunkt den Zustand 1. Diese sind in Abbildung 10 als graue Kästchen dargestellt. Über die benachbarten Kästchen kann dann festgestellt werden, ob sich ein Objekt im Messbereich befindet, oder nicht. Der Faktor s ist somit ausschlaggebend für die Qualität der Segmentierung. In [10] wird beschrieben, dass durch eine funktionale Anpassung des Faktors das Segmentierungsergebnis verbessert werden kann. Ein möglicher Ansatz dafür wäre eine mit dem Abstand zum Sensor wachsende Gittergröße, welche an die Winkel zwischen den einzelnen Messungen angepasst ist.

Die Berechnung, welche Kästchen des Gitters mit einer 1 belegt werden, wird wie folgt durchgeführt:

$$x = \frac{x_i}{s} + l_x \ , \tag{1.7}$$

$$y = \frac{y_i}{s} + l_y \ . \tag{1.8}$$

Wobei x_i und y_i jeweils die x- und y- Koordinaten des jeweiligen Messpunktes und l_x und l_y die Position des Lidarsensors auf der Grid-Map beschreiben. [10]

Connected Components Algorithmus (CC-Algorithm)

Zur Weiterverarbeitung und schließlich zur Segmentierung wird in [10] der Connected Components Algorithmus, kurz CC-Algorithmus verwendet. *Connected Component Labeling* wird in der Bildanalyse als ein grundlegender Verarbeitungsschritt verwendet. Verschiedene Bild-

bereiche werden demnach segmentiert, indem der Algorithmus zusammengehörige benachbarte Bildpunkte zu einem Bildbereich zusammenfasst. Den einzelnen Segmenten wird dann jeweils eine eindeutige Identifikationsnummer zugewiesen. Dabei wird das Bild von der Binärdarstellung in ein gelabeltes Bild umgewandelt, wie in Abbildung 6 dargestellt.

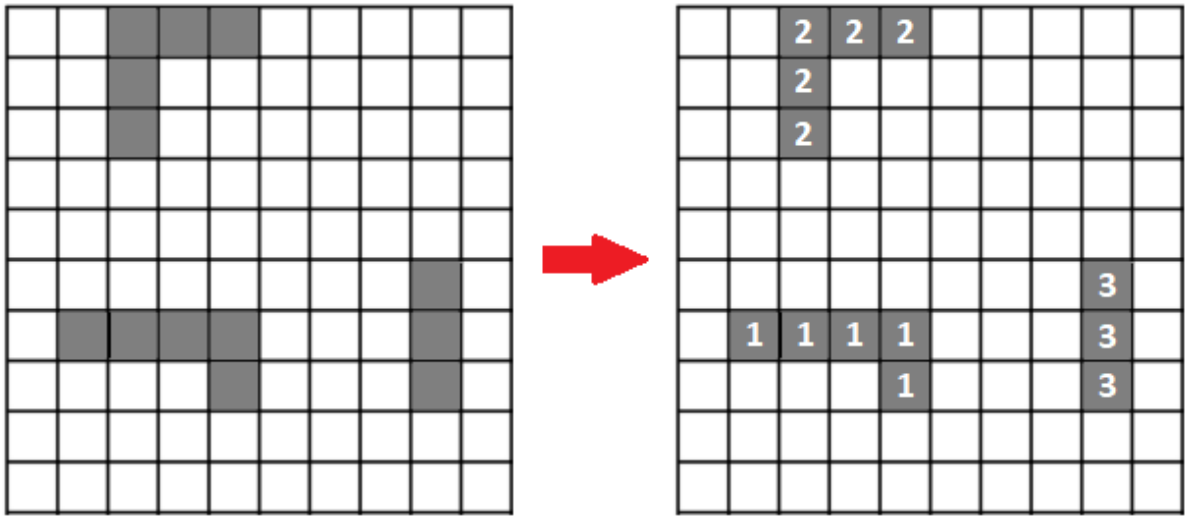

Abbildung 11: Umwandlung von Binärbild (links) in gelabeltes Bild (rechts)

Die jeweiligen Bildpunkte werden von dem Algorithmus genau dann zu einem Segment zusammengefasst, wenn die zu segmentierenden Bildpunkte unmittelbar nebeneinanderliegen. Ist dies der Fall, wird jedem Bildpunkt eines jeweiligen Segments eine Nummer zugewiesen, sodass alle Bildpunkte aus demselben Segment die gleiche Nummer besitzen. Parallel dazu wird Bildpunkten aus verschiedenen Segmenten eine andere Nummer vergeben.

Für die benachbarten Bildpunkte wird außerdem zwischen dem 4- und 8-Zusammenhang unterschieden. Bei dem 4-Zusammenhang werden ausschließlich die Bildpunkte zusammengefasst, welche unmittelbar an den Pixelgrenzen zueinander anliegen. Das heißt von einem Bildpunkt aus gesehen, dass der Weg nur über die unteren und oberen, sowie die rechts und links liegenden Nachbarn verlaufen darf. Der 8-Zusammenhang schließt ebenfalls die vier Bildpunkte, welche Diagonal zu diesem liegen mit ein.

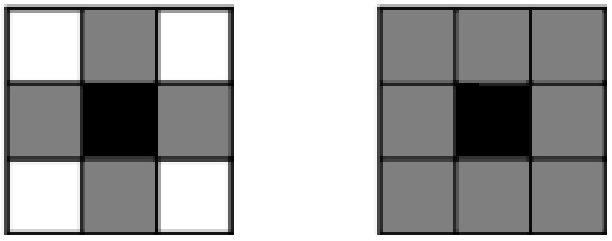

Abbildung 12: CC-Algorithmus Segmentierung ausgehend von Zentralbildpunkt (schwarz) im Falle 4-Zusammenhang (links) und 8-Zusammenhang (rechts)

Die Wahl zwischen 4- , beziehungsweise 8-Zusammenhang kann im Ergebnis einen entscheidenden Unterschied machen. Ein gutes Beispiel dafür ist in Abbildung 10 zu erkennen, bei dem das Objekt oben rechts mit dem 4- Zusammenhang in zwei Segmente geteilt werden

würde. Außerdem würde das Objekt in dem Fall, dass einzelne Bildpunkte als Rauschen entfernt werden, als ein kleineres Objekt wahrgenommen werden würden.

1.4.4 RBNN-Segmentation

Der *radially bounded nearest neighbor* Algorithmus (RBNN-Algorithmus), welcher ursprünglich von *Klasing* [12] entwickelt wurde, bietet eine weitere Alternative zur 2D Punktwolken Segmentierung. Die RBNN-Segmentation benötigt laut *Klasing* vergleichsweise wenig Parameter und Rechenzeit. Um einen Datenpunkt wird in einem bestimmten Abstand r nach benachbarten Messpunkten gesucht. Befindet sich innerhalb des vom Radius aufgespannten Kreises ein weiterer Messpunkt, wird dieser demselben Segment zugeordnet. Dadurch muss nicht jeder Messpunkt geprüft werden, sondern nur diese, welche noch keinem Segment zugeordnet wurden. Dadurch kann die Rechenzeit reduziert werden. Fall sich keine weiteren Punkte in dem Suchradius befinden, oder das Segment aus weniger als einer vorher bestimmten Mindestanzahl von Punkten besteht, können die Punkte verworfen werden.

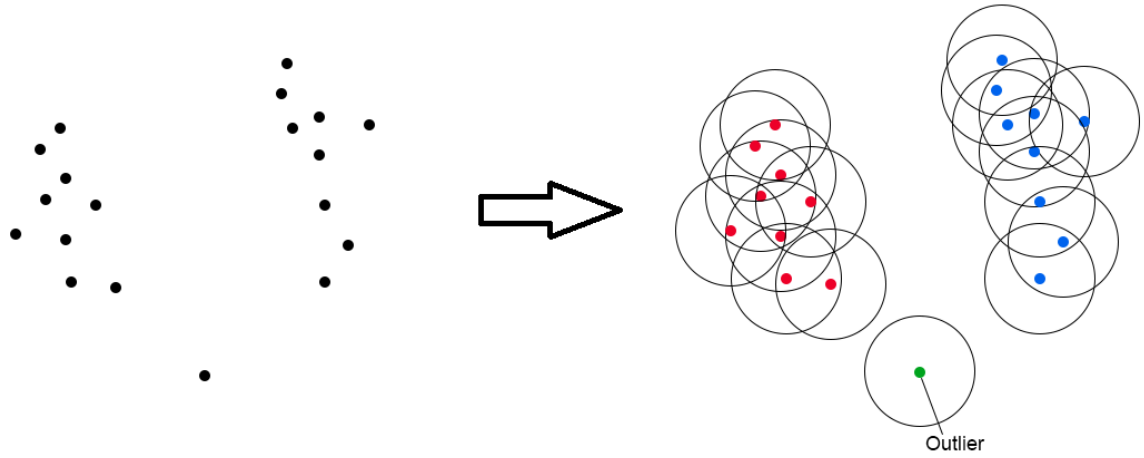

Abbildung 13: RBNN-Segmentierung von zwei Segmenten und einem Ausreißer

Einen interessanten Ansatz der Weiterentwicklung des RBNN-Algorithmus wird in [13] von Yangeun Choe, Seunguk Ahn und Myung Jin Chung beschrieben. Diese erläutern das Problem, dass der Suchradius r für alle Messpunkte gleich ist und somit der Algorithmus zur Über,- bzw. Untersegmentierung neigt. Demnach tendiert die RBNN-Segmentation mit zu großem Suchradius zur Untersegmentierung bei Objekten in kurzer Distanz zum Sensor, sowie gegensätzlich dazu bei einem zu kleinen Radius bei Objekten mit großer Distanz zum Sensor zur Übersegmentierung. Die Lösung dafür ist die Erweiterung des Algorithmus durch die Funktion, den Suchradius an den Abstand zwischen Sensor und Messpunkt anzupassen.

[12], [13]

1.4.5 Obstacle Detection and Obstacle Avoidance Algorithm

Der *Obstacle Detection and Obstacle Avoidance Algorithm* [14], vorgestellt von Yan Peng, Dong Qu, Yuxuan Zhong, Shaorong Xie und Jun Luo ist ein speziell auf 2-D Lidar Sensoren angepasster Algorithmus zur Segmentierung, Lokalisierung, Klassifizierung sowie der Vermeidung von Kollisionen beim Einsatz im Fahrzeug. Der Algorithmus wird im weiteren Verlauf der Arbeit als *ODAOA-Algorithmus* abgekürzt.

Dabei werden als erster Segmentierungsschritt alle erfassten Messpunkte i in Segmente aufgeteilt. Für die Aufteilung in die jeweiligen Segmente wird dafür eine Start-Bedingung, sowie eine Stop-Bedingung bestimmt. Diese legen fest, an welchen Messpunkt p ein Segment beginnt und bei welchem „Stopp-Messpunkt“ q das Segment abgegrenzt wird. Dabei gelten für den Abstand $R(i)$ zwischen Lidarsensor und Objekt folgende Bedingungen:

Start-Bedingung

$$(R(i-1) == 0)\ \&\&\ (R(i)\ != 0) \tag{1.9}$$

Stopp-Bedingung

$$(R(i)\ != 0)\ \&\&\ (R(i+1) == 0) \tag{1.10}$$

Das heißt, alle Messungen, bei denen der vom Lidarsensor ausgesendete Laserstrahl nicht von einem Objekt reflektiert wird, und somit nicht wieder vom Receiver des Sensors eingefangen wird, werden nicht beachtet. Ab dem Zeitpunkt, an dem ein Laserstrahl zum Sensor zurückkehrt, ist die gemessene Distanz von $R(i)\ != 0$ und der Messpunkt i wird als Startpunkt p eines Segments festgelegt. Im Gegensatz dazu, wird der Endpunkt q bei dem Punkt i festgelegt, bei dem der Receiver nichts mehr empfängt.

Daraus resultiert, dass die Punktewolke in N „Blöcke“ [14] eingeteilt ist. Innerhalb der Blöcke wird jeweils der Euklidische Abstand d zwischen den Punkten $P(i)$ und $P(i+1)$ geprüft. Wenn dieser innerhalb eines Blocks einen gewissen Wert T (threshold) übersteigt, so wird der einzelne Block in zwei Segmente unterteilt. Die Anzahl der Blöcke N wird dadurch um eins erweitert. Im Falle des in [14] beschriebenen Algorithmus wird T durch die Multiplikation der Roboterbreite *Width* mit einem Vergrößerungsfaktor *k* bestimmt. Dadurch wird sichergestellt, dass die Blöcke nur dann in verschiedene Segmente geteilt werden, wenn der Roboter auch zwischen diesen hindurchfahren kann.

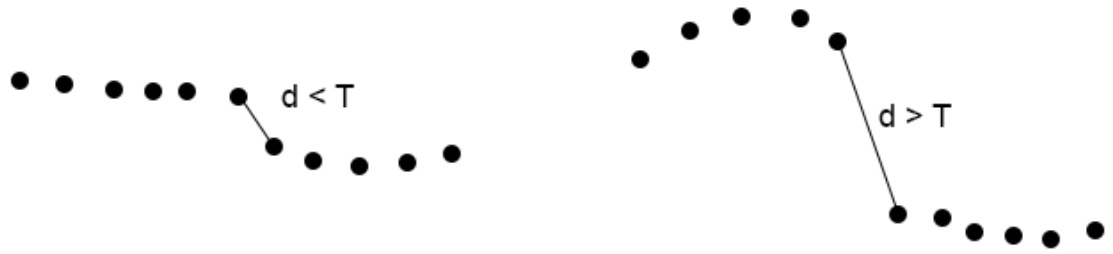

Abbildung 14: Teilen oder verschmelzen der Blöcke aufgrund von Grenzwert T

Ebenfalls prüft der Algorithmus die Euklidischen Distanzen d zwischen den jeweiligen Endpunkten $q(i)$ und dem Startpunkt des nächsten Blocks $p(i+1)$, wobei in diesem Fall i die einzelnen Blöcke beschreibt. Wenn d wiederum kleiner ist als ein gewisser Grenzwert T, werden die zwei Blöcke i und $i+1$ verbunden. In [14] wird ebenfalls wieder die Roboterbreite *Width* für *d* als Grenzwert verwendet. Dafür werden zwischen den Punkten $p(i)$ und $q(i+1)$ weitere Datenpunkte eingefügt, welche besonders in der späteren Klassifizierung wichtig sind. Vorausgesetzt, dass die Anzahl num an Datenpunkten zwischen $p(i)$ und $q(i+1)$ ein-

gefügt werden sollen, und die Abstände der jeweiligen Grenzpunkte mit $R(p)$ und $R(q)$ beschrieben werden, so gilt für die Abstände der neuen Datenpunkte mit $i \in [1, num]$ nach [14] folgende Gleichung:

$$R(i) = R(p) + i * \frac{R(q) - R(p)}{num} \tag{1.11}$$

Nach dem Zusammenfügen der beiden Blöcke wird somit die Gesamtzahl N um eins reduziert. [14], [17]

1.5 Klassifizierung

Nachdem die Segmentierung auf die Punktewolke angewendet und somit aus den Daten verschiedene Segmente gebildet wurden, folgt auch in der Literatur meist der nächste Schritt der Klassifizierung. Die Klassifizierung weißt, wie Grilli et al. in [15] beschreibt, den einzelnen Punkten beziehungsweise Segmenten spezifischen Klassen zu. Die Segmente können zum Beispiel in gewissen Formen unterteilt und somit klassifiziert werden. Dafür wird meist zuerst die Segmentierung als Grundlage der Klassifizierung durchgeführt. Bestimmte Segmentierungsverfahren wie zum Beispiel die modellbasierte Segmentierung, schließen die Klassifizierung schon automatisch mit ein, da diese die Punktewolke von Anfang an auf bestimmte Formen überprüft.

Im Folgenden werden verschiedene Klassifizierungsverfahren erläutert.

1.5.1 Hough Transformation

Die Hough Transformation ist ein Bildverarbeitungsalgorithmus zur Erkennung von Geraden und Kreisen, aber auch anderen geometrischen Formen. Dafür werden die Datenpunkte aus dem Ortsraum in den Parameterraum, bzw. im Falle der Hough-Transformation in den sogenannten Hough-Raum transformiert. Im Hough-Raum können Geraden des Ortsraums als Punkte mit den Parametern der Hessischen Normalform [34], also dem Winkel θ zwischen Gerade und x-Achse und dem Euklidischen Abstand d zwischen Ursprung und Lotfußpunkt dargestellt werden:

$$d = x * \cos(\theta) + y * \sin(\theta) \tag{1.12}$$

Jede mögliche Gerade, welchen durch einen Punkt im Ortsraum geht, kann somit durch einen Punkt im Hough-Raum beschrieben werden. Dadurch entsteht aus jenen Punkten eine Kurve. Die Idee der Hough-Transformation ist, dass sich Kurven, die aus mehreren Punkten im Ortsraum, welche zusammen auf einer Gerade liegen und in den Hough-Raum transformiert werden, an einem gewissen Punkt schneiden. Dieser Punkt steht dann für die Gerade, auf welcher sich alle Punkte des Ortsraums befinden.

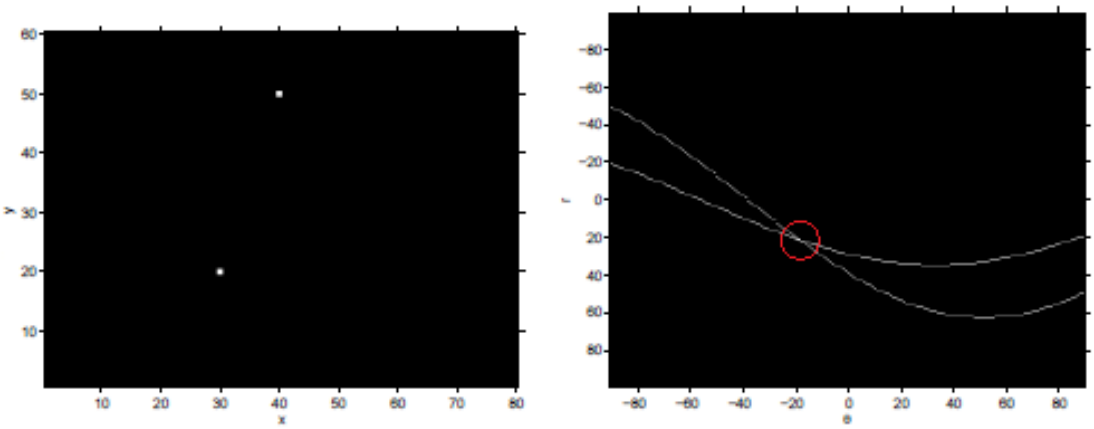

Abbildung 15: Hough-Transformation zur Geradenfindung [16]

Bei der Hough-Transformation zur Kreisbestimmung wird um jeden Datenpunkt mit der Position x und y ein Kreis mit dem Radius R aufgespannt. Liegen die Datenpunkte wie in Abbildung 16 auf einem Kreis, treffen sich die aufgespannten Kreise ähnlich wie bei der Geradentransformation in einem gewissen Punkt. Dieser Schnittpunkt ist dann der Mittelpunkt des gesuchten Kreises mit dem Radius R.

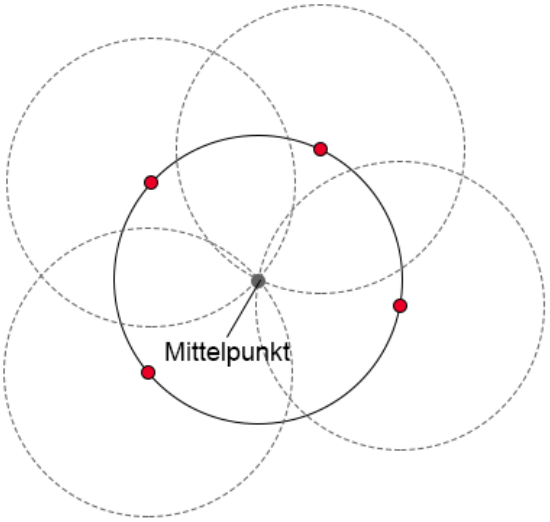

Abbildung 16: Kreisbestimmung durch Hough-Transformation

Dadurch, dass bei der Hough-Kreis-Transformation drei Parameter entscheidend sind, handelt es sich auch um einen dreidimensionalen Hough-Raum. Dadurch resultieren längere Rechenzeiten, welche sich bei komplexeren Formen besonders bemerkbar machen. Ein weiteres Problem der Hough-Transformation ist, dass die durch die Hough-Transformation entdeckten Geraden keinen Start und Endpunkt besitzen. Diese müssen durch eine Nachbearbeitung eingefügt werden. [16]

1.5.2 Graham Scan

Graham Scan ist ein Algorithmus zur Erstellung der Konvex-Hülle einer Punktewolke. In [17] wird dieser dafür verwendet, um anschließend an die Segmentierung, die in verschiedene Blöcke eingeteilte Punktewolke zu untersuchen. Der Algorithmus ergänzt die Segmentierung durch die jeweiligen Formstrukturen der einzelnen Objekte.

Die Bestimmung der Konvexen Hülle durch den Graham Scan Algorithmus erfolgt dabei über die Durchführung der folgenden Schritte:

1. Bestimmung des Startpunktes P_0:

Als Startpunkt wird im Allgemeinen der Punkt festgelegt, welcher sich am nächsten zu der X-Achse befindet, das heißt der Punkt mit der kleinsten Ordinate $Ymin$. Ist dies für mehrere Punkte gleichzeitig der Fall, so wird der Punkt mit der ebenfalls kleinsten Abszisse $Xmin$ gewählt.

2. Einteilen der Punktereihenfolge $P_0, P_1, \dots, P_i$:

Die Reihenfolge der restlichen Punkte wird nach aufsteigendem Winkel zwischen der Horizontalen durch P_0 und der Geraden $\overline{P_0 P_i}$ festgelegt.

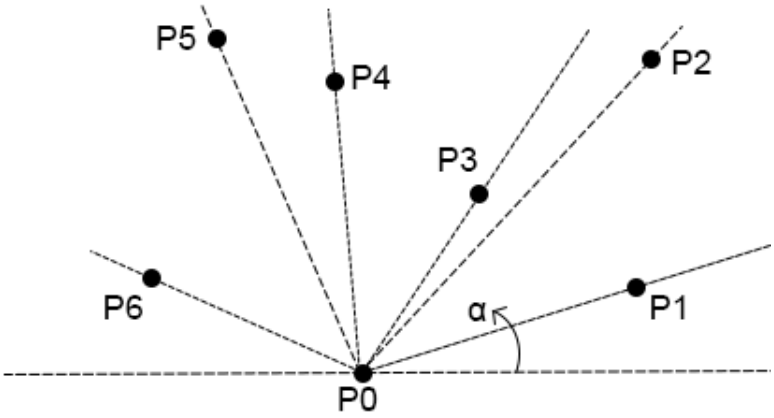

Abbildung 17: Punkteeinteilung Graham Scan

3. Die aufeinander folgenden Punkte werden dann in der vorher festgelegten Reihenfolge miteinander verbunden. Dabei gilt die Regel, dass solange die Winkeländerung zwischen zwei Geraden $\overline{P_i P_j}$ und $\overline{P_j P_k}$ gegen den Uhrzeigersinn größer wird, P_j als Teil der konvexen Hülle angesehen wird. Wird der Winkel mit dem Uhrzeigersinn kleiner, so ist P_j nicht Teil der Hülle. Die einzelnen Punkte werden nach diesem Schema solange miteinander verbunden, bis die Konvexe Hülle durch die Gerade $\overline{P_i P_0}$ geschlossen wird.

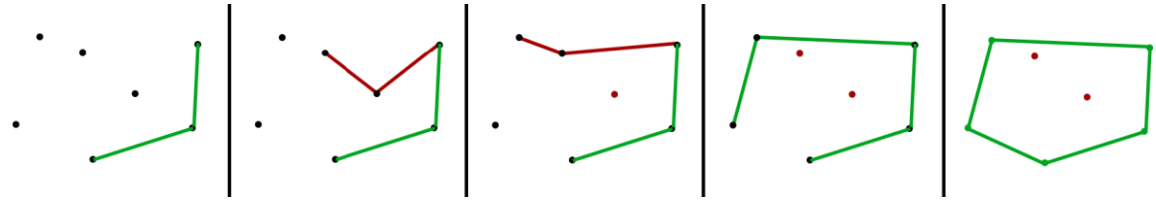

Abbildung 18: Schrittweise Hüllenbildung Graham Scan

Graham Scan ist nicht der einzige Algorithmus zum Erstellen einer Konvex-Hülle. Weitere Algorithmen sind *Gift Wrapping*, *Quickhull* und *Chan's Algorithm*. Diese unterscheiden sich in der Vorgehensweise zur Erstellung der Konvex-Hülle und somit auch in den Rechenzeiten. Welcher Algorithmus dabei das schnellste Ergebnis liefert, muss auf den jeweiligen Anwendungsfall angepasst werden.

Grundlegend ist der Algorithmus eine effiziente Methode zur Kollisionsvermeidung von Objekten im autonomen Fahrbetrieb, da die durch die Punktewolke beschriebene Form der Objekte möglichst genau dargestellt wird. Allerdings bietet der Algorithmus ohne Weiterverarbeitung keine Funktion der Objektklassifizierung in Hinsicht auf bestimmte Formen wie Kreise oder Rechtecke. [17], [35]

1.5.3 Obstacle Detection and Obstacle Avoidance Algorithm

Nach der in [14] beschriebenen Segmentierungsmethode, folgt im gleichen Artikel auch ein Ansatz zur 2D Lidar Klassifizierung. Nach der Segmentierung sind die gesamten Laserdaten, wie in Abschnitt *2.2.6* beschrieben in einzelne Blöcke unterteilt, welche die Segmente bzw. Objekte darstellen. Jeder Block *i* besitzt außerdem einen Startpunkt p und einen Stopppunkt q, welche sich auf den jeweils ersten und letzten Datenpunkt jedes Blocks beziehen. Der *ODAOA-Algorithmus* klassifiziert diese Blöcke in drei verschiedene Klassen: Kreisförmig, linear und rechteckig. Die Einteilung erfolgt dabei nach folgender Annahme:

1. Wenn die Gesamtzahl von Datenpunkten eines Blocks als x angenommen wird, und x kleiner ist als fünf, dann wird das Objekt als kreisförmig klassifiziert. Um die Größe des Kreises zu bestimmen, wird als erstes der mittlere Punkt $O(Xo, Yo)$ zwischen p und q bestimmt und als Mittelpunkt des Kreises festgelegt. Von dem Mittelpunkt werden dann jegliche euklidischen Abstände Dj zu den anderen Datenpunkten im Block berechnet. Der größte Abstand $Djmax$ ist dann der Radius des Kreises.

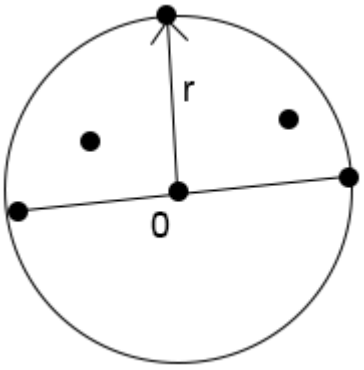

Abbildung 19: Kreisklassifizierung des ODAOA-Algorithmus

$$X_O = \frac{X_p + X_q}{2} \qquad Y_O = \frac{Y_p + Y_q}{2} \tag{1.13}$$

$$r = Djmax \qquad j \in [p, q]$$

2. Falls x größer als fünf ist, wird eine Linie L: $y = ax + b$ gesetzt, welche die Punkte p und q miteinander verbindet. Nun kann der kürzeste Abstand jedes im Block enthaltenen Datenpunktes zur Linie berechnet werden. Der größte Abstand $Dmax$ zwischen einem Datenpunkt und der Linie wird dann ausschlaggebend zur weiteren Klassifizierung verwendet. Für $|S|$, welcher die Euklidische Distanz zwischen p und q beschreibt gilt, wenn $Dmax < 0{,}2 * |S|$ ist, wird das Objekt als Linie klassifiziert. Die beiden Endpunkte der Linie werden dann als $Point1$ und $Point2$ festgelegt.

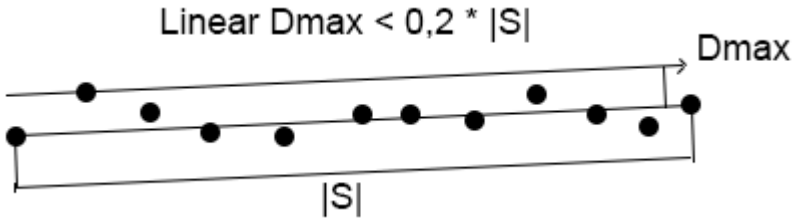

Abbildung 20: Linearklassifizierung des ODAOA-Algorithmus

3. Falls $Dmax \geq 0{,}2 * |S|$ ist, gilt die Annahme, dass das Objekt rechteckig ist. Die Eckpunkte $Point1$, $Point2$, $Point3$ und $Point4$ werden in der Praxis dann durch vier Linien verbunden, die Form und Ausrichtung des Rechtecks beschreiben.

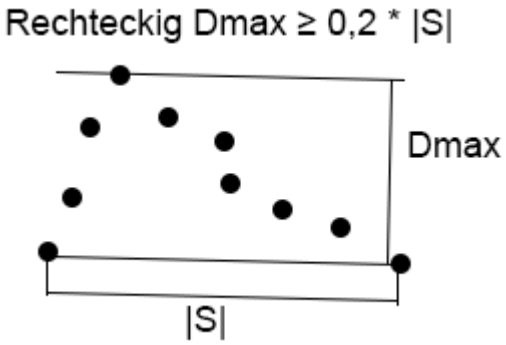

Abbildung 21: Rechteckklassifizierung des ODAOA-Algorithmus

[14], [17]

2 Auswahl Segmentierungs- und Klassifizierungsverfahren

Tabelle 1: Gewichtete Punktebewertung der verschiedenen Verfahren anhand der gegebenen Anforderungskriterien.

Anforderung / **Verfahren**	Erkennen einer unbekannten Anzahl von Objekten	Umsetzung auf Kleinstrechner möglich	Eignung auf 2D Punktewolke von RPLIDAR	Klassifizierung von Kreis und Rechteck	Positionsbestimmung	Aussage über Orientierung
Gewichtung	5	5	5	5	3	2
Kmeans	0	2	1	0	1	0
Dbsacn	3	1	3	0	0	0
Spektrales Clustering	0	2	2	0	0	0
MeanShift	3	1	1	0	1	0
Occupancy Grid + CC-Algorithmus	3	3	3	0	0	0
RBNN-Segmentation	3	2	3	0	0	0
ADAOA- Algorithmus	3	3	3	2	2	3
Hough Transformation	3	2	2	3	2	3
Graham Scan	0	3	3	0	0	3

Um eine sinnvolle Auswahl eines geeigneten Segmentierungsverfahren aufgrund der vorgegebenen Anforderungen zu treffen, wurde das von der TU-Braunschweig vorgestellte Verfahren der „gewichteten Punktebewertung“ [21] gewählt. Dabei werden die gegebenen Anforderungen wie in der Tabelle 1 gelistet und gewichtet. Somit werden die Anforderungen, die der Algorithmus von Anfang an erfüllen sollte, mit fünf gewichtet. Dies betrifft hierbei das Erkennen einer unbekannten Anzahl von Objekten, eine mögliche Umsetzung auf einem Kleinstrechner, die Eignung des Algorithmus in Bezug auf die von RPLIDAR erfassten 2D Punktewolke sowie der Klassifizierungsmöglichkeit der verschiedenen Objekttypen. Die Anforderung der Positionsbestimmung wurde mit drei gewichtet, da eine Aussage dazu nach einer erfolgreichen Segmentierung und Klassifizierung erst einmal unabhängig von der Genauigkeit anhand der segmentierten Punktewolke getroffen werden kann. Die Anforderung, eine Aussage über die Orientierung treffen zu können, ist laut der Aufgabenstellung erwünscht, aber nicht unbedingt notwendig. Daher wurde diese nur mit zwei gewichtet.

Jedes Verfahren erhält dann je nach Erfüllung der einzelnen Anforderungen Punkte zwischen null und drei. Die einzelnen Punkte werden dann mit der jeweiligen Gewichtung multipliziert und anschließend für jede Anforderung miteinander addiert. Das Verfahren, mit den meisten Punkten ist somit am geeignetsten zur Erfüllung der Aufgabenstellung.

Bei der Punktevergabe der Anforderung, eine unbekannte Anzahl von Objekten erkennen zu können, wurden entweder null Punkte für nicht erfüllt, oder drei Punkte für erfüllt vergeben.

Für die Umsetzbarkeit auf dem Kleinstrechner wurden die Punkte abhängig davon vergeben, wie groß der Rechenaufwand des Algorithmus ist, welcher die Umsetzung auf dem Kleinstrechner verlangsamen oder unmöglich machen könnte. Da nicht zu jedem Algorithmus eine explizite Aussage über die Rechenzeit gegeben ist, wurde die Punktevergabe anhand der zu überprüfenden Messpunkte getroffen. Somit erhält ein Verfahren weniger Bewertungspunkte, wenn beispielsweise die Berechnungen für jeden Punkt der Punktewolke mit jedem anderen Messpunkt durchgeführt werden muss. Ein Verfahren, welches von Anfang an die Messpunkte nach bestimmten Kriterien auswählt und nur diese miteinander vergleicht beziehungsweise berechnet, erhält somit mehr Bewertungspunkte, da vergleichsweise weniger Rechnungen durchgeführt werden müssen.

Die Eignung auf die 2D Punktewolke des RPLIDARs wird mit drei Punkten bewertet, wenn die recherchierten Verfahren explizit für die Segmentierung einer Lidarmessung ausgelegt, beziehungsweise an dieser schon beispielhaft angewandt wurden. Ein Punkt wurde vergeben, wenn das Verfahren kein gutes Segmentierungsergebnis im Falle der 2D Punktewolke liefern würde. Ein Algorithmus wie MeanShift, welcher beispielsweise Dichtezentren ermittelt, hätte bei der gegebenen Punktewolke Schwierigkeiten. Das jeweilige Verfahren wurde mit zwei Punkten bewertet, wenn gute Voraussetzungen einer erfolgreichen Segmentierung gegeben sind, das jeweilige Verfahren aber nicht in den recherchierten Quellen für die Objekterfassung mittel 2D Laserscanner verwendet wurde.

Einen Ansatz der expliziten Klassifizierung von Kreis und Rechteck bieten nur die beiden Verfahren *Obstacle Detection and Obstacle Avoidance Algorithm* und *Hough Transformation*. Dabei wurden dem *ODAOA-Algorithmus* aber nur zwei Punkte vergeben, da zwar ein vielversprechender Ansatz beschrieben wird, dieser aber verbessert werden müsste, um ein zufriedenstellendes Ergebnis zu erhalten.

Ähnliches gilt für den *ODAOA-Algorithmus* bei der Positionsbestimmung der Objekte. Wiederum sind Ansätze gegeben, diese müssten aber weiter ausgearbeitet werden. MeanShift und Kmeans geben zwar eine Aussage über die Position der Segmentzentren, diese würden aber bei der 2D Punktewolke zu fehlerhaften Ergebnissen führen.

Als letztes wird noch die Aussage zur Orientierung bewertet. Dabei werden wieder die Verfahren mit null Punkten bewertet, die keine Aussage zur Orientierung beinhalten. Dabei ist bei dem *Obstacle Detection and Obstacle Avoidance Algorithm,* der *Hough Transformation* und dem *Graham Scan* jeweils ein vielversprechender Ansatz gegeben, um die Orientierung der gemessenen Objekte richtig im Raum darstellen zu können.

Als Ergebnis in Tabelle 3 der gewichteten, aber auch der ungewichteten Punktebewertung stechen der *Obstacle Detection and Obstacle Avoidance Algorithm,* aber auch der Ansatz der *Hough Transformation* heraus. Hierbei erzielt der *ODAOA-Algorithmus* eine höhere Punktzahl bei den Anforderungen der Segmentierung. Die *Hough Transformation* schneide insgesamt besser in der Klassifizierung ab. Insgesamt ist aber die Gesamtpunktzahl des *ODAOA-Algorithmus* am höchsten.

Tabelle 2: Gesamtpunktzahl der gewichteten Punktebewertung

Punkte / Verfahren	**Ungewichtet** Einfache Punktebewertung		**Gewichtet** Gewichtete Punktebewertung	
	Summe	Rang	Summe	Rang
Kmeans	4	8	18	9
Dbsacn	7	6	35	6
Spektrales Clustering	4	8	20	8
MeanShift	6	7	28	7
Occupancy Grid + CC-Algorithmus	9	3	45	3
RBNN-Segmentation	8	5	40	4
ODAOA-Algorithmus	16	1	67	1
Hough Transformation	15	2	62	2
Graham Scan	9	3	36	5

Ein weiteres Argument des *ODAOA-Algorithmus* ist, dass die relativ einfache mathematische Beschreibung der Segmentierungs- und Klassifizierungsberechnungen die einzelnen Bearbeitungsschritte verständlicher macht. Dadurch fällt es leichter, kreative Ansätze zur Weiterentwicklung und somit zur Verbesserung des Algorithmus zu entwickeln. Durch die wie in Kapitel 3 erläuterte Umsetzung können dadurch die Schwächen in Hinsicht auf Klassifizierung und Positionierung behoben werden. Demnach kann auch die Bewertung für die genannten Anforderungen von zwei auf drei Punkte angehoben werden. Der Algorithmus erhält dadurch für alle Anforderungen die volle Punktzahl und eignet sich somit sehr gut für die Umsetzung der Aufgabenstellung.

3 Implementierung

Das folgende Kapitel befasst sich mit der Implementierung des recherchierten und in Kapitel 3 ausgewähltem Segmentierungs- und Klassifizierungsverfahren in das ROS-Framework. In Abschnitt 3.1 wird kurz die für das Projekt entworfene Konstruktion vorgestellt. In den darauffolgenden Abschnitten wird zum einen in 3.2 die ROS Umgebung weiter erläutert, sowie in den Abschnitten 3.3 bis 3.6 die Umsetzung und Weiterentwicklung des Algorithmus beschrieben. In 3.7 wird außerdem die Implementierung auf dem Raspberry Pi erläutert.

3.1 Aufbau

Für den Aufbau wurde mittels Catia V5 eine Konstruktion angefertigt und daraufhin mit dem im Labor vorhandenen 3D-Drucker gedruckt. Die Idee der Konstruktion stammt aus dem Youtube-Video „2D Mapping Using Google Cartographer and RPLidar with Raspberry Pi 3B+“ [22] und wurde daraufhin nachkonstruiert und auf die Anforderungen dieser Bachelorarbeit angepasst.

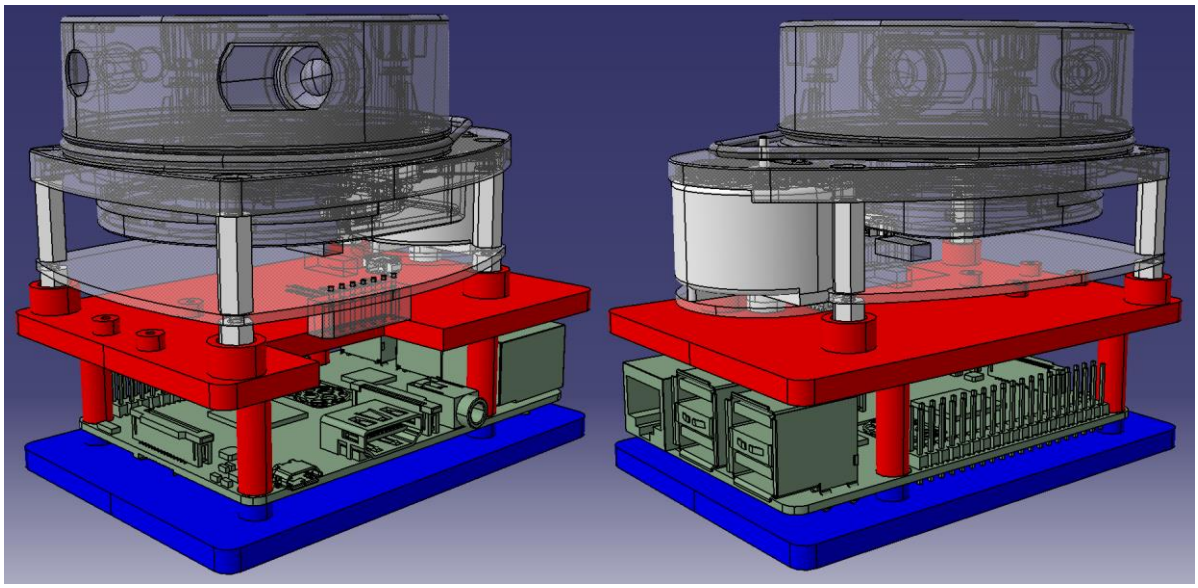

Abbildung 22: Konstruktion und Zusammenbau des Aufbaus in CATIA V5

Der Aufbau besteht dabei neben dem RPLIDAR und dem Raspberry Pi aus den zwei selbstkonstruierten Teilen, welche in der Abbildung 22 in den Farben Rot und Blau dargestellt sind. Das Catia Modell des Sensors [23] und des Kleinstrechners [24] stehen in den beigefügten Quellen frei zur Verfügung und wurden aus Anschauungzwecken in den dargestellten Zusammenbau mit eingefügt. Um die Ausmaße und Abstände korrekt einzuhalten, wurden die technische Zeichnungen aus den jeweiligen Datenblättern verwendet. [32], [33]

Das in Rot dargestellte Bauteil dient als Halterung sowohl für RPLIDAR als auch für den Raspberry Pi. Das blaue Teil fungiert als Standplatte. Sowohl Sensor, als auch Rechner werden jeweils mittels vier M2,5x10mm Schrauben an der Konstruktion verschraubt.

Die Größe des gesamten Zusammenbaus beträgt im verschraubten Zustand 96,5x66x84 Millimeter. Die Messhöhe des Sensors liegt somit, ausgehend von der Standfläche des Aufbaus, bei 75 Millimetern.

3.2 ROS – Umgebung

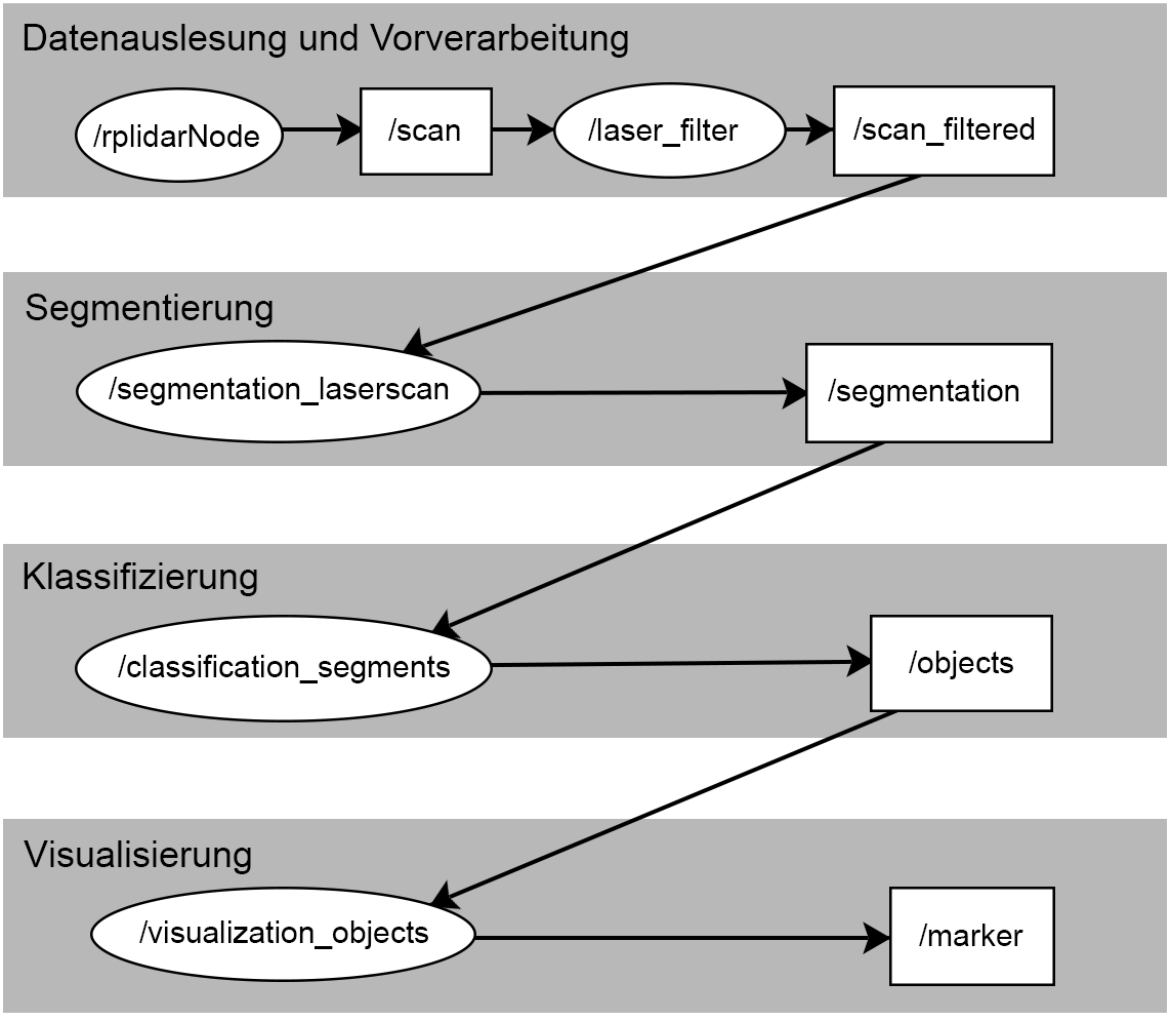

Abbildung 23: Rqt-Graph der erstellten ROS-Umgebung

In Abbildung 23 ist der rqt_graph der erstellen ROS-Umgebung dargestellt. Dieser zeigt die Schritte der Datenverarbeitung, angefangen von den Sensor-Rohdaten bis hin zur visuellen Ausgabe der erkannten Objekte. In der Darstellung werden ROS-Knoten in Ovalform und die ROS-topics als Rechtecke dargestellt. Die Pfeile zwischen den Knoten und topics geben je nach Richtung an, ob die jeweiligen Knoten die Daten auf der unmittelbar mit dem Pfeil verbundenen topic veröffentlichen oder abonnieren.

Alle dargestellten Knoten sind innerhalb des erstellten Workspace *objectsegmentation_ws* angelegt und über Pakete integriert worden. Ein- und Ausgabe sowie Funktion der Knoten werden im Folgenden kurz erläutert.

3.2.1 rplidar_ros

Das Paket *rplidar_ros* wird speziell für die 2D Laser Scanner RPLIDAR A1, A2 und A3 auf [30] bereitgestellt. Es fungiert als Treiber-Paket, um den „Roh-Scan“ des Sensors auszulesen und die Daten in den ROS-Datentyp *LaserScan* umzuwandeln. Der im Paket enthaltene Knoten *rplidarNode* wird ausgeführt und veröffentlicht folgende Daten des *LaserScans*:

Sensor_msgs/LaserScan.msg

```
float32 angle_min          #Start Winkel des Scans [rad]
float32 angle_max          #Stopp Winkel des Scans [rad]
float32 angle_increment    #Winkelschritte zwischen den einzelnen Messpunkten [rad]
float32 time_increment     #Zeit zwischen einzelnen Messungen [sec]
float32 scan_time          #Zeit zwischen Scans [sec]
float32 range_min          #Minimaler möglicher Messabstand [m]
float32 range_max          #Maximaler möglicher Messabstand [m]
float32[] ranges           #Liste der gemessenen Abstände [m]
float32[] intensities      #Lister der Intensität der einzelnen Messpunkte
```

Die *LaserScan* Daten werden von dem Treiber-Paket auf der ROS-topic *scan* veröffentlicht. Es ist ebenfalls möglich, eine visuelle Scanausgabe im ROS-integrierten Visualisierungs-Tool *rviz* anzeigen zu lassen:

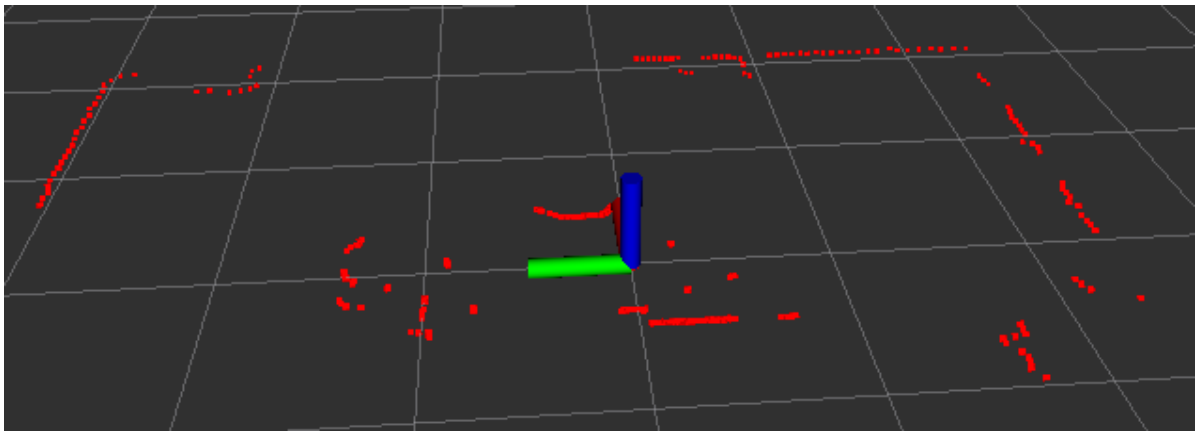

Abbildung 24: Beispiel LaserScan, visualisiert im rviz mit Sensorposition im Achsenursprung.

3.2.2 laser_filters

Als zweiter Datenverarbeitungsschritt in der Kette wird das ROS-Paket *laser_filters* mit eingebunden. Wie im *rqt_graph* dargestellt, greift der gleichnamige Knoten durch das abonnieren der topic *scan* auf die vom Knoten *rplidarNode* bereitgestellten Messdaten zu. Das *laser_filters* Paket kann auf [31] heruntergeladen und für den jeweiligen Anwendungsfall angepasst werden. Es bietet die Möglichkeit der Anwendung verschiedener Filtertypen, welche für die Verarbeitung von ROS-*LaserScan* Daten entwickelt wurden.

Eine weitere wichtige Funktion des Pakets ist die Anwendung von Filterketten. Dabei kann zwischen *scan_to_scan_filter_chain* und *scan_to_cloud_filter_chain* gewählt werden.

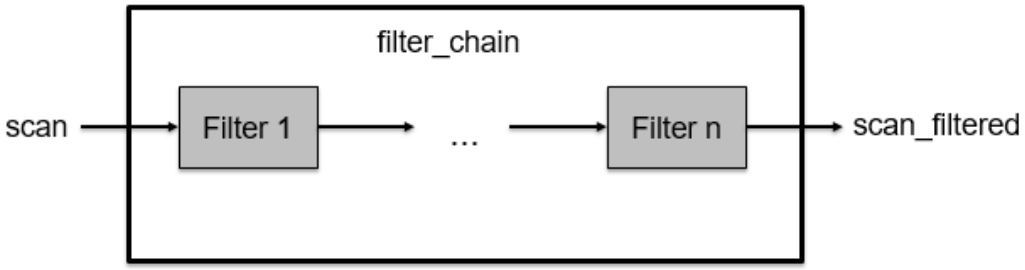

Abbildung 25: scan_to_scan_filter_chain des ROS-Pakets laser_fitlers

Durch die Filterketten können die von dem Paket bereitgestellten Filtertypen in einer Kette hintereinander angewendet werden, um das gewünschte Ergebnis zu erzielen. Der wesentliche Unterschied der beiden Ketten liegt in der Ausgabe. Die *scan_to_scan_filter_chain* gibt die gefilterten Daten wiederum als *LaserScan* Datentyp aus. Bei der *scan_to_cloud_filter_chain* werden die Eingangs-LaserScan Daten zusätzlich zu der Filterbearbeitung noch in den ROS Datentyp *PointCloud* umgewandelt. Welcher Datentyp sich als Ausgabe als Sinnvoll erweist, muss individuell an die Aufgabenstellung angepasst werden. Im Laufe dieses Projektes wurde sich für die Ausgabe als *LaserScan* entschieden, die dieser die Messpunkte in Polarkoordinaten ausgibt, welche bei dem gewählten Segmentierungsansatz verwendet werden.

Als Filterung wurde ausschließlich der *Median Filter* angewendet. Dieser vergleicht mehrere aufeinander folgende Messungen miteinander und erkennt somit Ausreißer, welche ungewöhnlich von den anderen Messungen abweichen. Die Daten des *laser_filters* werden dann auf der topic *scan_filtered* veröffentlicht.

3.2.3 segmentation_laserscan

Das Paket *segmentation_laserscan* im Rahmen dieses Projekts wurde selbst angelegt und der Programmcode in der Programmiersprache Python verfasst. Angelehnt und weiter ausgearbeitet werde der Segmentierungsansatz an den in 1.4.5 und 1.5.3 vorgestellten *ODAOA-Algorithmus* [14], für den sich aufgrund der in Kapitel 2 beschriebenen Kriterien entschieden wurde.

Dabei wird auf die gefilterten *LaserScan* Daten von *scan_filtered* zugegriffen und diese weiterverarbeitet. Die genaue Umsetzung der Segmentierung wird in Kapitel 3.3 ausführlich beschrieben. Für die Ausgabe der Segmentierung wurde der *custom message type - segmentation_msg* erstellt. Dadurch werden anschließend an die Segmentierung von dem Knoten *segmentation_laserscan* folgende Daten auf der topic *segmentation* veröffentlicht:

segmentation_laserscan/segmentation_msg.msgs

```
uint8 amount_segments        #Anzahl der gefundenen Segmente
float32 angle_increment      #Winkelschritte zwischen den einzelnen Messpunkten [rad]
float32[] ranges             #Liste der gemessenen und verarbeiteten Abstände [m]
int32[] start_points         #Liste der Startpunkt der Segmente
int32[] stop_points          #Liste der Stopppunkte der Segmente
```

3.2.4 classification_segments

Die segmentierten Daten werden dann von dem Knoten *classification_segments* abonniert, welcher ebenfalls in Python codiert und in einem gleichnamigen Paket angelegt wurde. Der Klassifizierungsansatz ist ebenfalls an den Ansatz aus des *ODAOA-Algorithmus* angelehnt und daraufhin weiterentwickelt worden.

Aus den Start- und Stopppunkten können dann für jedes Segment die einzelnen Kriterien für die jeweilige Formklassifizierung überprüft werden und die entsprechenden Segmente in kreisförmig, linear oder rechteckig eingeteilt werden. Segmente, die aufgrund der festgelegten Kriterien nicht eingeteilt werden können, werden verworfen. Um die Berechnung auf Vektorebene möglich zu machen, werden die einzelnen Messpunkte als erstes von Polarkoordi-

naten in kartesische Koordinaten umgewandelt. Die einzelnen Kriterien sowie die mathematische Vorgehensweise werden in 3.4 ausführlich erläutert. Der Knoten veröffentlicht dann Informationen wie Form, Position und Ausrichtung (resultierend aus den Koordinaten der veröffentlichten Punkte) der einzelnen Objekte über den selbst erstellten *custom message type - objects_msg* auf der topic *objects*:

classification_segments/objects_msg.msgs	
float32[] lines_start_x	#x-Koordinate der Linien-Startpunkte
float32[] lines_start_y	#y-Koordinate der Linien-Startpunkte
float32[] lines_stop_x	#x-Koordinate der Linien-Stopppunkte
float32[] lines_stop_y	#y-Koordinate der Linien-Stopppunkte
float32[] lines_closest_point_x	#x-Koordinate des Linienpunktes mit dem geringsten Abstand zum Sensor
float32[] lines_closest_point_y	#y-Koordinate des Linienpunktes mit dem geringsten Abstand zum Sensor
float32[] lines_middle_point_x	#x-Koordinate des Linienmittelpunktes
float32[] lines_middle_point_y	#y-Koordinate des Linienmittelpunktes
float32[] circles_center_x	#x-Koordinate der Kreismittelpunkte
float32[] circles_center_y	#y-Koordinate der Kreismittelpunkte
float32[] circles_radius	#Kreisradien
float32[] rectangles_corner_start_x	#x-Koordinate der Rechteck-Ecken, welche durch die Segment-Startpunkte entstehen
float32[] rectangles_corner_start_y	#y-Koordinate der Rechteck-Ecken, welche durch die Segment-Startpunkte entstehen
float32[] rectangles_corner_max_distance_x	#x-Koordinate der Rechteck-Ecken, welche durch den jeweiligen Segmentpunkt mit dem größten Abstand zum Start-Stopp Vektor entstehen.
float32[] rectangles_corner_max_distance_y	#y-Koordinate der Rechteck-Ecken, welche durch den jeweiligen Segmentpunkt mit dem größten Abstand zum Start-Stopp Vektor entstehen
float32[] rectangles_corner_stop_x	#x-Koordinate der Rechteck-Ecken, welche durch die Segment-Stopppunkte entstehen
float32[] rectangles_corner_stop_y	#x-Koordinate der Rechteck-Ecken, welche durch die Segment-Stopppunkte entstehen
float32[] rectangles_corner_max_distance_mirror_x	#x-Koordinate der Rechteck-Ecken, welche durch die Spiegelung der „max_distance_points“ über den Start-Stopp Vektor entstehen.
float32[] rectangles_corner_max_distance_mirror_y	#y-Koordinate der Rechteck-Ecken, welche durch die Spiegelung der „max_distance_points“ über den Start-Stopp Vektor entstehen.
float32[] rectangles_center_x	#x-Koordinate der Rechteck Mittelpunkte
float32[] rectangles_center_y	#y-Koordinate der Rechteck Mittelpunkte

Eine ausführlichere Erklärung der Festlegung beziehungsweise Berechnung der einzelnen Punkte findet sich im Kapitel 3.5.

3.2.5 visualization_objects

Als letzten Verarbeitungsschritt folgt die Visualisierung der Objekte über den Knoten *visualization_objects* (programmiert in Python). Über den Ros-Datentyp *Marker* können verschiedene Formen im *rviz* visualisiert werden. Dabei lassen sich Parameter wie Objektzentrum, Größe (bestimmt durch Aufspannung der Form, ausgehend von dem Objektzentrum, in Richtung x,y,z) und Orientierung der Objekte festlegen. Die Marker-Informationen werden dann von dem Knoten auf der topic */marker* veröffentlicht und lassen sich zur Visualisierung einfach im *rviz* auswählen.

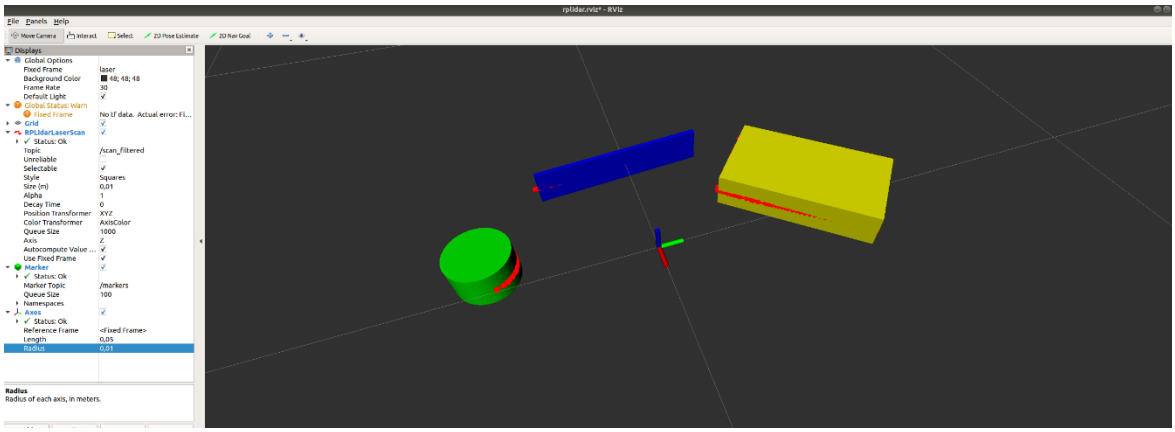

Abbildung 26: Visualisierung der Objekte im rviz

Eine genauere Beschreibung der Umsetzung folgt im Kapitel 3.6.

3.3 Umsetzung des Segmentierungsansatzes

Wie in 3.2.1 bereits beschrieben werden die vom Laser erfassten Daten durch das Treiberpaket *rplidar_ros* als Ros-Datentyp *LaserScan* ausgegeben. Diese werden daraufhin durch den über das ROS-Paket *laser_filter* implementierten Medianfilter auf Ausreißer untersucht und diese entfernt. Der Medianfilter vergleicht dafür mehrere aufeinanderfolgende Messwerte. Aus diesen erkennt er Messausreißer, welche nach der Filterung nicht mehr weiter beachtet werden. Beim Ausführen des Medianfilters kann die Anzahl der zu vergleichenden Messwerten festgelegt werden. Dieser Parameter wurde in diesem Projekt mit fünf Messungen festgelegt.

Die *LaserScan* Daten, welche nach der Filterung auf der topic *scan_filtered* veröffentlicht werden, werden von dem Knoten *segmentiation_laserscan* abonniert. Aus den *LaserScan*-Daten resultiert, dass der minimale Messabstand bei 0,15 Metern und der maximale Messabstand bei zwölf Metern liegt. Aus dem Datenblatt des Sensors ist aber zu entnehmen, dass der Sensor im 360° Betrieb nur Entfernungen bis sechs Meter messen kann. Ebenfalls zu beachten ist, dass der Messwinkel ein Bogenmaß von $-\pi$ bis $+\pi$ besitzt, und somit wie erwartet einen Messbereich von 360° abdeckt.

Bei der weiteren Betrachtung der Messergebnisse wurde wie in Abbildung 27 gezeigt, das standardisierte Fahrzeugkoordinatensystem gewählt.

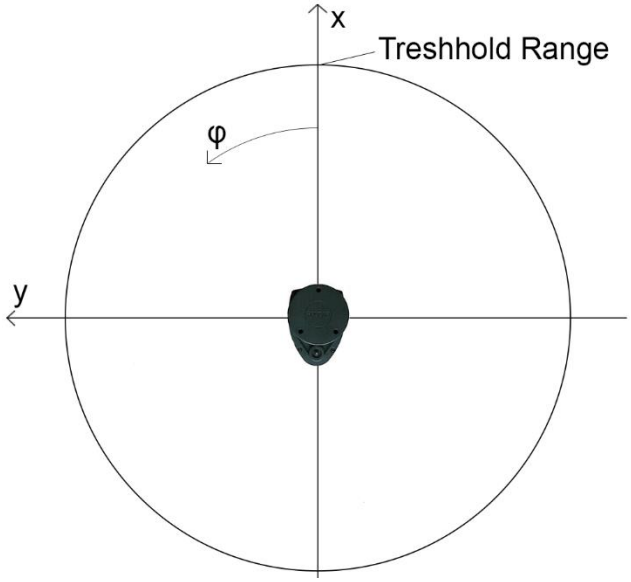

Abbildung 27: RPLIDAR Sensor im Koordinatensystem

Für die anschließende Segmentierung werden primär die von der topic abonnierten Werte *ranges* und *angle_increment* verwendet. *Ranges* wird dabei als Liste, bestehend aus 360 Messwerten übergeben. Jeder der innerhalb der Liste übergebenen Werte beschreibt eine gemessene Reichweite innerhalb des Messbereichs. *Angle*_increment besagt, wie groß die Winkelschritte zwischen den einzelnen Messpunkten und somit zwischen den einzelnen Werten der Liste sind. Dadurch können die Messwerte im Raum platziert werden. Der Wert von *anlge_increment* wird dabei als Radiant übergeben. Wie sich aus den 360 Messwerten schon schließen lässt, ist der übergebene Wert von *angle_increment* $\varphi = 0{,}01745\ldots$ rad. Dies entspricht einem Winkelschritt von genau $1°$.

Im folgenden Verlauf wird die Liste *ranges* mit der Abkürzung $R(i)$ dargestellt. Wobei $i \in [0,359]$ die Position des Messwertes in der Liste beschreibt. Somit ist $R(i=0)$ der erste Wert und parallel dazu $R(i=359)$ der letzte Wert der Liste.

Nach dem festgelegten Koordinatensystem befindet sich der Punkt $R(i=0)$ genau auf der x-Achse, oder bezogen auf die Fahrzeugtechnik in Fahrtrichtung. Die weiteren Messpunkte folgen dann entgegengesetzt des Uhrzeigersinns, wie in der Abbildung 27 durch den Winkel φ dargestellt. Somit befindet sich der Punkt $R(i=90)$ genau auf der y-Achse.

3.3.1 Segmentierungsschritte

Auf der Abbildung 28 ist links eine Beispielmessung des RPLIDAR dargestellt. Dabei ist zu beachten, dass diese nicht aus realen Messwerten angefertigt wurde, sondern einer reinen Skizzendarstellung entspricht, welche die Segmentierungsschritte veranschaulichen soll. Die Gesamtanzahl der Messpunkte sowie die Winkelschritte zwischen den Punkten sind somit in der Darstellung zufällig entstanden. Angelehnt ist die Beispielskizze an eine Messung in einem geschlossenen Raum, in dessen Mitte der Sensor platziert wurde. Im unmittelbaren Umfeld des Sensors befinden sich dabei unterschiedlich geformte Objekte. Im Folgenden werden die einzelnen Segmentierungsschritte einer Messung erläutert.

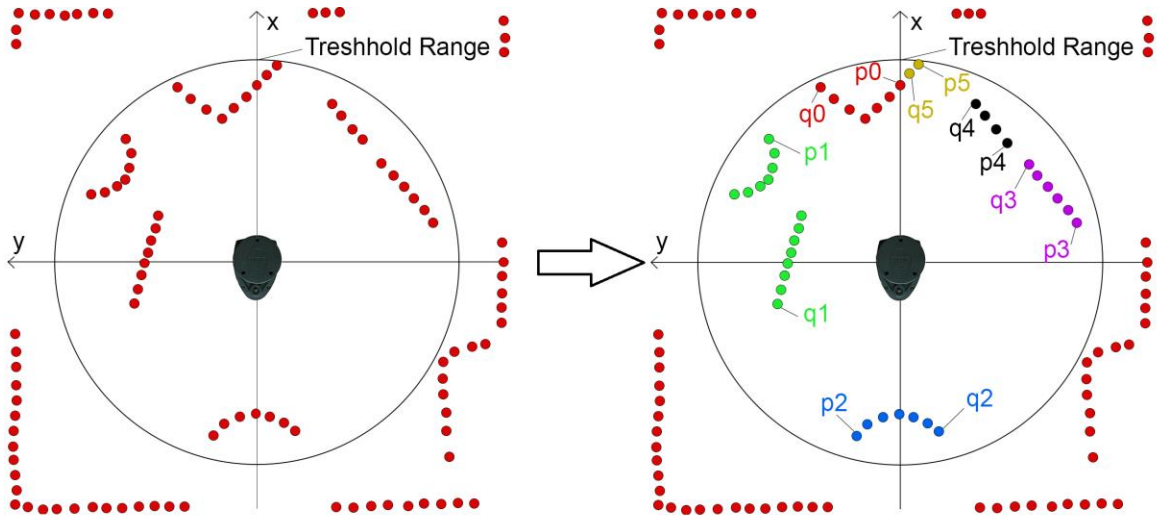

Abbildung 28: Links: Beispiel Messung Sensor, rechts: Segmentierungsschritt 1

1. Das Ergebnis des ersten Segmentierungsschritts ist rechts in Abbildung 28 dargestellt. Dafür wird als erstes im Programmcode der Parameter *threshold_range* festlegt. Dieser bestimmt den maximalen Abstand zum Sensor, innerhalb dem die verschiedenen Segmente gebildet werden sollen. Soll der gesamte Messbereich des Sensors beachtet werden, so wird der Parameter gleich der maximalen Reichweite des Sensors gesetzt.

Anschließend werden die Messwerte in verschiedene Blöcke eingeteilt. Dafür werden die i-Werte, also die Position der jeweiligen Messpunkte innerhalb der *ranges* Liste auf Basis der unten aufgelisteten Bedingungen untersucht. Es wird entschieden, bei welchen der jeweiligen Messpunkte von *range* der Block startet (Startwerte p), bzw. der Block endet (Stoppwerte q). Die i-Werte werden zur weiteren Datenverarbeitung in jeweils eigene Listen für Start- und Stoppwerte eingespeichert. Die Bezeichnungen *p* für Startwerte und *q* für Stoppwerte werden in der weiteren Erläuterung beibehalten. Über die folgenden Bedingungen werden die Start- sowie Stopppunkte der einzelnen Blöcke gesetzt:

Bedingungen für das Setzen eines Startpunktes:

$R(0) <= threshold\_range$ → Startpunkt p bei $R(i = 0)$ (3.1)

$R(i) > threshold\_range \;\&\&\; R(i+1) <= threshold\_range$ (3.2)
→ Startpunkt p bei $R(i+1)$

Bedingungen für das Setzen eines Stopppunktes:

$R(i) <= threshold\_range \;\&\&\; R(i+1) > threshold\,range$ (3.3)
→ Stopppunkt q bei $R(i)$

$R(last\_range) <= threshold\_range$ (3.4)
→ Stopppunkt q bei $R(last\_range)$

Das bedeutet, dass ein Startwert mit $p = i$ genau dann gesetzt wird, sobald die Reichweite $R(i)$ eine kleinere Entfernung als *threshold_range* zum Sensor aufweist, wobei die Messung $R(i-1)$ noch nicht in der betrachteten Reichweite liegt. Parallel dazu wird ein Stopppunkt

mit $q = i$ belegt, wenn sich die Messung bei $R(i)$ innerhalb der Reichweite und der nächste Messwert $R(i+1)$ außerhalb dieser befindet. Ebenfalls müssen der erste Wert bei $R(0)$, als ein möglicher Startwert und der letzte Wert der *ranges* Liste bei $R(last\_range)$, als möglicher Stoppwert geprüft werden, da diese in den Bedingungen (3.2) und (3.3) nicht mit eingeschlossen werden. Der i-Wert für *last_range* passt sich dabei der Anzahl der Messwerte innerhalb der abonnierten *ranges* Liste an. Im Falle der verwendeten RPLIDAR Messdaten ist $last\_range = 359$, da genau 360 Messwerte abonniert werden.

Nach Schritt 1 der Segmentierung ergibt sich somit eine erste Einteilung der Messpunkte in Blöcke, welche ein potentielles Segment darstellen könnten. Alle Messwerte, die außerhalb von *threshold_range* liegen werden somit zunächst nicht weiter beachtet. Jeder der Blöcke ist durch einen jeweiligen Start- sowie Stopppunkt eingegrenzt und beinhaltet somit alle Punkte, welche sich zwischen den Start- und Stopppunkten der einzelnen Blöcke befinden.

Die Einteilung in die einzelnen Blöcke wird in Abbildung 28 farbig dargestellt. Ebenfalls sind die Startpunkte p und Stopppunkte q der einzelnen Blöcke gekennzeichnet. Betrachtet man die einzelnen Blöcke, fallen ein paar Dinge auf, die in den weiteren Segmentierungsschritten verarbeitet werden müssen:

- Der grüne Block, welcher durch den Startpunkt $p1$ und den Stopppunkt $q1$ eingegrenzt wird, bildet sich offensichtlich aus zwei verschiedenen Objekten. Da aber zwischen den Objekten kein Messwert einen größeren Abstand als *threshold_range* aufweist, werden beide Objekte zu einem Block zusammengefasst.
- Der violette Block ($p3$ bis $q3$) und der schwarze Block ($p4$ bis $q4$) liegen sehr nahe beieinander. Beide Blöcke wurden getrennt, weil zwischen $q3$ und $p4$ mehrere Messwerte außerhalb von *threshold_range* liegen. Somit könnten beide Blöcke ein einzelnes Objekt darstellen, welches, wie zum Beispiel ein Gartenzaun, nicht durchgängig ist. Ebenfalls könnte es sich um zwei Objekte handeln, die sehr nahe nebeneinander platziert wurden.
- Der rote Block ($p0$ bis $q0$) und der gelbe Block ($p5$ bis $q5$) bilden offensichtlich zusammen ein Objekt, werden aber nach den Bedingungen in Schritt 1 in zwei unterschiedliche Blöcke geteilt.

2. In der Abbildung 29 ist das Ergebnis des zweiten Segmentierungsschrittes (rechts) als Grundlage des ersten Segmentierungsschrittes (links) dargestellt.

Der zweite Segmentierungsschritt beinhaltet primär die Überprüfung der Euklidischen Abstände zwischen den Stopppunkten $q(j)$ und den Startpunkten $p(j+1)$, wobei j den jeweils zu betrachtenden Block beschreibt. Das heißt, dass die Abstände zwischen dem Stopppunkt des einen Blocks mit dem Startpunkt des nächsten Blocks überprüft werden. Ebenfalls werden die Abstände zwischen dem letzten Stopppunkt (hier $q5$) und dem ersten Startpunkt $p0$ berechnet. Wie auch in der Abbildung 29 links auch dargestellt, liegen diese aufgrund der Messung von 360° oft unmittelbar nebeneinander und gehören somit zum selben Objekt. Die Berechnung der Euklidischen Abstände wird in Abschnitt 1.4.1 beschrieben.

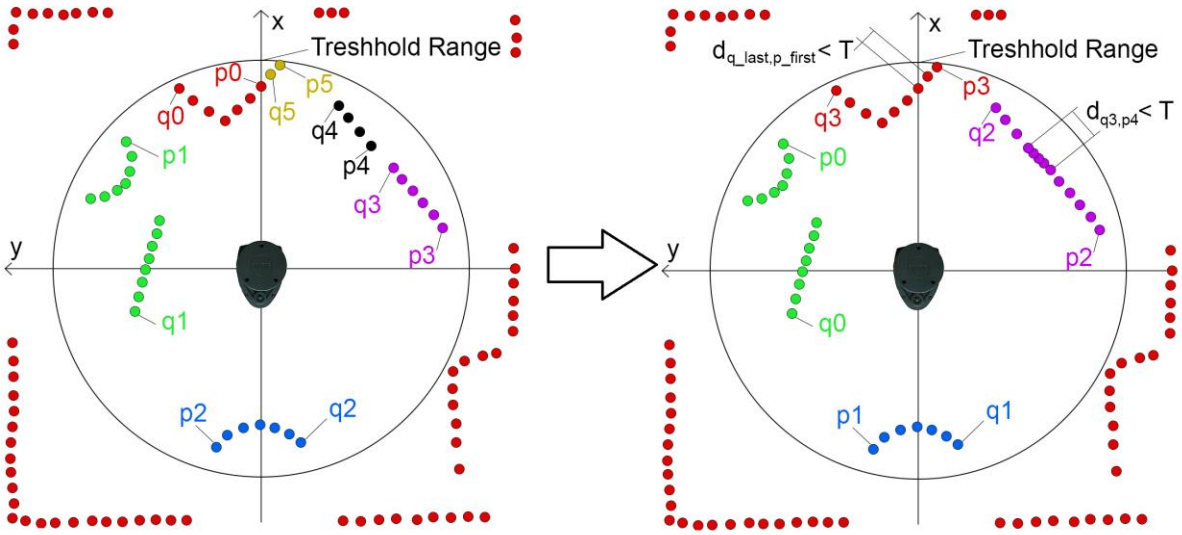

Abbildung 29: Links: Segmentierungsschritt 1, rechts: Segmentierungsschritt 2

Ist der berechnete Abstand d dabei kleiner als ein der Grenzwert *threshold_connect*, so werden die jeweiligen Punkte $q(j)$ und $p(j+1)$ bzw. der letzte Stopppunkt und erste Startpunkt aus den jeweiligen Listen gelöscht und die Blöcke somit verbunden. Der Wert *threshold_connect* ist dabei aufgrund von der mit der Entfernung anwachsenden Messtoleranz des Sensors von der durchschnittlichen Entfernung $\bar{R}_{qp}$ der beiden zu verbindenden Punkte abhängig. Der Wert *threshold_connect* berechnet sich wie folgt:

$$\bar{R}_{qp} = \frac{R(q(j))+R(p(j+1))}{2}. \tag{3.5}$$

$$\text{wenn } \bar{R}_{qp} \leq 1m \quad \rightarrow \quad threshold\_connect = 0{,}1m \tag{3.6}$$

$$\text{wenn } \bar{R}_{qp} > 1m \quad \rightarrow \quad threshold\_connect = 0{,}05 * \bar{R}_{qp} + 0{,}05m \tag{3.7}$$

Als weiterer Schritt wird überprüft, ob sich zwischen den verbundenen Blöcken in der Liste *ranges* Messpunkte befinden (z.B. zwischen $q3$ und $p4$ in Abbildung 29). In diesem Fall werden die Messwerte der dazwischenliegenden Punkte mittels folgender Gleichung überschrieben:

$$R(i) = R(q(j)) + i * \frac{R(p(j+1) - R(q(j))}{num+1} \tag{3.8}$$

mit $i \in [1, num]$

Hierbei entspricht num der Anzahl der enthaltenen Messpunkte zwischen den zu verbindenden Blöcken. Dadurch entsteht zwischen Stopp- und Startpunkt eine Gerade von Messpunkten, welche alle den gleichen Abstand zueinander aufweisen. Dies ist in der Abbildung 29 rechts bildlich in dem violetten Block dargestellt.

Durch den Segmentierungsschritt 2 werden die Blöcke, die vermutlich zu dem gleichen Objekt gehören zusammengefasst und Messwerte aus der Liste *range* so verändert, dass zwischen den zu verbindenden Blöcken eine gerade Linie von Messwerten entsteht. Dies ist ein Verarbeitungsschritt, der das Ergebnis der späteren Klassifizierung verbessern soll.

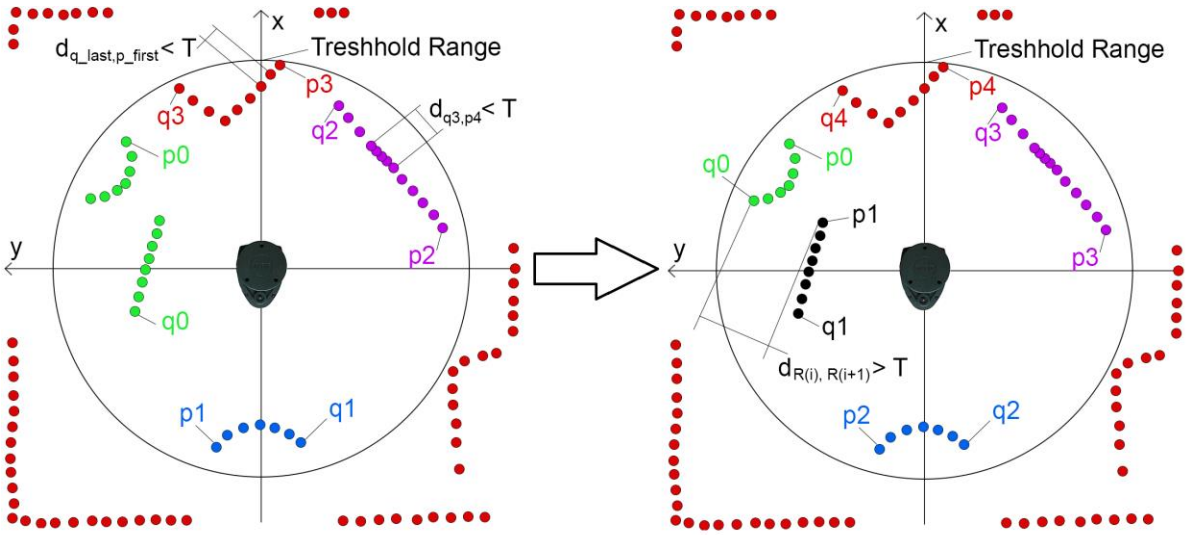

Abbildung 30: Links: Segmentierungsschritt 2, rechts: Segmentierungsschritt 3

3. In der Abbildung 30 ist das Ergebnis des dritten Segmentierungsschrittes (rechts) als Grundlage des zweiten Segmentierungsschrittes (links) dargestellt.

Dieser beinhaltet die Überprüfung aller Euklidischen Abstände zwischen $R(i)$ und $R(i+1)$ in der Reichweite der jeweiligen Block Start- und Stopppunkte $p(j)$ und $q(j)$. Das heißt, dass alle Abstände der benachbarten Punkte innerhalb eines Blockes überprüft werden. Ist der errechnete Abstand größer als der Grenzwert *threshold_divide*, wird der Block zwischen den jeweiligen Punkten i und $i+1$ geteilt und somit der Stopppunkt Liste der Wert i und der Startpunkt Liste der Wert $i+1$ hinzugefügt. Der Wert *threshold_divide* ist dabei von dem durchschnittlichen Abstand $\bar{R}_{i,i+1}$ der zu überprüfenden Punkte abhängig. Dieser berechnet sich nach demselben Prinzip wie der Durchschnittswert von $\bar{R}_{qp}$ in (3.5). Für den Wert *threshold_divide* gelten dann die gleichen Bedingungen wie in (3.6) und (3.7) beschrieben.

Sind zwei Objekte, wie im Falle des grünen Blocks in Abbildung 30 links dargestellt, hintereinander positioniert, sodass sie aufgrund von Schritt 1 zu einem Block zusammengefasst werden, werden diese durch den dritten Segmentierungsschritt in zwei Blöcke aufgeteilt. Dadurch entstehen, wie in der Abbildung rechts, der grüne und der neue schwarze Block.

4. Als finalen Schritt der Segmentierung werden Blöcke, die eine geringere Anzahl, als der im Programm festgelegten Wert *threshold_points* an Messpunkten besitzen, aus den Segmentlisten entfernt. Aufgrund der zu geringen Anzahl von Segmentpunkten kann in der weiteren Datenverarbeitung keine hinreichende Aussage zu der Klassifizierung und somit der Form des Objektes getroffen werden. Somit werden diese Blöcke für die Klassifizierung nicht weiter beachtet. Im diesem Projekt wurde der Wert *threshold_points* mit mindestens fünf Messpunkten für ein Segment festgelegt.

Als finale Ausgabe der Segmentierung kann also zum einen gesagt werden, wie viele potenzielle Objekte sich im gewählten Umkreis des Sensors befinden, an welchem Punkt der *ranges* Liste die einzelnen Segmente beginnen, sowie an welchem Punkt diese aufhören. Zusätzlich dazu wird ebenfalls die teilweise überarbeitete *ranges* Liste auf der topic *segmentation* veröffentlicht.

3.4 Umsetzung des Klassifikationsansatzes

Wie in 3.2.4 schon beschrieben, werden die zu klassifizierenden Segmentdaten von der topic *segmentiation* abonniert. Ziel der Klassifizierung ist es dann zu entscheiden, ob die gefundenen Segmente rechteckig oder kreisförmig sind. Ebenfalls wurde sich dafür entschieden zu untersuchten, ob die Segmente linear sind. Der Klassifizierungsansatz des *ODAOA-Algorithmus* bietet gute Ansätze, führt aber bei der Kreis- bzw. Rechteckklassifizierung nicht zu zuverlässigen Ergebnissen. Im Laufe dieser Bachelorarbeit wurde daher die Grundlagen des ODAOA-Algorithmus übernommen und auf mathematischer Ebene weiterentwickelt.

Um die mathematische Beschreibung der einzelnen Punkte innerhalb der Segmente zueinander zu vereinfachen, wird dafür im weiteren Verlauf der Ansatz der Vektorrechnung verwendet. Dafür werden als erster Arbeitsschritt die abonnierten Segmentpunkte über die *ranges* Liste von Polarkoordinaten in kartesische Koordinaten umgerechnet.

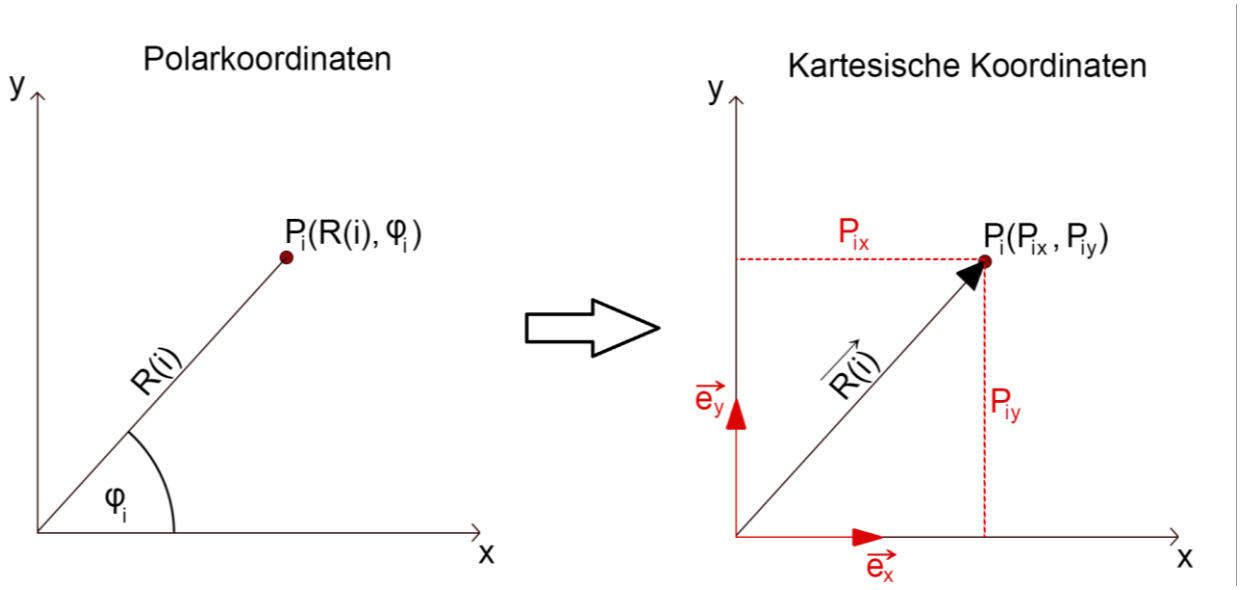

Abbildung 31: links: Beispiel Polarkoordinaten, rechts: Beispiel kartesische Koordinaten

Dies bedeutet, dass für jeden Punkt P_i, der laut der Start,- und Stopppunktliste Teil eines Segmentes ist, von der Form in Polarkoordinaten $P_i(R(i), \varphi_i)$ in die Form der Kartesischen Koordinaten $P_i(P_{ix}, P_{iy})$ umgerechnet wird. Für den Winkel φ_i gilt dabei für jeden Punkt P_i:

$$\varphi_i = i * angle\_increment \tag{3.10}$$

Wobei i wiederum durch die Position des Messwerts in der *ranges* Liste festgelegt wird. Für die Punkte in Polarkoordinaten gilt dann:

$$P_{ix} = R(i) * \cos\varphi_i \tag{3.11}$$

$$P_{iy} = R(i) * \sin\varphi_i \tag{3.12}$$

Für die Klassifizierung dies ODAOA-Algorithmus wie in 1.5.3 beschrieben werden die Abstände aller Punkte zwischen p und q im rechten Winkel zu einer Linie, welche durch p und q läuft geprüft. Diese Linie wird in [14] durch die Geradengleichung in der Normalform

L: $y = ax + b$ beschrieben. Bei dieser besteht aber das Problem, dass Geraden mit einer „unendlichen“ Steigung, das heißt parallel zur y-Achse liegend, nicht beschrieben werden können. Deshalb wurde sich im Laufe dieser Arbeit für die Parameterform [18] entschieden, welche wie folgt beschrieben wird:

$$L: \vec{x} = \vec{a} + \lambda * \vec{u} \tag{3.13}$$

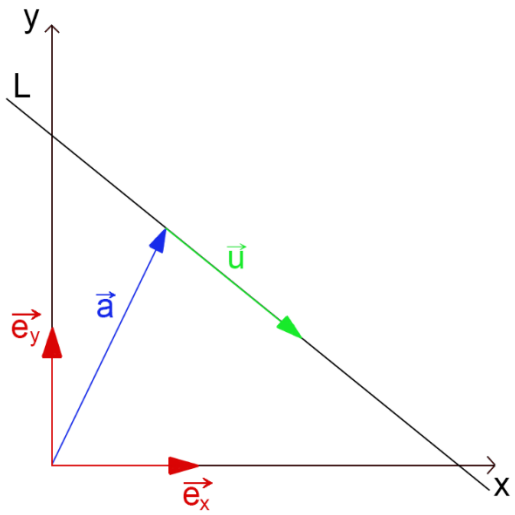

Abbildung 32: Darstellung einer Geraden L in Parameterform

Dabei steht der Vektor $\vec{a}$ für den Stützvektor, $\vec{u}$ für den Richtungsvektor und λ für einen Parameter, welcher angibt, um welchen Punkt auf der Gerade es sich handelt. Auf die Gerade zwischen einem Startpunkt p und einem Stopppunkt q bezogen, ergibt sich dann:

$$L: \vec{x} = \vec{p} + \lambda * \vec{u} \quad \text{mit} \quad \vec{u} = \vec{q} - \vec{p} \tag{3.14}$$

Anschließend kann über die Vektorrechnung mittels Lotpunktverfahren [19] der maximale Abstand $Dmax$ berechnet werden. Dieser resultiert aus den jeweiligen Abständen aller Messpunkte P_i im rechten Winkel zu der Gerade L, die zwischen dem Start- und Stopppunkt des jeweiligen Segments gezogen wird. Für die Messpunkte gilt dabei $P_i \, \epsilon \, [p_j + 1, \; q_j - 1]$, wobei j wiederum für die einzelnen Segmente steht.

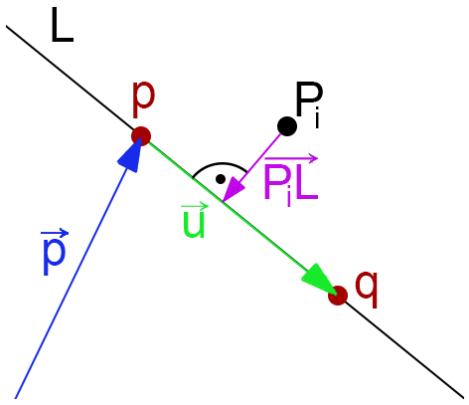

Abbildung 33: Ansatz des Lotpunktverfahrens für einen Punkt P_i.

Durch das Lotpunktverfahren kann der Abstand D_i des Punktes P_i im rechten Winkel zu der Geraden L berechnet werden. Der Abstand D_i ergibt sich dabei aus dem Betrag des Vektors $\overrightarrow{P_iL}$, welcher ausgehend von dem Punkt P_i, auf den von ihm den kürzesten Abstand aufweisenden Punkt der Gerade L zeigt. Dieser Punkt der Gerade wird der Lotpunkt $\vec{L}$ genannt. Da der Vektor $\overrightarrow{P_iL}$ senkrecht auf dem Richtungsvektor $\vec{u}$ liegt, muss das Skalarprodukt der beiden Vektoren gleich Null sein.

$$\overrightarrow{P_iL} \bullet \vec{u} = 0 \tag{3.15}$$

Gesucht wird somit der Wert λ_i, welcher die Position des Lotpunktes auf der Gerade beschreibt. Der Vektor $\overrightarrow{P_iL}$ kann dann einfach als Richtungsvektor von $\vec{P_i}$ zu $\vec{L}$ beschrieben werden. Durch das Einsetzen der Geradengleichung für $\vec{L}$ kann auf den Parameterwert λ_i umgestellt werden, so erhält man die Position des jeweiligen Lotpunktes.

$$\overrightarrow{P_iL} = \vec{L} - \vec{P_i} = \vec{x} - \vec{P_i} = \vec{p} + \lambda_i * \vec{u} - \vec{P_i} = \begin{pmatrix} p_x \\ p_y \end{pmatrix} + \lambda_i * \begin{pmatrix} q_x - p_x \\ q_y - p_y \end{pmatrix} - \begin{pmatrix} P_{ix} \\ P_{iy} \end{pmatrix} \tag{3.16}$$

Durch Einsetzen von (3.16) in (4.15) ergibt sich dann:

$$\left(\vec{p} + \lambda_i * \vec{u} - \vec{P_i}\right) \bullet \vec{u} = \left[\begin{pmatrix} p_x \\ p_y \end{pmatrix} + \lambda_i * \begin{pmatrix} q_x - p_x \\ q_y - p_y \end{pmatrix} - \begin{pmatrix} P_{ix} \\ P_{iy} \end{pmatrix}\right] \bullet \begin{pmatrix} q_x - p_x \\ q_y - p_y \end{pmatrix} = 0 \tag{3.17}$$

Durch die Berechnung der einzelnen Vektoren und das Umstellen auf λ_i ergibt sich für den Parameter dann:

$$\lambda_i = \frac{-[(p_x - P_{ix}) * (q_x - p_x) + (p_y - P_{iy}) * (q_y - p_y)]}{(q_x - p_x)^2 + (q_y - p_y)^2} \tag{3.18}$$

Über λ_i kann dann der Lotpunkt $\vec{L}$ auf der Geraden durch einsetzten in die Geradengleichung ermittelt und somit der Richtungsvektor $\left|\overrightarrow{P_iL}\right| = \vec{L} - \vec{P_i}$ berechnet werden. Der Abstand D_i ergibt sich anschließend aus dem Betrag des Vektors über:

$$D_i = |\overrightarrow{P_iL}| = \sqrt{(p_x - P_{ix} + \lambda_i * q_x - \lambda_i * p_x)^2 + (p_y - P_{iy} + \lambda_i * q_y - \lambda_i * p_y)^2} \tag{3.19}$$

Die Abstände D_i aller Segmentpunkte, welche zwischen dem Start- und Stopppunkt des jeweiligen Segmentes liegen, werden dann miteinander verglichen. Der Punkt mit dem größten Abstand D_{max} wird dann als *max_distance_point* festgelegt. Im Verlauf dieser Arbeit wird dieser Punkt als Punkt g gekennzeichnet.

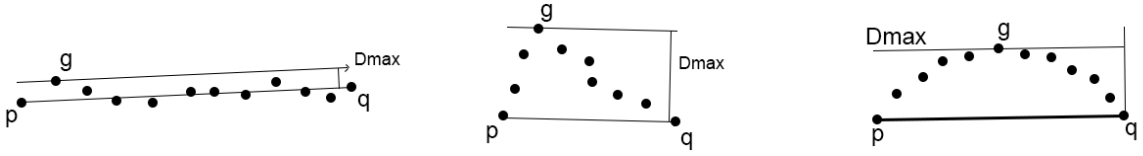

Abbildung 34: max_distance_point g bei den verschiedenen Objekttypen Linie (links), Rechteck (Mitte), Kreis (rechts)

Dabei liefert der Klassifizierungsansatz des ODAOA-Algorithmus für die Linienklassifikation ein zufriedenstellendes Ergebnis. Die Klassifikation von rechteckigen sowie kreisförmigen Objekten kann nach dem Ansatz in [14] aber nicht zuverlässig festgelegt werden. Demnach

können Objekte, die nach eben diesem Ansatz als rechteckig klassifiziert werden, ebenso kreisförmig sein, aber auch eine ganz andere Form besitzen, welche nach der Aufgabenstellung dieser Bachelorarbeit nicht ausgegeben werden sollen. Mit der Überlegung, das mathematische Prinzip der Vektorzusammenhänge weiter zu führen, wurde im Zuge dieser Arbeit ein neues Konzept der Kreis- und Rechteckklassifizierung ausgearbeitet.

3.4.1 Linienklassifikation

Wie vorher schon beschrieben wird der Ansatz der Linienklassifikation des ODAOA-Algorithmus weitestgehend übernommen. Das Objekt wird dabei als linear klassifiziert, wenn der maximale Abstand $Dmax$ von dem Punkt g kleiner ist als der Wert *threshold_lines*. Dieser Wert ist dabei aber nicht, wie in [14] beschrieben abhängig von der Länge des linearen Objekts und somit der Strecke S zwischen dem Startpunkt p und Stopppunkt q, sondern von dem Abstands des Mittelpunktes R_m zwischen p und q ausgehend vom Sensor. Der Abstand des Mittelpunktes wird dafür wie folgt berechnet:

$$R_m = \sqrt{(\frac{p_x+q_x}{2})^2 + (\frac{p_y+q_y}{2})^2}. \tag{3.20}$$

Für den Wert *threshold_lines*, sowie die dadurch resultierende Linienklassifikation gilt dann:

wenn $R_m \leq 1m$ → $threshold\_lines = 0{,}02m$ (3.21)

wenn $R_m > 1m$ → $threshold\_lines = 0{,}01 * R_m + 0{,}01m$ (3.22)

wenn $Dmax < threshold\_lines$ → Objekt ist linear! (3.23)

3.4.2 Kreisklassifikation

Ist der Abstand $Dmax$ größer als der berechnete Wert von *threshold_lines,* so wird als nächstes geprüft, ob das Segment kreisförmig ist. Der neu entstandene Ansatz dabei besagt, dass für $P_i \in [p_j + 1, g_j - 1]$ bei einem kreisförmigen Objekt der Winkel $\alpha = \measuredangle(\overrightarrow{pq}, \overrightarrow{pg})$ kleiner ist als die Winkel $\beta_i = \measuredangle(\overrightarrow{pq}, \overrightarrow{pP_i})$. Parallel dazu gilt für $P_i \in [g_j + 1, q_j - 1]$, dass der Winkel $\delta = \measuredangle(\overrightarrow{qp}, \overrightarrow{qg})$ kleiner ist als die Winkel $\varepsilon_i = \measuredangle(\overrightarrow{qp}, \overrightarrow{qP_i})$.

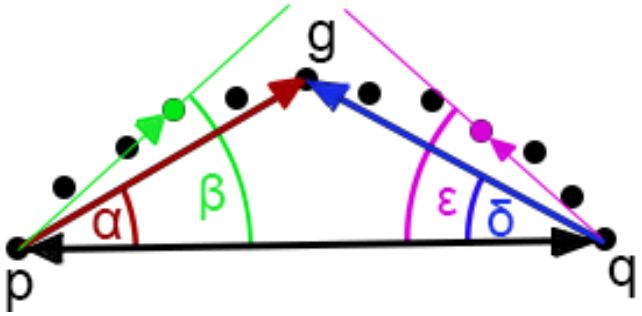

Abbildung 35: Winkel zur Kreisklassifizierung

Das heißt, dass für die Punkte P_i zwischen p und g der Winkel β zwischen den Vektoren $\overrightarrow{pP_i}$ und $\overrightarrow{pq}$ jeweils größer ist, als der Winkel α zwischen $\overrightarrow{pg}$ und $\overrightarrow{pq}$. Der gleiche Ansatz gilt ebenfalls für die Punkte P_i zwischen q und g. Hierbei ist der Winkel ε zwischen den Vektoren $\overrightarrow{qP_i}$ und $\overrightarrow{qp}$ jeweils größer als der Winkel δ zwischen $\overrightarrow{qg}$ und $\overrightarrow{qp}$.

Aufgrund von Abweichungen wird die Annahme getroffen, dass mindestens 80% der Winkel β_i und ε_i größer sind als die jeweils zu vergleichenden Winkel, damit das Segment als rund klassifiziert wird. Ebenfalls wurde festgelegt, dass die Differenz zwischen β und α sowie zwischen ε und δ mindestens 2° betragen muss, damit der Winkel als eindeutig größer eingeordnet wird. Dieser Wert hat sich als ein aus den Messwerten resultierender sinngemäßer Wert herausgestellt. Über die Vektorrechnung lassen sich Winkel zwischen Vektoren [20] über folgende Gleichung bestimmen (Beispiel für α):

$$\alpha = \cos^{-1}\left(\frac{\overrightarrow{pq} \bullet \overrightarrow{pg}}{|\overrightarrow{pq}| * |\overrightarrow{pg}|}\right) \tag{3.24}$$

Ausgerechnet ergibt sich dann als Beispiel für den Winkel α:

$$\alpha = \cos^{-1}\left(\frac{(q_x - p_x) * (g_x - p_x) + (q_y - p_y) * (g_y - p_y)}{\sqrt{(q_x - p_x)^2 + (q_y - p_y)^2} * \sqrt{(g_x - p_x)^2 + (g_y - p_y)^2}}\right) \tag{3.25}$$

Wenn die Anzahl aller Segmentpunkte mit *amount_segmentpoints*, und die Anzahl von größeren Winkeln mit *amount_greater_angle* beschrieben wird, so gilt dann für die Kreisklassifizierung:

wenn $amout\_greater\_angle > 0{,}8 * (amount\_segmentpoints - 3)$ (3.26)
→ Segment ist Kreisförmig

Die Anzahl der Segmentpunkte wird mit drei subtrahiert, da die Anzahl auch der maximal möglichen Anzahl von *amount_greater_angle* entsprechen soll. Die drei Punkte p,q und g werden somit nicht mitgezählt.

3.4.3 Rechteckklassifikation

Ist der Abstand $Dmax$ größer als der berechnete Wert von *threshold_lines* und das Segment wurde nicht als lineares Objekt beziehungsweise kreisförmiges Segment klassifiziert, so wird das jeweilige Segment nach Merkmalen eines Rechtecks untersucht. Dafür wird zum einen geprüft, ob der Winkel $\alpha = \measuredangle(\overrightarrow{gp}, \overrightarrow{gq})$ zwischen 80° bis 100° liegt. Der Bereich für α wurde ebenfalls aufgrund von Messabweichungen festgelegt.

Außerdem wird noch einmal dasselbe Prinzip wie bei der Linienklassifikation angewendet. Hierbei werden aber die Abstände für $P_i \in [p_j + 1, g_j - 1]$ zu der Geraden, welche durch die Punkte g und p gezogen wird, sowie die Abstände für $P_i \in [g_j + 1, q_j - 1]$ zu der Geraden durch g und q berechnet. Durch j wird dabei wieder das jeweilige Segment beschrieben.

Das heißt, dass im Prinzip jede „Kante“ des Rechtecks als Linie klassifiziert wird. Der Grenzwert *threshold_rectangle* ist dabei wie bei der Linienklassifikation von dem Abstand des Mittelpunktes R_m abhängig. Somit gilt für die Grenzwertbestimmung, sowie für die Bedingungen der Rechteckklassifizierung folgendes:

wenn $R_m \leq 1m \quad \rightarrow \quad threshold\_rectangle = 0{,}02m$ (3.27)

wenn $R_m > 1m \quad \rightarrow \quad threshold\_rectangle = 0{,}02 * R_m$ (3.28)

wenn $Dmax_{\overrightarrow{gp}} < threshold\_rectangle$ (3.29)
&& $Dmax_{\overrightarrow{gq}} < threshold\_rectangle$ && $80° < \alpha < 100°$
→ Segment ist Rechteckig!

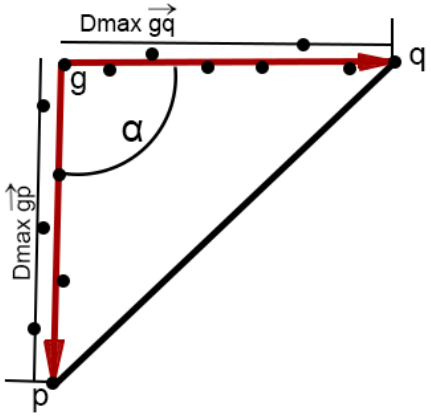

Abbildung 36: Winkel und Dmax zur Rechteckklassifizierung

Die Werte α, $D_{i\overrightarrow{gp}}$ und $D_{i\overrightarrow{gq}}$ errechnen sich dabei wie nach dem in Absatz 3.4.2 für den Winkel und im Kapitel 3.4 für die Abstände ausführlich erläuterten Schema.

Nach dem Schritt der Rechteckklassifizierung sind die Segmente entweder als lineares Objekt, rundes oder rechteckiges Segment, oder in Segmente, die nach dem Klassifizierungsverfahren nicht einzuordnen sind, aufgeteilt. Die Segmente, welche nicht zugewiesen werden können, werden dementsprechend auch nicht in der folgenden Positionsbestimmung mitberücksichtigt, da nur runde, rechteckige oder lineare Objekte ausgegeben werden sollen. Ebenfalls werden runde und rechteckige Segmente noch nicht als Objekte betrachtet, da diese wie in 3.4.4 erklärt, erst noch als Raum oder Objekt klassifiziert werden.

3.4.4 Klassifizierung als Raum oder Objekt

Der Klassifizierungsschritt, welcher unterscheidet, ob die rechteckigen und runden Segmente einen Raum oder ein Objekt darstellen, setzt voraus, dass die Positionierung der Segmente, beziehungsweise die der Segmentmittelpunkte bekannt ist. Da dieser Schritt aber als Teil der Klassifizierung einzuordnen ist, wird dieser schon in diesem Abschnitt erklärt. Der Verarbeitungsschritt der Positions-, Ausrichtung- und Größenberechnung wird ausführlich im darauffolgenden Kapitel 4.5 erklärt. Das Prinzip der Objekt- beziehungsweise Raumklassifizierung kann sowohl auf kreisförmige, als auch auf rechteckige Segmente angewendet werden. Hierbei wird der Abstand ausgehend von dem Sensor zum Punkt g und zum Mittelpunkt m des jeweiligen Objekts miteinander verglichen.

Nachdem die Position von m, wie in Kapitel 3.5 erläutert ermittelt wurde, kann darüber entschieden werden, ob der Sensor wie in Abbildung 37 links ein Objekt misst, oder parallel dazu wie in Abbildung 37 rechts einen Raum. Ein Raum könnte somit zum Beispiel eine Ecke in einem Zimmer sein. Würde der Schritt der Raum-/Objektklassifizierung nicht durchgeführt werden, würde das Programm die Ecke in einem Raum aufgrund der Form als ein rechteckiges Objekt wahrnehmen und in der Ausgabe ein Objekt darstellen, welches nicht vorhanden ist.

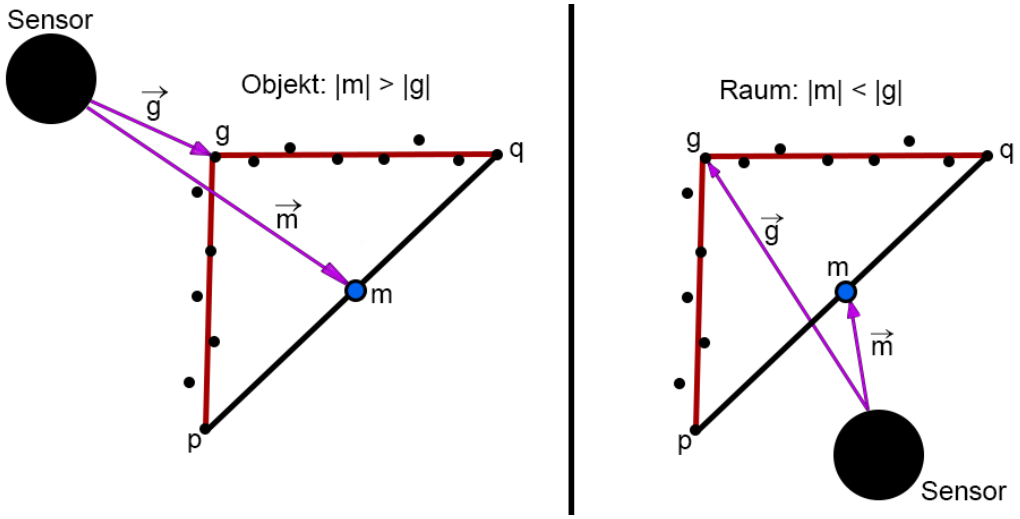

Abbildung 37: links: Sensor misst ein Objekt, rechts: Sensor misst einen Raum

Für die Klassifizierung des Segments j als Raum, beziehungsweise als Objekt gilt dabei:

wenn $|\overrightarrow{m_j}| > |\overrightarrow{g_j}|$ → Das Segment ist ein Objekt (3.30)

wenn $|\overrightarrow{m_j}| < |\overrightarrow{g_j}|$ → Das Segment ist ein Raum (3.31)

Die einzelnen Werte berechnen sich dabei aus den Beträgen der jeweiligen Vektoren:

$$|\overrightarrow{m_j}| = \sqrt{{m_{jx}}^2 + {m_{jy}}^2} \quad (3.32)$$

$$|\overrightarrow{g_j}| = \sqrt{{g_{jx}}^2 + {g_{jy}}^2} \quad (3.33)$$

Das heißt, dass das Segment als Objekt klassifiziert wird, wenn der Abstand ausgehend vom Sensor zum *max_distance_point* g kleiner ist als der Abstand zum Mittelpunkt m des Segments. Ist der Abstand zum Mittelpunkt geringer, ist das Segment ein Raum. Dieser Ansatz wurde jeweils auf Segmente, die als Rechteck, wie aber auch auf jene, welche als kreisförmig klassifiziert wurden, angewendet.

Nach der gesamten Klassifizierung erhält man somit eine Liste aller Objekte, die laut der Aufgabenstellung ausgegeben werden sollen und dementsprechend auch klassifiziert worden sind. Alle in Kapitel 4.5 errechneten Werte zur Beschreibung der Position, Ausrichtung und Größe der Objekte werden dann auf der topic *objects* veröffentlicht und können zur Visualisierung von dem Knoten *visualization_objects* abonniert werden.

3.5 Position, Ausrichtung und Maße der Objekte

Damit die jeweiligen Objekte in der finalen Visualisierung richtig dargestellt werden können, wird im Folgenden die Berechnung der Position, Ausrichtung und Größe der Objekte erläutert. Ebenfalls werden jene Parameter zur Klassifizierung als Raum oder Objekt, wie in 3.4.4 erläutert verwendet.

3.5.1 Objekte mit Linienform

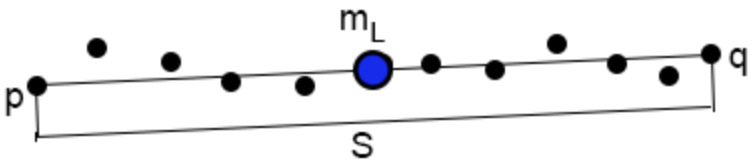

Abbildung 38: Positions- und Längenbeschreibung von linearen Objekten

Zur Bestimmung von Position, Ausrichtung und Länge von Objekten mit Geradenform werden ausschließlich die Start- und Stopppunkte des jeweiligen Objektes benötigt. Die Länge S des Objekts ergibt sich aus dem Betrag des Vektors $\overrightarrow{pq}$. Die Koordinaten des Linienmittelpunkts m_L lassen sich somit auch einfach über den Mittelpunkt des $\overrightarrow{pq}$ Vektors berechnen:

$$S = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2} \tag{3.34}$$

$$m_{Lx} = \frac{p_x + q_x}{2} \tag{3.35}$$

$$m_{Ly} = \frac{p_y + q_y}{2} \tag{3.36}$$

Die errechneten Werte zur Beschreibung der linearen Objekte werden dann in die Listen *lines_start_x* (p_x), *lines_start_y* (p_y), *lines_stop_x* (q_x), *lines_stop_y* (q_y), *lines_middle_point_x* (m_{Lx}) und *lines_middle_point_y* (m_{Ly}) gespeichert. Je nach weiterer Anwendung wurde sich ebenfalls dafür entschieden, die Koordinaten des Linienpunktes mit dem geringsten Abstand zum Sensor mit auszugeben. Diese werden in die Listen *lines_closest_point_x* und *lines_closest_point_y* eingespeichert. Über die Koordinaten von Start und Stopppunkt kann wie in 3.6 erläutert, die Ausrichtung des Objekts berechnet werden.

3.5.2 Objekte mit Rechteckform

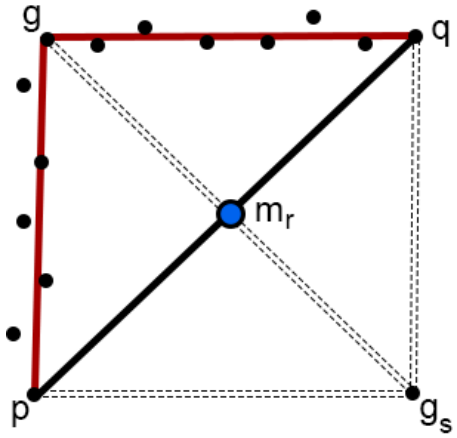

Abbildung 39: Positions- und Formbeschreibung der als Rechteck klassifizierten Objekte

Um die Größe, Position und Ausrichtung des Rechtecks beschreiben zu können, werden zum einen die vier Eckpunkte, aber auch zusätzlich der Mittelpunkt des Rechtecks bestimmt und ausgegeben. Wie in der Abbildung 39 gut zu erkennen, werden die Punkte p, g und q

dabei auch als Eckpunkte des Rechtecks behandelt. Die Koordinaten der jeweiligen Punkte werden dafür in die Listen *rectangle_corner_start_x* (p_x), *rectangle_corner_start_y* (p_y), *rectangle_corner_max_distance_x* (g_x), *rectangle_corner_max_distance_y* (g_y), *rectangle_corner_stop_x* (q_x) *und rectangle_corner_stop_y* (q_y), eingespeichert.

Der Mittelpunkt m_r wird wie bei linearen Objekten auf die Mitte des $\overrightarrow{pq}$ Vektors gelegt und wie in (3.35) und (3.36) berechnet.

Für den Punkt *g-spiegel* oder kurz g_s wird angenommen, dass sich der Mittelpunkt m ebenfalls genau auf der Mitte des Vektors $\overrightarrow{gg_s}$ befindet. Stellt man dann für $\overrightarrow{gg_s}$ nach dem Prinzip von (3.35) und (3.36) die Formeln für die Mittelpunktkoordinaten auf und setzt beide Formel für m_{rx} und m_{ry} gleich, so erhält man nach Umstellen die Koordinaten des Punktes g_s. Dafür ergibt sich dann:

$$g_{sx} = p_x + q_x - g_x \tag{3.37}$$

$$g_{sy} = p_y + q_y - g_y \tag{3.38}$$

Die Koordinaten der neu berechneten Punkte werden dann in die Listen *rectangle_corner_max_distance_mirror_x* (g_{sx}), *rectangle_corner_max_distance_mirror_y* (g_{sy}), sowie *rectangle_center_x* (m_{rx}) und *rectangle_center_y* (m_{ry}) eingespeichert.

Über die gespeicherten Punkte können dann nach Belieben die Ausrichtung sowie Kantenlängen, wie in 3.6 beschrieben berechnet werden.

3.5.3 Objekte mit Kreisform

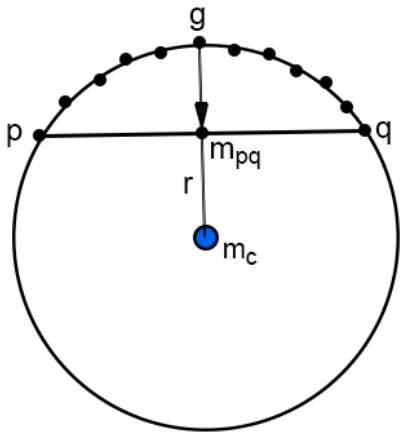

Abbildung 40: Position- und Größenbeschreibung der als Kreis klassifizierten Objekte

Um die Position und Größe eines Kreises sinngemäß zu beschreiben, werden die Koordinaten des Kreismittelpunktes sowie der Kreisradius benötigt. Die Schwierigkeit bei der Bestimmung des Kreismittelpunktes m_c entsteht dadurch, dass der Sensor nicht die Hälfte des Kreise, bzw. nicht 180° des Kreisumfanges erkennt. Somit steht nur ein kleiner Abschnitt des Kreisbogens zur Verfügung. Zum bestimmen des Kreismittelpunktes wurde deshalb angenommen, dass der Mittelpunkt sich durch das Verlängern des Vektors $\overrightarrow{gm_{pq}}$ um einen gewissen Faktor c ergibt. Nach den gleichen mathematischen Prinzipien wie in Abschnitt 3.5.2 beschrieben, ergeben sich dann für die Mittelpunktkoordinaten folgende Gleichungen:

$$m_{cx} = g_x + c * \left(\frac{p_x + q_x}{2} - g_x \right) \tag{3.39}$$

$$m_{cy} = g_y + c * \left(\frac{p_y + q_y}{2} - g_y \right) \tag{3.40}$$

Über den Betrag des Vektors $\overrightarrow{gm_c}$ kann dann anschließend der Radius ermittelt werden.

$$r = \sqrt{(m_{cx} - g_x)^2 + (m_{cy} - g_y)^2}. \tag{3.41}$$

Durch überprüfen der Ausgabe von Messungen mit runden Objekten, bei denen der Radius bekannt war, konnte ein sinnvoller Wert für den Faktor c ermittelt werden. Demnach wurde sich für den Faktor $c = 1{,}9$ entschieden.

Die Koordinaten des Kreismittelpunktes sowie der Radius werden daraufhin in den Listen *circle_center_x* (m_{cx}), *circle_center_y* (m_{cy}) und *circle_radius* $((r)$ eingespeichert.

3.6 Objekt Visualisierung

Um die durch die Segmentierung und Klassifizierung detektierten Objekte auch visuell anschaulich darzustellen, wurde der ROS-Datentyp *Marker* verwendet. Dieser bietet die Möglichkeit, verschiedene Formen im ROS-Visualisierungsprogramm *rviz* auszugeben. Ebenfalls können Parameter wie Maße, Orientierung, Position und Farbe der dargestellten Formen festgelegt werden.

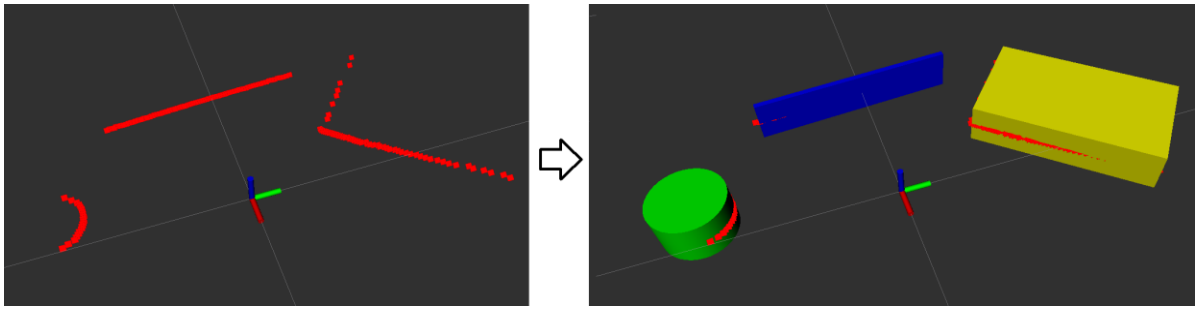

Abbildung 41: Links: Darstellung der Messpunkte in rviz, rechts: Darstellung der erkannten Objekte mittels Marker im rviz

In Abbildung 41 ist eine Beispielmessung von drei verschiedenen geformten Objekten dargestellt. Links in der Abbildung sind die Messpunkte des RPLIDARs nach der Filterung durch den Medianfilter zu sehen. Der Sensor befindet sich dabei in dem Ursprungspunkt der auf beiden Bildern mittig abgebildeten Koordinatenachsen. Rechts sind die nach der Segmentierung und Klassifizierung durch das Paket *visualization_objects* visualisierten Marker im *rviz* dargestellt. Wie in der Abbildung zu erkennen, werden runde Objekte in Grün, lineare Objekte in Blau und rechteckige Objekte in Gelb ausgegeben.

Für die Form der jeweiligen Marker wurde als Ausgabe für runde Objekte die Form eines Zylinders gewählt. Rechteckige sowie lineare Objekte werden beide als Quader visualisiert. Die Breite b des linearen Objektes ist dabei aber immer $b = 1\ cm$, also ein sehr schmales Quader. Um trotzdem eine Verwechselung zu einem sehr schmalen rechteckigen Objekt auszuschließen, werden die Objekttypen außerdem in zwei verschiedenen Farben dargestellt.

Die Koordinaten der Marker Position wird für runde Objekte aus den abonnierten Werten von *circles_center_x* und *circles_center_y* übernommen. Bei linearen Objekte werden die Listen *lines_middle_point_x* und *lines_middle_point_y,* sowie bei Rechtecken die Werte aus *rectangles_center_x* und *rectangles_center_y* verwendet. Dabei ist zu beachten, dass die positive x- und y-Richtung des für die Sensorwerte festgelegten Koordinatensystems genau entgegengesetzt des *rviz* internen Koordinatensystems verläuft. Um die Objekte somit im *rviz* richtig zu positionieren, werden die jeweiligen Koordinaten der Marker negativ angenommen.

Die Dargestellte Höhe der Marker wurde an die, wie in Kapitel 3.1 erläuterte Messhöhe des Sensors von 75 Millimetern angepasst. Dabei ist zu beachten, dass die gemessenen Objekte somit mindesten 75 Millimeter groß sind. Aufgrund der Einschränkung durch die 2D Messung kann keine genaue Angabe zu dem Maß der Objekte in z-Richtung getroffen werden.

Für die Objektmaße in x-, beziehungsweise y-Richtung werden die wie in Kapitel 3.5 erläuterten errechneten Werte der jeweiligen Objektformen verwendet. Für kreisförmige Objekte ist dies somit der errechnete Radius r. Die Länge der linearen Objekte ist gleich dem Betrag des Vektors $\overrightarrow{pq}$. Für die rechteckigen Objekte werden dafür die Beträge der Vektoren $\overrightarrow{gp}$ und $\overrightarrow{gq}$ und somit der Kantenlänge des jeweiligen Objekts dargestellt.

Um die Orientierung der linearen und rechteckigen Objekte mittels der Marker im $rviz$ richtig darzustellen, werden die jeweiligen Rotationswinkel φ_z um die z-Achse der Objekte berechnet.

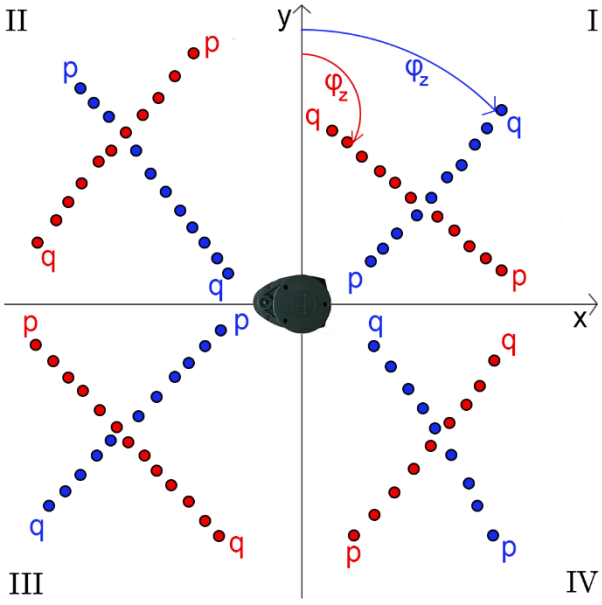

Abbildung 42: Rotationswinkel um die z-Achse von linearen Objekten

Wie in der Abbildung 42 dargestellt, muss bei der Berechnung der Rotationswinkel die jeweilige relative Position des Objekts zum Sensor beachtet werden. Dies wird für den Fall von linearen Objekten im Folgenden erläutert. Dasselbe Prinzip wird ebenfalls für die rechteckigen Objekte angewendet. Für die runden Objekte muss keine Ausrichtung berechnet werden, da der Zylinderradius ausgehend von Mittelpunkt in alle Richtung gleich ist.

Um die Ausrichtung korrekt darzustellen, muss als erstes beachtet werden, in welchem Quadranten sich das Objekt befindet. Zur Bestimmung des Quadranten wurden im Falle des

linearen Objekts die Koordinaten des Geradenmittelpunktes m_L gewählt. Ist somit sowohl die x-Koordinate, als auch die y-Koordinate von m_L positiv, wird das Objekt in dem ersten Quadranten I positioniert. Nun muss unterschieden werden, wie das Objekt relativ zum Sensor positioniert ist, da dadurch in der Winkelberechnung verschiedene Vektoren beachtet werden müssen. Die beiden Möglichkeiten sind in Abbildung 42 als blaues sowie als rotes Objekt dargestellt. Somit muss für das blaue Objekt im ersten Quadranten der Vektor $\overrightarrow{pq}$ und für das rote Objekt der Vektor $\overrightarrow{qp}$ verwendet werden. Um zu unterscheiden, wie das Objekt in Relation zum Sensor positioniert ist, werden im Beispiel von I die x-Koordinaten der Startpunkte zu den x-Koordinaten der Stopppunkte miteinander verglichen. Somit gilt, dass wenn $p_x < q_x$ ist, das Objekt so positioniert wird, wie die blaue Darstellung in der Abbildung. Parallel dazu gilt für das rote Objekt $p_x > q_x$. Demnach kann im Programm unterschieden werden, welcher Vektor für die Berechnung verwendet werden muss. Für die Rotationswinkel im Falle das $p_x < q_x$ ergibt sich dann:

$$\varphi_z = \frac{q_y - p_y}{\sqrt{(q_x - p_x)^2 + (q_y - p_y)^2}}. \tag{3.42}$$

Das oben angewendete Schema muss für die korrekte Winkelbestimmung für jeden der Quadranten ausgelegt werden.

3.7 Implementierung des Workspace im Raspberry Pi

Die Schwierigkeit, die bei der Implementierung des Workspace im Raspberry Pi entstehen kann, liegt primär an der geringen Größe von Arbeitsspeicher. Läuft der Arbeitsspeicher voll, kann es zu Systemabstürzen des Raspberry Pi kommen. Um dies zu vermeiden, kann vorübergehend ein sogenannter SWAP-Speicher genutzt werden. Ist der Arbeitsspeicher voll, so kann der Kernel die im RAM gespeicherten Daten, welche momentan nicht benötigt werden, zwischenzeitlich auf den Massenspeicher der SD-Karte verschieben, um somit Speicherplatz für weitere Prozesse zu schaffen. Werden die Daten wieder benötigt, so werden diese wieder in den RAM geladen.

Daher wurde geprüft, wieviel Arbeitsspeicher der gesamte Prozess der Sensordatenauslesung, Median-Filterung, Segmentierung, Klassifizierung sowie Visualisierung in Anspruch nimmt. Laut Terminalausgabe mit dem Befehl „free –h“ verfügt der Raspberry Pi über einen gesamten verfügbaren Arbeitsspeicher von 872 Megabyte. Werden alle Knoten inklusive der Visualisierung ausgeführt, verfügt der Raspberry Pi noch über 32 Megabyte freien Arbeitsspeicher. Ohne Visualisierung, somit nur die reine Berechnung und Datenverarbeitung bis hin zur Klassifizierung, verfügt dir Kleinstrechner noch über 104 Megabyte RAM. Da bei der Programmausführung inklusive Visualisierung der freie Arbeitsspeicher nur noch sehr begrenzt ist, wurde zur Vermeidung von Systemabstürzen ein SWAP-Speicher von einem Gigabyte eingerichtet.

Außerdem ist bei der ROS-Einbindung des *laser_filter* Pakets über den Befehl *catkin_make* der Arbeitsspeicher des Raspberry Pi vollgelaufen. Somit wurde für die Einbindung der maximale verfügbare SWAP-Speicher von fünf Gigabyte zur Verfügung gestellt. Dieser wird aber nach der einmaligen Einbindung des Pakets nicht mehr benötigt. [36]

4 Evaluierung

4.1 Versuchsreihe

Um die gegebene Funktionalität des in Kapitel 3 umgesetzten Segmentierungs- und Klassifizierungsverfahrens zu bewerten, wurden als Versuchsreihe verschieden geformte Objekte im statischen Zustand geprüft.

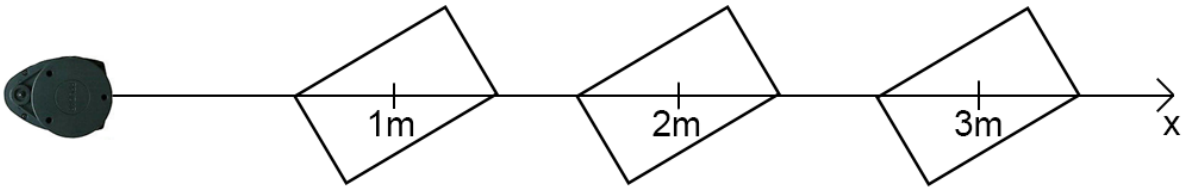

Abbildung 43: Versuchsaufbau mit rechteckigem Objekt

Dafür wird jeweils ein rechteckiges, ein rundes und ein lineares Objekt in den Abständen ein Meter, zwei Meter und drei Meter zum Sensor platziert. Ebenfalls soll das Objekt so genau wie möglich auf der festgelegten x-Achse der Messung positioniert werden. Pro Objekt und Abstand werden dabei 20 Messungen aufgenommen und geprüft, ob der Algorithmus das Objekt erkennt, dieses richtig im Raum platziert (Abstand und Winkel zur x-Achse) und ebenfalls, ob die Objekt-maße (Breite, Länge, Durchmesser) richtig berechnet werden.

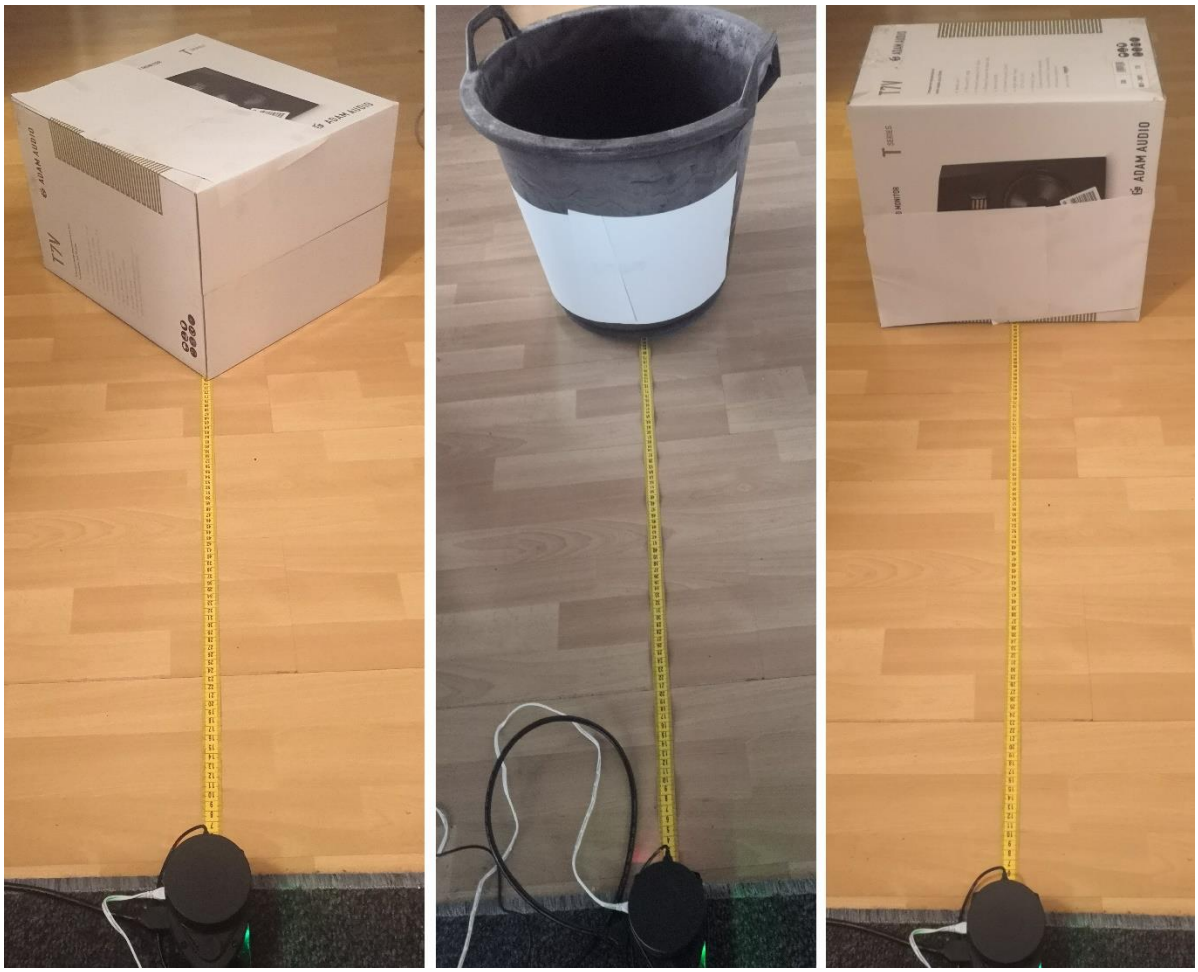

Abbildung 44: Versuchsaufbau mit Karton (Rechteck), Eimer (rund) und Karton (linear) mit einem Meter Abstand zum Sensor

Wie in Abbildung 44 dargestellt, wurde als rechteckiges Objekt ein weißer Karton verwendet. Die zu messenden Kantenlängen des Kartons betragen 46 cm und 39,5 cm. Die längere Kante wurde ebenfalls für die Versuchsreihe des linearen Objekts verwendet, indem der Karton zum Sensor so positioniert wurde, dass nur eine der Quaderflächen vom Sensor erfasst wird. Als kreisförmiges Objekt wurde ein schwarzer Eimer verwendet. Damit die Oberfläche des Eimers der des Kartons ähnelt, wurde auf dem Eimer weißes Papier im Messbereich des Sensors befestigt. Da der Durchmesser des Eimers nicht konstant ist, wurde der Durchmesser bei der Maßbestimmung mit dem Durchmesser des Eimers auf der Messhöhe des Sensors festgelegt. Dieser beträgt auf der Messhöhe 37 cm.

Für die Messungen wurde jeweils der Mittelpunkt der verschiedenen Objekte auf den zu messenden Abständen positioniert. Dadurch soll neben der richtigen Klassifizierung auch die gesamte Mittelpunktsberechnung geprüft werden.

4.2 Erwartetes Ergebnis und Fehlerquellen

Neben den offensichtlichsten Fehlerquellen, wie den Messtoleranzen des Sensors sowie den Messfehlern im Versuchsaufbau wird erwartet, dass sich das Ergebnis der Objekterkennung und parallel dazu auch die berechneten Werte mit zunehmenden Messabständen verschlechtert. Je größer die Entfernung zwischen Sensor und dem zu messenden Objekt ist, desto weniger Messpunkte stehen zur Objekterkennung und Klassifizierung zur Verfügung. Dies resultiert aus den gleichbleibenden Winkelabständen zwischen den einzelnen Messpunkten. Durch eine geringere Anzahl von Messpunkten, die zur Klassifizierung zur Verfügung stehen, haben ebenfalls die Messtoleranzen des Sensors eine größere Auswirkung auf die Berechnung. Es wird erwartet, dass sich dieser Effekt besonders bei der Kreisklassifizierung bemerkbar macht, da aufgrund der geringeren Anzahl von Objektmesspunkten sowie der durch die Messtoleranz entstehenden Winkeländerungen, nicht mehr 80% der Winkel größer als die zu vergleichenden Winkel sind (detailliert in 3.4.2 erklärt).

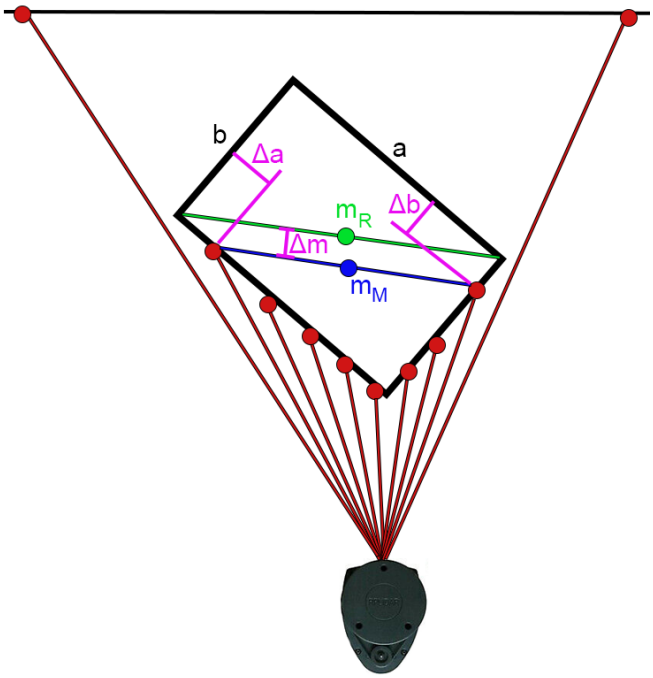

Abbildung 45: Fehlerquellen der Abstands- sowie Maßberechnung

Außerdem werden zu kleine Objekte bei großen Entfernungen, wegen der in der Segmentierung festgelegten Mindestanzahl von Messpunkten für ein Objekt nicht mehr als Objekte klassifiziert. Die Objektgröße für diesen Versuch wurde aber bewusst so gewählt, dass auch bei einer Entfernung von drei Metern immer noch genug Messpunkte von den Objekten reflektiert werden.

Außerdem wird erwartet, dass es, wie in Abbildung 45 im Beispiel eines rechteckigen Objektes dargestellt, aufgrund der Winkelschritte zwischen den Messpunkten zu fehlerbehafteten Abstands- sowie Maßberechnungen kommt. Somit werden nicht die gesamten Kantenlängen a und b vom Sensor erfasst. Daraus resultiert, dass die Eckpunkte des erkannten Rechtecks fehlerbehaftet positioniert werden und somit die berechneten Werte um die Fehler Δa und Δb abweichen. Ebenfalls entsteht dadurch eine fehlerhafte Abstandsberechnung um Δm zwischen dem realen Mittelpunkt m_R und dem gemessenen Mittelpunkt m_M. Es wird ebenfalls erwartet, dass dieser Fehler bei steigenden Abständen zunimmt. Die in Abbildung 45 dargestellte Fehlerquelle wirkt sich ebenfalls auf die Abstands- und Radius- bzw. Längenberechnung bei kreisförmigen und linearen Objekten aus.

4.3 Auswertung

Zur Auswertung der gemessenen und berechneten Ergebnisse wurde aus den 20 aufgenommen Werten der Mittelwert ($\bar{x}$) und die Standartabweichung (σ_x) gebildet, wobei x als Variable für die jeweils zu messende Größe zu verstehen ist. Im Anhang dieser Arbeit findet sich ein ausführliches Versuchsprotokoll, in dem die einzelnen Messwerte sowie die daraus resultierenden Normalverteilungsdiagramme dargestellt sind.

4.3.1 Kreisförmiges Objekt

Tabelle 3: Versuchsergebnisse kreisförmiges Objekt

Objekt	Rund Eimer						
Durchmesser	37cm						
Oberfläche	Papier						
Farbe	Weiß						
Realer Abstand R	Erkannt zu	$\bar{R}$ [cm]	σ_R [cm]	$\bar{\varphi}$ [°]	σ_φ [°]	$\bar{D}$ [cm]	σ_D[cm]
100cm	100%	100,5	1	-0,5	0,9	37,8	2,2
200cm	100%	199,5	1,9	-0,2	0,9	38,3	3,8
300cm	75%	292,4	1,8	-0,5	1	30,8	5,9

Bei der Kreismessung wird das Objekt bei einem Abstand von 100cm, aber auch noch bei 200cm zu 100% (von 20 Messungen) als Kreis klassifiziert. Wie in Kapitel 4.2 beschrieben nimmt dieser Wert erwartungsgemäß bei größeren Abständen ab. Bei 300cm Abstand wird das Objekt nur noch in 75% der Messungen als Kreis erkannt.

Die Differenz $\Delta R = \bar{R} - R$ fällt ebenfalls bei den Messungen im Abstand von 100cm und 200cm mit $\Delta R = 0{,}5\ cm$ sehr gering aus. Bei der Messung von 300cm wächst diese etwas an und liegt bei $\Delta R = 7{,}6\ cm$. Dies bestätigt den in Abbildung 45 dargestellten Fehler, wodurch nicht das gesamte Objekt durch den Sensor erkannt wird und der Abstand somit zu gering berechnet wird. Ebenfalls spiegelt sich dies in der Durchmesserberechnung wieder, wobei

der Durchmesser bei 300cm mit $\Delta D = -6{,}2$ deutlich kleiner ausfällt. Wie die Standardabweichung zeigt, sind die gemessenen Abstände bei allen realen Abständen mit einem maximalen Wert von $\sigma_R = 1{,}9\ cm$ bei der 200 cm Messung sehr konstant. Bei der Durchmesserberechnung nimmt die Standardabweichung mit größer werdenden Abständen zu.

Bei der Winkelbestimmung beinhaltet vermutlich die reale Positionierung des Objekts zur x-Achse die größere Fehlerquelle als die Messung selbst, wobei das Objekt mit einer maximalen Winkeldifferenz von $\Delta\varphi = -0{,}5°$ mit einem sehr geringen Abstand zur x-Achse platziert wurde. Ebenfalls ist die Standardabweichung φ_σ bei jeglichen Entfernungen mit maximal $\sigma_\varphi = 1°$ sehr konstant, was zeigt, dass der Mittelpunkt der runden Objekte mit einem geringen Fehler im Raum platziert wird.

4.3.2 Rechteckiges Objekt

Tabelle 4: Versuchsergebnisse rechteckiges Objekt

Objekt	Rechteck Karton								
Länge a	46cm								
Breite b	39,5cm								
Oberfläche	Papier								
Farbe	Weiß								
Realer Abstand R	Erkannt zu	$\bar{R}$ [cm]	σ_R [cm]	$\bar{\varphi}$ [°]	σ_φ [°]	$\bar{a}$ [cm]	σ_a [cm]	$\bar{b}$ [cm]	σ_b [cm]
100cm	100%	98,7	0,4	0,4	0,4	43,5	0,9	37,6	1,1
200cm	100%	196,8	1,2	1,9	0,4	41,7	2,8	35,0	1,8
300cm	85%	296,8	2,2	-0,6	0,3	41,7	5,0	34,6	2,2

Wie bei dem kreisförmigen Objekt wurde das rechteckige zu 100% bei einem Abstand von 100cm und 200cm erkannt. Bei einem Abstand von 300cm wurde das Objekt in 85% der Messungen als Rechteck klassifiziert. Dazu muss aber hinzugefügt werden, dass in den restlichen 15% das Objekt als Kreis klassifiziert wurde und somit nicht weiter geprüft wurde, ob ggf. die Kriterien einer Rechteckklassifizierung ebenfalls zutreffend waren. Die Abstände wurden bei allen Messungen etwas zu gering berechnet. Bei der 300cm Messung liegt die Differenz zum realen Wert bei $\Delta R = -3{,}2\ cm$, was darauf schließen lässt, dass die Abstandsberechnung der rechteckigen Objekte bessere Ergebnisse liefert als die der Kreisklassifikation. Was aber auffällt ist, dass die Standardabweichung der berechneten Abstandswerte mit größeren Abständen ansteigt.

Der Mittelwert der berechneten Winkel zur x-Achse ist wiederum schwer zu bewerten, da der Objektmittelpunkt vermutlich nicht genau auf der x-Achse platziert worden ist. Die geringe Standardabweichungen aller Winkel von maximal σ_φ = 0,4° zeigt aber auch, dass das Objekt konstant ohne große Abweichungen im Raum platziert werden kann.

Die Kantenlängen wurden bei allen Messabständen im Durchschnitt kleiner wahrgenommen, als es der Realität entspricht. Dies resultiert vermutlich aus dem in Abbildung 45 gezeigten Fehler. Insgesamt bleiben die Kantenlängen aber auch mit steigenden Abständen fast konstant. Somit sind diese bei der 200cm und der 300cm nahezu identisch. Wie erwartet steigen

aber auch die Standardabweichungen der gemessenen Kantenlänge mit größer werdenden Abstand zum Sensor mit an.

4.3.3 Lineares Objekt

Tabelle 5: Versuchsergebnisse lineares Objekt

Objekt	Linie Karton						
Länge	46cm						
Oberfläche	Papier						
Farbe	Weiß						
Realer Abstand	Erkannt zu	$\bar{R}$ [cm]	σ_r [cm]	$\bar{\varphi}$ [°]	σ_φ [°]	$\bar{a}$ [cm]	σ_a [cm]
100cm	100%	100,0	0,2	-0,8	0,3	45,0	1,3
200cm	100%	199,8	0,2	-1,0	0,1	42,0	0,0
300cm	100%	299,6	0,3	-1,0	0,1	41,7	1,2

Die Versuchsergebnisse der linearen Objekte weichen von den drei verschiedenen Objekttypen am wenigsten von den Realwerten ab. Außerdem wurde in allen Messungen in den drei zu messenden Abständen das Objekt als lineares Objekt erkannt. Ebenfalls ist der durchschnittlich gemessene Abstandswert mit einer maximalen Differenz von $\Delta R = -0{,}4\ mm$ bei der 300cm Messung sehr gering. Auch die geringen Werte der Standardabweichungen von σ_r zeigen, dass der Abstand konstant realitätsnah berechnet wurde. Dies resultiert aber auch daraus, dass das Objekt senkrecht zur x-Achse platziert wurde und somit durch die Berechnung bzw. die Objektmaße keine Abweichungen in der Mittelpunktberechnung entstehen.

Ähnlich wie bei den anderen Objekten konnte der Winkel mit einer geringen Standartabweichung σ_φ berechnet werden. Dies lässt wiederum darauf schließen, dass das Objekt im Versuchsaufbau geringfügig neben der x-Achse platziert wurde, die Berechnungen aber ein gutes Ergebnis liefern.

Ausschließlich bei der Messung der Länge kommt es bei größer werdenden Abständen zu größeren Fehlern in der Berechnung. Dies lässt sich vermutlich aber wieder auf den in Abbildung 45 dargestellten Fehler beziehen, da die Standartabweichung der berechneten Längen weiterhin gering bleibt.

5 Zusammenfassung und Ausblick

Die Evaluierung hat gezeigt, dass runde, rechteckige und lineare Objekte durch eine geeignete Segmentierungsmethode aus den Messergebnissen des RPLIDARS erkannt und auch positioniert werden können. Aus den recherchierten Segmentierungsmethoden konnte ein für die Aufgabenstellung passendes Verfahren ausgewählt und sogar noch weiter ausgearbeitet werden. Es hat sich aber auch gezeigt, dass mit dem verwendeten RPLIDAR A1 eine Klassifizierung von kleineren Objekten ab einer gewissen Distanz schwierig wird. Dafür könnte ein Laserscanner mit einer höheren Auflösung Abhilfe schaffen.

Der konstruierte Aufbau erfüllt die gegebenen Anforderungen einer robusten Halterung für RPLIDAR sowie Raspberry Pi. Gegebenenfalls könnte die Halterung so konstruiert werden, dass durch das Einbringen von Taschen im CAD-Programm weniger Material verbraucht wird und somit das Gewicht und die Druckzeit reduziert wird.

Dennoch ist eine Klassifizierung von gänzlich unbekannten Objekten unter Verwendung eines einzelnen Laserscanners nur sehr schwer umzusetzen. Demnach sind die auf der 2D Punktewolke zu erkennenden Konturen zwar von geometrischen Primitiven eindeutig, anders geformte Objekte können aber aus einer zweidimensionalen Messung kaum erkannt werden. Mit Ausblick auf eine Gefahrenerkennung im Straßenverkehr bieten sich somit verschiedene Möglichkeiten an.

Eine Möglichkeit, auch straßenverkehrsrelevante Hindernisse wie Fahrräder und Fußgänger zu erkennen, wäre die Sensorfusion von mehreren RPLIDAR-Sensoren, welche übereinander angeordnet werden. Dadurch kann von den gemessenen Objekten eine dreidimensionale Kontur erstellt werden, welche eine Klassifizierung vereinfachen würde.

Eine weitere Möglichkeit wäre der Einsatz von unterschiedlichen Umfeld erfassenden Sensortypen. Somit könnten mittels einer Sensorfusion die Vorteile der jeweiligen Sensoren genutzt werden. Durch die Verwendung einer Kamera könnte die Klassifizierung der Objekte eindeutiger bestimmt werden. Allerdings sind die durch eine Kamera auszuwertenden Daten deutlich größer als die eines 2D Laserscanners. Außerdem können Kamerasysteme die Abstände der jeweiligen Objekte nicht so genau bestimmen, wie beispielsweise der RPLIDAR. Demnach kann die in dieser Arbeit verwendete Segmentierungsmethode auf die Messergebnisse des Laserscanners angewendet werden, um potenzielle Objekte unabhängig von der Form beziehungsweise Klassifizierung zu lokalisieren. Von den Daten der Kamera müssten dann nur noch die Bereiche überprüft werden, wo sich potentielle Objekte befinden könnten. Durch die Sensorfusion könnten somit die Objekte genau im Raum positioniert werden und auch komplexer geformte Hindernisse klassifiziert werden.

Auch der in dieser Bachelorarbeit ausgearbeitete Algorithmus bietet noch Raum zur Verbesserung. Ein möglicher Ansatz wäre, den in dieser Arbeit verwendeten Faktor zur Radiusbestimmung von Kreisobjekten an die Entfernung vom Objekt zum Sensor anzupassen. Wie die Evaluierung gezeigt hat, wird der Radius bei runden Objekten ab einem gewissen Abstand zwischen Objekt und Sensor zu klein berechnet. Dementsprechend müsste der Faktor für größere Abstände anpasst werden.

Ebenfalls bietet es sich an, einen ähnlichen Faktor wie bei Radiusberechnung von runden Objekten bei rechteckigen und linearen Objekten mit einzuführen, um den in 4.2 beschriebenen Fehler zu minimieren.

Bei einer Messung, beispielsweise in einem Raum, wo viele verschiedene Konturen von dem Sensor gemessen werden, kann es aufgrund der festgelegten Klassifizierungsbedingungen dazu kommen, dass nicht vorhandene Objekte als diese erkannt werden. Daher bietet es sich an, dann Programmcode so zu erweitern, dass ein Objekt mehrere Male hintereinander als Objekt erkannt werden muss, damit es auch als dieses klassifiziert wird. Dadurch könnten fehlerhafte Klassifizierungen vermieden werden.

Die Zeit, wie lange ein veröffentlichter Marker im *rviz* angezeigt wird, sollte an die Zeit der Markerberechnung angepasst werden, sodass dynamische Objekte nicht mehrfach angezeigt werden. Dabei muss darauf geachtet werden, dass sich die Berechnungszeit mit der Anzahl der erkannten Objekte dauerhaft ändert.

Außerdem wäre es interessant zu prüfen, wie sich der Algorithmus im Falle von dynamischen Objekten und somit mit dem Ausblick des Objekttrackings verhält.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] **Studyflix. (o.J)): Euklidische Distanz.** https://studyflix.de/mathematik/euklidische-distanz-1972 - zuletzt abgerufen am 25.07.2020

[2] **scikits learn. (o.J.): Comparing different clustering algorithms on toy datasets.** https://ogrisel.github.io/scikit-learn.org/sklearn-tutorial/auto_examples/cluster/plot_cluster_comparison.html - zuletzt abgerufen am 25.07.2020

[3] **Luber, Stefan; Litzel, Nico. (18.07.2018): Was ist der k-Means-Algorithmus?** https://www.bigdata-insider.de/was-ist-der-k-means-algorithmus-a-734637/ - Zuletzt abgerufen am 25.07.2020

[4] **Josiger, Marcus; Kirchner, Kathrin.: Moderne Clusteralgorithmen- eine vergleichende Analyse auf zweidimensionalen Daten.** Projektarbeit, Friedrich-Schiller-Universität, Jena, o.J.

[5] **Demberg, Very.: Mathematische Grundlagen, Custering.** Universität des Saarlandes, 2012

[6] **Y. Ng., Andrew; I. Jordan, Michael; Weiss, Yair. On Spectral Clustering: Analysis and an algorithm**. In NIPS 14. 2002

[7] **J. Alpert, Charles; Yao, So-Zen. Spectral Partitioning: The More Eigenvectors, The Better.** UCLA Computer Science Department, Los Angeles. 1995

[8] **Comaniciu, Dorin; Meer, Peter. Mean Shift Analysis and Applications.** Department of Electrical and Computer Engineering, Rutgers University, Piscataway, 1999

[9] **Comaniciu, Dorin; Meer, Peter. Mean Shift: A Robust Approach Toward Feature Space Analysis.** In: 2002 5th IEEE Transactions on pattern Analysis and machine intelligence, S. 603 – 619, 2002

[10] Onoro. Daniel**, Ryu, Jee-Hwan, Lensky, Artem:** Connected Compenents for a Fast an Robust 2D Lidar Data Segmentation. DOI: 10.1109/AMS.2013.31. 2013

[11] **Nguyen, A.; Le, B.: 3D Point Cloud Segmentation**. In: 2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM), S. 225–230, 2013

[12] **Klasing, K. et al.: A clustering method for efficient segmentation of 3D laser data.** In: 2008 IEEE International Conference on Robotics and Automation, S. 4043–4048, 2008

[13] **Yangeun Choe, Seunguk Ahn, Myung Jon Chung:** Fast Point Cloud Segmentation for an Intelligent Vehicle Using sweeping 2D laser scanners, in :2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), S. 38-43, 2012

[14] **Yan Peng, Dong Qu , Yuxuan Zhong, Shaorong Xie, Jun Luo:** The Obstacle Detection and Obstacle Avoidance Algorithm Based on 2-D Lidar, Proceeding of the 2015 IEEE International Conference on Information and Automation Lijiang, China, S.1648 – 1653, 2015

[15] Grilli, E. et al.: Point Clouds Segmentation And Classification Algorithms (2017) Grilli, E.; Menna, F.; Remondino, F.: A Review Of Point Clouds Segmentation And Classification Algorithms, in: ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information SciencesXLII-2/W3, S. 339–344, 2017

[16] **O.V.: Line Detection by Hough transformation**, S. 1-7, 2009

[17] **Deepali Ghorpade, Anuradha D. Thakare, Sunil Doiphode: Obstacle Detection and Avoidance Algorithm for Autonomous Mobile Robot using 2D Lidar** in: 2017 Third International Conference on Computing, Communication, Control and Automation (ICCUBEA), S. 1-6, 2017

[18] **Mathebibel. (o.J): Parameterform.** https://www.mathebibel.de/parameterform - zuletzt abgerufen am 25.07.2020

[19] **StudyHelp. (o.J): Abstandsberechnung.** https://www.studyhelp.de/online-lernen/mathe/abstand-punkt-gerade/ - zuletzt abgerufen am 25.07.2020

[20] **Mathebibel. (o.J): Winkel zwischen zwei Vektoren.** https://www.mathebibel.de/winkel-zwischen-zwei-vektoren - zuletzt abgerufen am 25.07.2020

[21] **Technische Universität Braunschweig. (o.J.): Gewichtete Punktebewertung.** https://methodos.ik.ing.tu-bs.de/methode/GewichtetePunktbewertung.html - zuletzt abgerufen am 25.07.2020

[22] **Robotics Weekend. (2019): 2D Mapping Using Google Cartographer and RPLidar with Raspberry Pi 3B+.** https://www.youtube.com/watch?v=qNdcXUEF7KU - zuletzt abgerufen am 25.07.2020

[23] **Klimov Konstantion. (2018): RP Lidar A1 – Inventor reassembly, CAD-Modell.** https://grabcad.com/library/rp-lidar-a1-inventor-reassembly-1 - zuletzt abgerufen am 25.07.2020

[24] **RS Components. (o.J.) Raspberry Pi 3 Model B SBC, CAD-Modell.** https://www.traceparts.com/en/product/rs-components-raspberry-pi-3-model-b-sbc?Product=10-26052016-104861 - zuletzt abgerufen am 25.07.2020

[25] **Pohl, Hartmut: Robot Operating System (ROS): Safe & Insecure.** Hochschule Bonn-Rhein-Sieg, 2014

[26] **Vlasuk, Vladislav: Grundlagen des Robot Operating Systmes**. Technische Hochschule Köln, 2019

[27] **Rudschies, Wolfgang; Kroher, Thomas. (2019): Autonomes Fahren, Digital entspannt in die Zukunft.** https://www.adac.de/rund-ums-fahrzeug/ausstattung-technik-zubehoer/autonomes-fahren/technik-vernetzung/aktuelle-technik/ - zuletzt abgerufen am 25.07.2020

[28] **Eberhardt, Dirk; Härter Hendrik. (2018): Die Vorteile von Lidar-Sensoren im autonomen Fahrzeug.** https://www.next-mobility.news/die-vorteile-von-lidar-sensoren-im-autonomen-fahrzeug-a-710222/ - zuletzt abgerufen am 25.07.2020

[29] **Flaig, Imelda. (2017): Autonomes Fahren erfordert neue Sicherheitskonzepte.** https://www.stuttgarter-nachrichten.de/inhalt.autozulieferer-zf-chef-autonomes-fahren-erfordert-neue-sicherheitskonzepte.e9c33d28-01d1-4b6e-99ff-2869a4163355.html - zuletzt abgerufen am 25.07.2020

[30] **O.V. (o.J.): rplidar package.** http://wiki.ros.org/rplidar - zuletzt abgerufen am 25.07.2020

[31] **Tully Foote. (o.J.): laser_filters package.** http://wiki.ros.org/laser_filters - zuletzt abgerufen am 25.07.2020

[32] **SLAMTEC: RPLIDAR A1 Introduction and Datasheet.** Shanghai Slamtec.Co.,Ltd, 2016

[33] **element 14; Farnell; Newark: Raspberry PI 3 Model B.** o.J.

[34] **Mathebibel. (o.J.): Hessesche Normalform.** https://www.mathebibel.de/hessesche-normalform - zuletzt abgerufen am 25.07.2020

[35] **Wagner, Dorothea: Algorithmische Geometrie: Konvexe Hülle.** Karlsruher Institut für Technologie, 2012

[36] **Ko, Sohyang; Jun, Seonsoo; Ryu, Yeonseung: A new Linux Swap System for Flash Memory Storage Devices.** International Conference on Computational Sciences and Ist Applications ICCSA, S 151-156, 2008

[37] **Leblebici, Robin: Laserscanner basierte Hinderiserkennung für einen autonomen Quadrokopter.** Bachelorarbeit, Julius-Maximilian-Universität Würzburg, 2015

[38] **Wilken, Heiko: Laserscanner-basierte Objekterkennung für ein Fahrspurführungssystem auf einer Multiprozessor FPGA-Plattform**. Masterarbeit, Hochschule für Angewandte Wissenschaften Hamburg, 2012

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

_____ _____

Datum, Ort Unterschrift