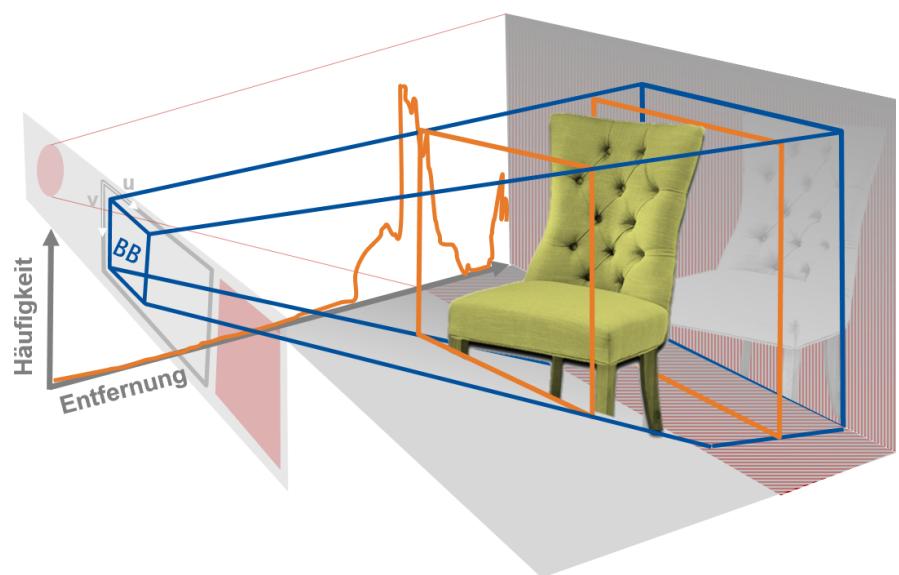


Kombinierte Kartierung und Objektlokalisierung mit einem mobilen Roboter



Studienarbeit S-02/17-599

Florenz Graf, B. Sc.
Matrikelnummer 3209710

Hannover, Februar 2017

Fachprüfer Prof. Dr.-Ing. Tobias Ortmaier
Betreuer Simon Aden, M. Sc.

Studienarbeit

Herr Florenz Graf, Matr.-Nr. 3209710

Kombinierte Kartierung und Objektlokalisierung mit einem mobilen Roboter

Combined Mapping and Object Localization with a Mobile Robot

Allgemeines:

Aktuelle Forschungen im Bereich der mobilen Robotik zielen auf generische Interaktion mit der Umgebung ab: Ein Robotersystem soll adaptiv und möglichst unabhängig von seiner Sensorik in der Lage sein, mit seiner Umwelt zu interagieren. Oft zieht man zu diesem Zweck eine Karte hinzu, insbesondere um eine Trajektorienplanung für Arm und Basis des mobilen Roboters umzusetzen. Während für die Basis meist eine 2D-Karte ausreicht, ist für die Armbewegung oftmals eine Umgebungskarte in drei Dimensionen erforderlich.

Aufgabe:

Die studentische Arbeit befasst sich mit der 3D-Kartierung einer Umgebung. Dazu wird eine geeignete Sensorik (RGBD-Kamera, Stereo-Kamera) verwendet, um die Strukturen der Umgebung möglichst genau in der Karte abzubilden. Parallel soll der mobile Roboter (KUKA youBot) Methoden aus dem Bereich des *Machine Learning* – genauer dem Bereich *Deep Learning* – verwenden, um übliche Objekte seiner Umgebung zu erkennen und in der Karte zu platzieren. In einem Wohnraum könnte der Roboter beispielsweise Stühle und Tische erkennen. Zur Umsetzung der Arbeit soll eine intensive Literaturrecherche entsprechende Methoden herausstellen und ggf. bereits bestehende Implementierungen sollen wiederverwendet werden.

Damit ergeben sich die folgenden Aufgabenpunkte:

- Recherche mit dem Ziel, möglichst geeignete Verfahren herauszustellen
- Implementierung eines Systems zur 3D Kartierung
- Implementierung eines Systems zur Detektion einer Auswahl von Objekten
- Implementierung eines Datenformats für 3D-Karten mit der Möglichkeit zum Einfügen und Verändern von Objekten
- Zusammenführen
- Analyse und Gegenüberstellung der Versuche bezüglich Zuverlässigkeit, Kartierungszeit, Genauigkeit der Objektplatzierung

Zur Implementierung der Arbeit werden Python/C++ und ROS als Framework empfohlen.

Die Bearbeitungszeit beträgt 300h.

Ausgabe der Aufgabenstellung: 04.11.2016

Betreuer: M. Sc. Simon Aden

(Prof. Dr.-Ing. Ortmaier)

(Florenz Graf)

Ich versichere, dass ich diese Studienarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Hannover, 05. Februar 2017

Inhaltsverzeichnis

1 Einleitung	1
1.1 Motivation	1
1.2 Problemstellung	2
1.3 Zielsetzung	3
1.4 Systemanforderungen	3
2 Material und Methoden	5
2.1 KUKA youBot	5
2.2 Robot Operating System	6
2.3 Objekterkennung mit Deep Learning	7
2.3.1 Grundlagen zu Deep Learning	7
2.3.2 Convolutional Neural Networks	10
2.3.3 Klassifizierung versus Detektion	12
2.3.4 Wettbewerbe zur Objektklassifizierung und -erkennung	13
2.3.5 Performance aktueller Objektlokalisierungen	14
2.4 RGB-D Kartierung	16
2.4.1 RGB-D-Kamera	16
2.4.2 Kartierung	17
3 Entwurf	19
3.1 Evaluation bestehender Ansätze	19
3.1.1 Vergleich von bestehenden Objekterkennungen	20
3.1.2 Vergleich von bestehenden RGB-D-Kartierungen	21
3.2 Konzepterstellung	23
3.2.1 Konzept der Objekterkennung	25
3.2.2 Konzept der Segmentierung	26
3.2.3 Konzept der Kartierung	27
4 Implementierung	29
4.1 Implementierung Package Objekterkennung	30
4.2 Implementierung Package Segmentierung	31
4.3 Implementierung Package Kartierung	36

5 Leistungsbewertung	37
5.1 Arena der Testumgebung	37
5.2 Vorbereitung und Konfiguration des youBots	38
5.3 Ergebnisse	39
5.3.1 Ergebnisse und Auswertung der Objekterkennung	41
5.3.2 Ergebnisse und Auswertung der Segmentierung	41
5.3.3 Ergebnisse und Auswertung der Kartierung	43
6 Fazit	44
6.1 Zusammenfassung	44
6.2 Ausblick	45
Anhang	47
A Objektklassen der Objekterkennung	48
B Weitere Abbildungen von Ergebnissen	49
C Codeausschnitte der <i>segmentation_node</i>	52
Literaturverzeichnis	58
Linksammlung	61

Bildverzeichnis

2.1	KUKA youBot (Quelle [MAK12])	5
2.2	ROS Konzept (Quelle: [Ade13])	7
2.3	Klassifizierung mittels Deep Learning (Quelle: [Bro15])	8
2.4	Vereinfachtes Feed-Forward-Netzwerk für die Bildklassifizierung (Quelle: nach [Nie16])	10
2.5	Rezeptives Feld eines <i>Convolutional Neural Networks</i> (Quelle: [Nie16]) . . .	10
2.6	<i>Convolutional Neural Network</i> für die Bildklassifizierung (Quelle: [KSH12])	11
2.7	Beispielhafter Ablauf einer regionbasierten Objekterkennung mit einem CNN (Quelle: [RDGF16])	13
2.8	YOLO9000: Echtzeit state-of-the-art Deep Learning Objekterkennung von 9000 Klassen (Quelle: [RF16])	14
2.9	Entscheidungsbaum zur Kombination von Datensätzen (Quelle: [RF16]) . . .	15
2.10	Aktive Triangulation durch phasenverschobene Streifenbilder. Abgebildet ist das Muster einer Phasenverschiebung von 0, 90 und 180 Grad (Quelle: [Jäh05])	16
2.11	<i>Octomap</i> eines Arbeitsplatzes, erzeugt mit einer RGB-D-Kamera (Quelle: [End15])	18
3.1	Konzept für die kombinierte Kartierung und Objektlokalisierung mit dem youBot bei ausgelagerter Berechnung des CNN	25
3.2	Konzept zur Transformation zwischen 2D und 3D	27
4.1	Schematische Kommunikation der Nodes (blau) über Topics (grün) unter dem ROS-Master	29
4.2	Farb- und Tiefenbild eines Drehstuhles	33
4.3	Histogramm aus Tiefenbild, Auschnitt Bounding Box Bild 4.2b. Horizontal ist die Entfernung zur Kamera (links nah, rechts fern), vertikal ist die Häufigkeit der Pixel mit identischer Entfernung aufgetragen	34
4.4	Datenverarbeitung innerhalb der <i>segmentation_node</i>	35
5.1	Übersicht der Arena	37
5.2	Gesamtergebnis der aufgenommenen Testarena	39
5.3	Gesamtergebnis der aufgenommenen Testarena	40

5.4 Fehlerhafte Segmentierung eines Stuhles	42
5.5 Erfolgreiche Segmentierung einer Tasse und einer Flasche	42
B.1 Vortests zur histogrammbasierten Segmentierung	49
B.2 Absorption schwarzer Flächen am Beispiel einer schwarzen Tastatur	50
B.3 Bild aus Videosequenz, Verdeckung dynamischer Objekte	51

Tabellenverzeichnis

3.1	Echtzeitfähige state-of-the-art (12/2016) Objekterkennungen durch <i>Convolutional Neural Networks</i> (regionbasiert)	21
3.2	Echtzeitfähige state-of-the-art (12/2016) RGB-D-Kartierungen	22
A.1	COCO-Datensatz Objektklassen	48
A.2	VOC-Datensatz Objektklassen	48

1 Einleitung

Dieses Kapitel verleiht dem Leser eine erste Orientierung über das Thema und zeigt die Motivation, die Problemstellung sowie die Zielsetzung dieser Studienarbeit. Weiterhin werden grundlegende Anforderungen definiert.

1.1 Motivation

Durch nahezu unbegrenzte Einsatzmöglichkeit mobiler Roboter entstehen neue Forschungstrends, die durch den technischen Fortschritt der vergangenen Jahren ermöglicht werden. Der alltägliche und sinnvolle Einsatz von mobilen Robotern kann nur gewährleistet werden, wenn deren Eigenschaften die Erwartungen hinsichtlich des Autonomiegrades erfüllen. Hierfür muss die Umgebung von den Robotern wahrgenommen und verstanden werden. Die Umsetzung erfordert einen hohen Komplexitätsgrad, welcher zunehmend mit Methoden aus dem maschinellen Lernen erreicht wird. Der Ansatz "*Perception-Action-Learning*" charakterisiert hierbei die wesentlichen Bestandteile, die für die Interaktion mit der Umwelt notwendig sind [Sch14].

Klassische Methoden trennen die Bestandteile der Interaktion. Zur Perzeption (engl. *perception*) werden in der mobilen Robotik verschiedene Sensortypen verwendet: Kameras, Laserscanner, Ultraschallsensoren und Kraft- und Momentsensoren. Hinsichtlich des sehr hohen Informationsgehaltes und der kontaktlosen Messmethode haben RGB-D-Kameras eine hohe Bedeutung [Ort16]. Auf die Perzeption folgen Handlungsabläufe (engl. *action*), die einerseits Informationen der Perzeption und andererseits Wissen - das angelernt oder hinterlegt werden kann (engl. *learning*) - benötigen. Klassisch wird hierzu unterteilt:

Die Navigation beruht auf einer zwei- oder dreidimensionalen Karte der Umgebung. Diese Karte kann aus der zeitlichen Fusion von Sensordaten erlernt werden.

Der Manipulation mobiler Robotern liegt meist eine Objektlokalisierung zugrunde. Durch eine bildbasierte Manipulation werden Objekte erkannt, erfasst und aufgenommen.

Gelingt es die Objektlokalisierung und die Kartierung zu kombinieren, entstehen neue Möglichkeiten. Das Ziel dieser Arbeit ist die Erstellung eines intelligenten Umgebungsmodells. Die Informationen der Sensoren können vollständiger verwendet werden, indem zusätzliche Informationen aus der Objekterkennung in die Kartierung einfließen. Somit können beispielsweise dynamische Objekte, durch ihre temporäre Position in der Karte ausgeblendet, oder das

Verhalten des Roboters gegenüber dynamischen Objekten differenziert werden.

Bei einer echtzeitfähigen Implementierung der kombinierten Kartierung und Objektlokalisierung werden somit deutliche Vorteile hinsichtlich des interaktiven Verhaltens zwischen mobilen Robotern und unbekannten Umgebungen erwartet. Die Umsetzbarkeit wird in dieser Arbeit untersucht.

1.2 Problemstellung

Als Mensch fällt es uns einfach, unsere Umgebung wahrzunehmen, uns zu orientieren und Entscheidungen ausgehend von unserem Vorwissen zu treffen. Aber wie muss ein Roboter anhand von mathematischen Zusammenhängen implementiert werden, um vergleichbare Ergebnisse zu erlangen? Einer Maschine muss diese für uns triviale Fähigkeit erst angelernt werden. Der bisherige Ansatz mobiler Roboter ist, dass ein auf einen Anwendungsfall abgestimmtes Modell zugrunde liegt. Hierbei wird ablaufgesteuert zwischen verschiedenen Methoden gewechselt. Die Kartierung und die Objektlokalisierung haben durch den direkten Bezug zur Perzeption und somit als Äquivalenz zum Auge eine besonders hohe Bedeutung und Komplexität.

Durch Lokalisierung des Roboters und gleichzeitiger Aufnahme der Umgebung (engl. *Simultaneous Localization and Mapping*, kurz SLAM) kann eine Karte erstellt werden. Die Karte dient zur Navigation und enthält üblicherweise keine weiteren Informationen.

Für die Objektlokalisierung kann über neuronale Strukturen, die dem menschlichen Gehirn nachempfunden sind, ein Modell trainiert werden, dass in der Lage ist Objekte wie der Mensch zu erkennen, klassifizieren und lokalisieren. Rückschlüsse auf die Klasse von einem noch nie zuvor gesehenen Objekt sind möglich, was insbesondere bei mobilen Systemen durch die mögliche unbekannte Umgebung relevant ist. Moderne Methoden aus dem maschinellen Lernen, in diesem Fall dem *Deep Learning*, eignen sich hierfür.

Für das intelligente Umgebungsmodell, dass aus der Kombination von Kartierung und Objektlokalisierung erstellt werden kann, ist, um echtzeitfähiges Verhalten zu erreichen, modernste Technik notwendig. Pro Sekunde müssen terrawise Rechenoperationen von parallel ablauenden Prozessen durchgeführt werden. Eine besondere Herausforderung sind hierbei die Deep-Learning-Methoden, die im Vergleich zu konventionellen Methoden eine erhöhte Anzahl an Berechnungsschritten benötigen. Die Lösung kann entweder durch das Erhöhen der Rechenkapazität oder einer effizienten Implementierung erreicht werden. Wobei ersteres durch die Hardware mobiler Systeme limitiert ist.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist es einen möglichst adaptiven Ansatz zu implementieren, um eine kombinierte Kartierung und Objektlokalisierung bei einem mobilen Roboter zu ermöglichen. Für die Zielerfüllung sind mehrere Schritte notwendig. Ausgehend von einer umfangreichen Literaturrecherche werden möglichst geeignete Verfahren zur dreidimensionalen Kartierung und zur Objekterkennung mittels eines adaptiven Deep Learning Ansatzes herausgearbeitet. Die Adaptivität bezieht sich hierbei auf die zu erkennenden Objektklassen, die austauschbar sein sollen. Vorhandene Implementierungen können wiederverwendet werden. Die verwendeten Ansätze werden anschließend fusioniert.

Um die Performance der erstellten Implementierung zu ermitteln und mögliche Schwachstellen zu identifizieren und zu optimieren wird ein mobiler Roboter in einer unbekannten Umgebung seine Fähigkeiten unter Beweis stellen müssen. Abschließend werden die Ergebnisse hinsichtlich der Genauigkeit, Zuverlässigkeit und Echtzeitfähigkeit analysiert.

1.4 Systemanforderungen

In diesem Abschnitt werden knapp die Systemanforderungen definiert. Diese müssen bei der Erarbeitung der Implementierung (bis einschließlich Kapitel 4) berücksichtigt werden.

- (1) Implementierung einer kombinierten Kartierung und Objektlokalisierung auf einem mobilen Roboter
- (2) Kartierung mit einer RGB-D Kamera
- (3) Objektlokalisierung mit Methoden aus dem Deep Learning. Adaptiv hinsichtlich der Austauschbarkeit der zu erkennenden Objekte.
- (4) Sinnvolle Realisierung eines intelligenten Umgebungsmodells
- (5) Echtzeitfähigkeit des mobilen Roboters
- (6) Effiziente Implementierung auf mobiler Hardware
- (7) Hohe Zuverlässigkeit
- (8) Hohe Genauigkeit

Die Hauptanforderungen werden ergeben sich aus der Motivation (Abschnitt 1.1). Kombinierte Objektlokalisierung und Kartierung muss ermöglicht werden (1). Die Kartierung soll mit einer

RGB-D-Kamera durchgeführt werden (2), vgl. Abschnitt 2.4, um die Informationen aus der Umgebung bestmöglich aufzunehmen.

Die Objekterkennung mit Methoden aus dem Deep Learning, siehe Abschnitt 2.3, muss dabei möglichst adaptiv hinsichtlich der Austauschbarkeit der zu erkennenden Objekte sein. Grund hierfür ist, dass zunächst die Umsetzbarkeit auf einer mobilen Plattform überprüft werden soll. Die zu erkennenden Objekte können exemplarisch gewählt werden und sollen für Folgearbeiten geändert beziehungsweise erweitert werden können (3).

Aus dem kombinierten Ansatz soll eine intelligente Umgebungskarte angeleitet werden. Hierbei sollen sinnvolle Features erarbeitet werden, um Vorteile gegenüber einer konventionellen Kategorisierung zu erhalten (4).

Umfangreiche Implementierungen sind nötig um diese Aufgaben zu erfüllen. Dennoch müssen diese Implementierungen es ermöglichen, dass mobile Roboter in Echtzeit agieren kann (5). Je schneller die Befehle abgearbeitet sind, desto bessere Ergebnisse werden hinsichtlich Reaktionszeit und Genauigkeit erwartet. Idealerweise ist die Rechenkapazität ausreichend um weitere Anwendungen parallel in Echtzeit ausführen zu können.

Zur Erfüllung dieser Anforderung muss entweder die Rechenkapazität erhöht, oder die Effizienz der gesteigert werden. Ein geringer Rechenbedarf auf dem youBot ermöglicht den Einsatz von effizienterer Hardware, sodass wenig Energie aus dem Akku benötigt wird (6).

Vorhandene Ressourcen sollen wenn möglich verwendet werden.

Zuverlässigkeit (7) und eine hohe Genauigkeit (8) der Implementierung werden selbstverständlich erwartet.

2 Material und Methoden

Für die vorliegende Arbeit wird ein mobiler, bodengebundener Roboter von KUKA (youBot) verwendet. Selbstverständlich können die Erkenntnisse und Implementierungen aus dieser Arbeit auch auf andere mobile Roboter mit entsprechender Sensorik transferiert werden.

2.1 KUKA youBot

Der youBot (KUKA Roboter GmbH, Augsburg, Deutschland) ist ein mobiler, elektrisch angetriebener Schulungs- und Forschungsroboter, der vom Institut für Mechatronische Systeme an der Universität Hannover für verschiedene Lehrveranstaltungen und wissenschaftliche Projekte, wie diese Studienarbeit, verwendet wird.

In Hinblick auf die Applikation der Implementierung auf den youBot (Bild 2.1), werden in diesem Abschnitt die für diese Arbeit wesentlichen Leistungsmerkmale genannt.

Die holonome Plattform wird von vier Omniwheels angetrieben. Auf dieser Plattform ist ein 5-achsiger Roboterarm mit Zweifingergreifer montiert [MAK12]. Einige Modifizierungen und



Bild 2.1: KUKA youBot (Quelle [MAK12])

Optimierungen wurden am youBot gegenüber dem Auslieferungszustand vom Institut für Mechatronische Systeme vorgenommen. Derzeit (Stand 12/2016) sind zwei Laserscanner (Hokuyo URG-04¹) für die horizontale Überwachung und eine RGB-D-Kamera (Intel RealSense SR300²) für die dreidimensionale Überwachung des näheren Umfeldes montiert.

In der Plattform ist ein Prozessor (Intel Core i7-4790T³), auf dem die Linux Distribution Ubuntu Trusty Tahr (14.04 LTS) installiert ist, untergebracht. Hierauf agiert das Robot Operating System, kurz ROS (Abschnitt 2.2), in der Version Indigo Igloo, welches die Datenverwaltung übernimmt und somit Roboterabläufe koordiniert. Über eine WLAN-Schnittstelle am youBot können Daten extern übermittelt werden.

Der youBot wird für diese Arbeit ohne Hardwaremodifikationen übernommen. Die Software muss jedoch um die Objekterkennung mit Deep Learning und die RGB-D Kartierung erweitert werden. Hierzu bestehen keine direkten Vorarbeiten am Institut.

2.2 Robot Operating System

In Hinblick auf den Entwurf (Kapitel 3) und der Implementierung (Kapitel 4) des mobilen Roboters wird in diesem Abschnitt die Grundstruktur von ROS knapp beschrieben, um dessen Zusammenhänge bei der Konzepterstellung in Abschnitt 3.2 mitberücksichtigen zu können.

Das Robot Operating System (ROS)[L4] ist ein open-source Software-Framework für Robotersysteme und wird auf dem youBot (Abschnitt 2.1) verwendet. ROS bietet integrierte Bibliotheken und Werkzeuge, die eine effektive Roboterimplementierung ermöglichen.

Die Softwarestruktur von ROS sieht vor, dass auf dem Roboter ein Netzwerk von mehreren kleinen Programmen, sogennanten *Nodes*, in einem *Workspace* ausgeführt wird. Die Nodes werden in *Packages* organisiert und kommunizieren untereinander über standardisierte Nachrichten (vgl. Bild 2.2). Der Vorteil dieser modularen Struktur besteht in der Unabhängigkeit und Austauschbarkeit dieser Nodes. Nodes können in unterschiedlichen Programmiersprachen - üblicherweise in C++ oder Python - implementiert werden. Die zentrale Einheit bildet der *Roscore*, bestehend aus einem Master und einem Parameterserver. Der Master koordiniert die Prozessabläufe. Der Parameterserver speichert zentral die wichtigsten Parameter des Systems. Die Nachrichten zwischen den Nodes werden entweder als *Topic* oder *Service* übertragen. Topics können Nachrichten ausschließlich unidirektional übertragen. Versendet beziehungsweise veröffentlicht (engl. *publish*), eine Node Nachrichten, wird dies als *Publisher* bezeichnet. Abonniert (engl. *subscribe*) eine Node solch eine Nachricht, wird sie als *Subscriber* bezeichnet.

¹Datenblatt unter [L1] verfügbar

²Datenblatt unter [L2] verfügbar

³Technische Spezifikationen unter [L3] verfügbar

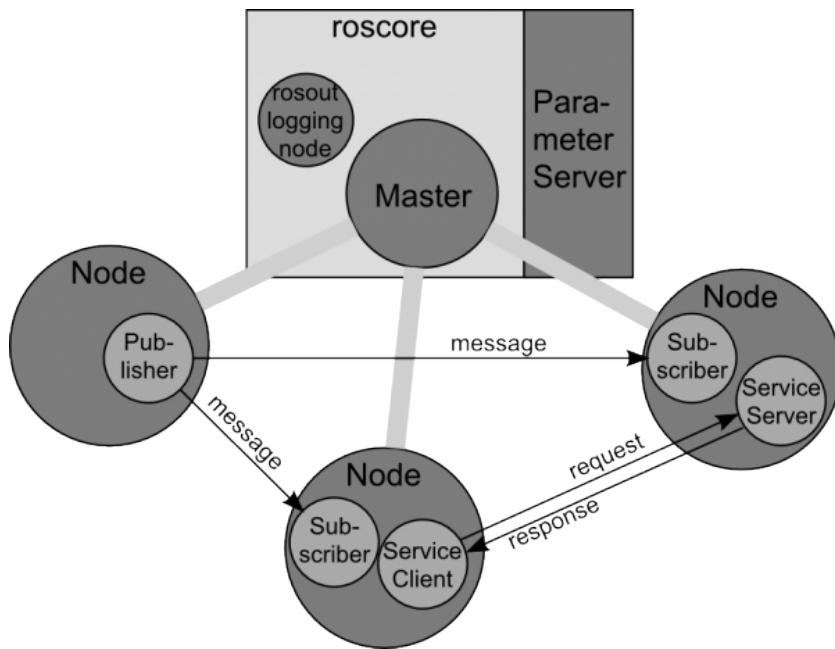


Bild 2.2: ROS Konzept (Quelle: [Ade13])

Die Kommunikation als Service ist hingegen bidirektional. Auf Anfrage (engl. *request*) einer Node wird von der anderen Node eine Rückmeldung (engl. *response*) gegeben [Kou16]. Alle genannten Zusammenhänge und Begriffe sind für das Verständnis der Implementierung in Kapitel 4 von großer Bedeutung. Für interessierte Leser wird auf Tutorials von ROS verwiesen, zu finden sind diese unter: wiki.ros.org/ROS/Tutorials

2.3 Objekterkennung mit Deep Learning

Wie eingangs erwähnt, wird in dieser Arbeit die Objekterkennung mittels Deep Learning implementiert. Zunächst wird knapp beschrieben was Deep Learning ist, wie Deep Learning Netzwerke aufgebaut sind und wie der Ablauf von Deep Learning ist.

2.3.1 Grundlagen zu Deep Learning

Deep Learning ist ein vergleichsweise neues (seit 1997) und schon sehr populäres Teilgebiet des maschinellen Lernens. Zwei Faktoren sind für den derzeitigen Popularität ausschlaggebend: Big Data und schnelle Rechner. Deep Learning ist eine Kombination aus verschiedenen Techniken. Die Basis bilden erweiterte künstliche neuronale Netze, die ähnlich wie das menschliche Gehirn funktionieren.

Sei es durch Google⁴, Facebook⁵, Spotify⁶ oder Siri von Apple⁷ - nahezu überall wird Deep Learning für Interaktionen zwischen Mensch und Maschine eingesetzt. Nach Larry Brown von NVIDIA kann Deep Learning wie folgt definiert werden:

Systems that learn to recognize objects that are important, without us telling the system explicitly what that object is ahead of time [Bro15].

Aus diesem Zitat geht hervor, dass Deep-Learning-Netzwerke in der Lage sind, komplexe Zusammenhänge ohne Vorgabe einer expliziten Lösung zu verstehen. Die größten Einsatzgebiete sind die Verarbeitung von Bild-, Ton- und Textdaten. Somit genau da, wo herkömmliche mathematische Beschreibungen durch den hohen Anteil an unklaren Zusammenhängen an ihre Grenzen stoßen. Deep Learning liefert bei den genannten Einsatzgebieten teilweise deutlich bessere Ergebnisse als traditionelle Methoden⁸[Jäs16].

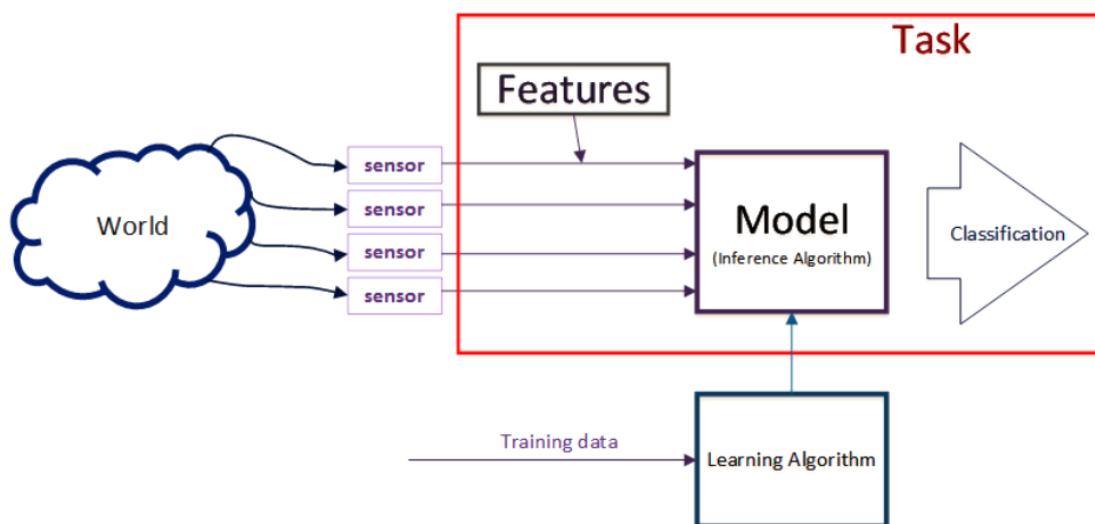


Bild 2.3: Klassifizierung mittels Deep Learning (Quelle: [Bro15])

Für die Erkennung von Objekten mittels Sensoren, ist es notwendig, dass die Sensordaten der Umwelt korrekt interpretiert werden. Hierzu müssen Features der Sensordaten, bei Deep Learning basierend auf einem trainierten Modell, richtig zugeordnet werden. Dieses Modell wird mit klassifizierten Trainingsdaten angelernt und ist in der Lage die Features den entsprechenden

⁴Google verwendet Deep Learning bei der Bildersuche. Weiterhin bietet Google open-source Software für die Erstellung von Deep Learning Netzwerken an (TensorFlow) [L5]

⁵Facebook verwendet Deep Learning für Textverstehen [L6]

⁶Spotify generiert mittels Deep-Learning-Methoden Musikvorschläge, basierend auf bereits wiedergegebenen Musiktiteln [L7]

⁷Siri ist eine Sprachverarbeitungssoftware von Apple und verwendet Deep Learning um komplexe Spracheingaben zu verstehen [L8]

⁸Support Vector Maschine, Regression etc. siehe [Bro15]

Objektklassen zuzuordnen (Bild 2.3). Neue Techniken erlauben nachträgliches unabhängiges Training einer lokalen Region im Netzwerk, Standardmäßig wird jedoch ein Netzwerk für eine Problemstellung nur einmal vor dessen Einsatz trainiert.

Das Lernen wird üblicherweise überwacht durchgeführt (*supervised learning*). Ein ideales System wechselt automatisch zwischen überwachten, unüberwachten (*unsupervised learning*) und verstärkten Lernen (*reinforcement learning*) [Bro15].

Als Trainingsdaten dienen archivierte Sensordaten mit zusätzlichem Label, beispielsweise Bilder mit der Information was zu sehen ist. Je mehr Trainingsdaten - Stichwort *Big Data* - desto bessere Ergebnisse können erwartet werden. Social-Media-Unternehmen können hierbei auf besonders große Datenmengen ihrer Nutzer zurückgreifen. Auf viele Datenmengen kann öffentlich zugegriffen werden. Für eine 2D-Datenbank kommt beispielsweise die Google Bildersuche⁹, für eine 3D-Datenbank die Redwood Datenbank [L10] infrage. Selbstverständlich können auch eigene Daten mit Label verwendet werden. Features der Trainingsdaten werden beim Anlernen des Deep-Learning-Netzwerkes automatisch identifiziert.

Im Unterschied zu herkömmlichen Lernverfahren liegt bei Deep-Learning-Netzwerken (engl. *Deep Neural Networks*, kurz DNN) eine mehrschichtige und somit tiefe (engl. *deep*) Lernstruktur vor. DNNs bestehen aus künstlichen Neuronen, die in bis zu 1000 miteinander verknüpften Schichten (engl. *Layers*) angeordnet sind [Jäs16]. Für grundlegendes Verständnis über die Funktionsweise von künstlichen Neuronen, beziehungsweise neuronalen Netzen, wird auf [Ert13] und [Neh02] verwiesen. Der Schichtaufbau besteht aus einer Eingangsschicht, gefolgt von einer Anzahl an verdeckten Schichten und einer Ausgangsschicht. Die Ergebnisse von mehrschichtigen neuronalen Netzen liefern nach [PMB13] bei nicht trivialen Problemstellungen im Vergleich zu flachen neuronalen Netzen bessere Ergebnisse. Mit Hilfe von verdeckten Lagen kann das Abstraktionslevel gesteigert werden. Hierdurch können die eingangs erwähnten, komplexen, nicht-lineare Zusammenhänge verstanden werden. Ausgehend von Sensordaten werden schichtweise hierarchische Strukturen gebildet. Ein exemplarischer Schichtaufbau für Bildklassifizierungen von Objekten könnte sein (siehe Bild 2.4):

Pixel \Rightarrow Kanten \Rightarrow lokale Muster \Rightarrow Objektteile \Rightarrow Objekte Die Verknüpfung der künstlichen Neuronen zwischen den Schichten kann auf vielfältige Art implementiert werden. Es wird zwischen voll vernetzten und partiell vernetzten Schichten unterschieden. Sind die Neuronen ausschließlich in Laufrichtung verknüpft, wird dies als *Feed-Forward-Netz* bezeichnet. Gibt es hingegen seitliche oder rückläufige Verknüpfungen, ist von rekurrenten Netzen die Rede. Für weiterführende Informationen zu Deep Learning Architekturen wird auf [Ben09] verwiesen. Folgend wird sich ausschließlich auf die für die Bilderkennung relevanten Architekturen beschränkt.

⁹Hilfreich sind Programme für den automatischen Download aller Suchergebnisse, z.B. für Google Chrome die Erweiterung Fatkum Batch [L9]

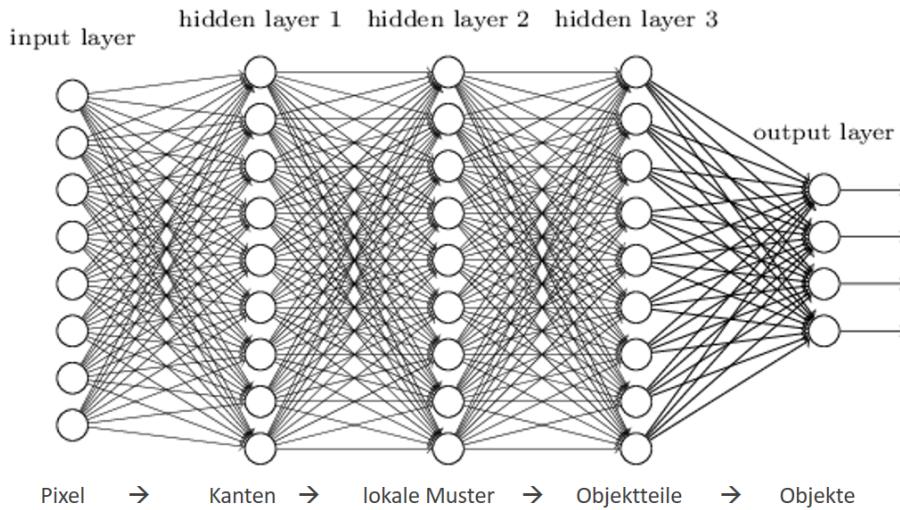


Bild 2.4: Vereinfachtes Feed-Forward-Netzwerk für die Bildklassifizierung (Quelle: nach [Nie16])

2.3.2 Convolutional Neural Networks

In Anlehnung an den visuellen Cortex des Menschen, also der Bereich des Gehirns indem die visuelle Wahrnehmung stattfindet, werden für die Bildverarbeitung typischerweise Faltungsnetzwerke (engl. *Convolutional Neural Networks*, kurz CNN), welche zu den Feed-Forward-Netzen gehören, verwendet. Bei einer *Convolution Layer* wird die Anzahl an Neuronen einer Schicht, die mit einem Neuron der nachfolgenden Schicht verknüpft ist, als *rezeptives Feld* bezeichnet (Bild 2.5).

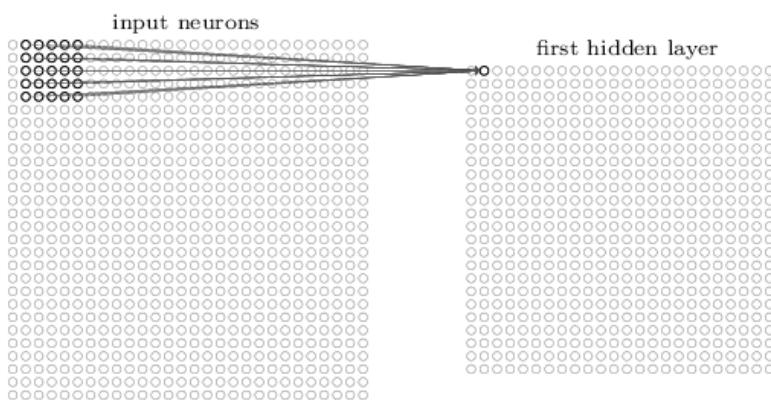


Bild 2.5: Rezeptives Feld eines *Convolutional Neural Networks* (Quelle: [Nie16])

Pro Schicht können mehrere dieser Felder überlagert sein. Die Informationen der rezeptiven Felder werden mit einem Filter mathematisch gefaltet (engl. *convolution*) und an die Neuronen der nachfolgenden Schicht weitergegeben. Die Filtergewichte sind dabei innerhalb einer *Convolution Layer* konstant. Neben Faltungsschichten gibt es sogenannte *Max Pooling Layers*. Diese folgen in der Regel einer Faltungsschicht und haben das Ziel die Datenmenge zu reduzieren. Hierbei werden Einträge mit geringer Aussagekräftigkeit entfernt, indem nur der maximalen Wert aus einer quadratischen Region gefiltert und an die Folgeschicht weitergegeben wird. Die Anzahl der Convolution- und Max-Pooling-Schichten kann beliebig gewählt werden. Die ersten Schichten sind meist partiell verknüpft und die letzten Schichten voll verknüpft [Bro15] [Nie16]. In der letzten Schicht wird üblicherweise über eine Softmax Funktion die prozentuale Zuordnungswahrscheinlichkeit zwischen 0 und 1 berechnet (Sigmoidfunktion), die dem Nutzer eine Aussage über die Qualität der Ausgabe erlaubt. Bild 2.6 zeigt den Aufbau eines CNNs für die Bildklassifizierung. Dieses Netzwerk gewann 2012 die *Large Scale Visual Recognition Challenge* [Bro15].

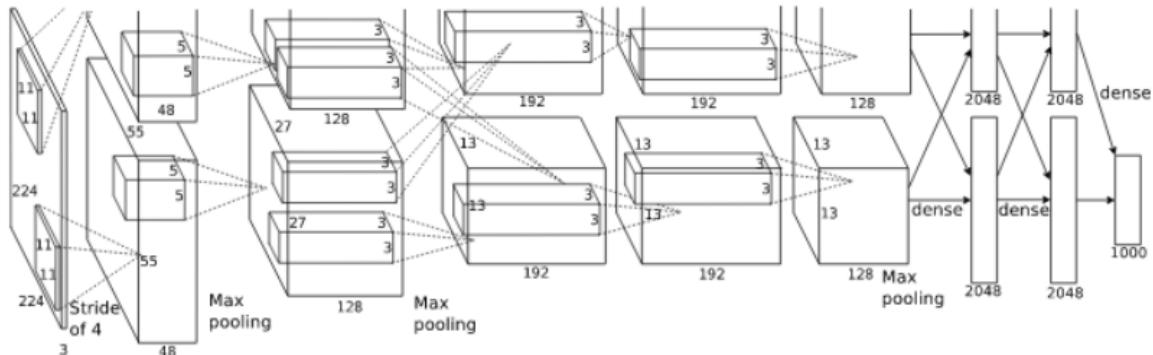


Bild 2.6: Convolutional Neural Network für die Bildklassifizierung (Quelle: [KSH12])

Die Anzahl an Neuronen in der Eingangsschicht ist 150.528 gefolgt von verdeckten Lagen mit 253.440, 186.624, 64.896, 64.896, 43.264, 4096 und 4096 Neuronen und der Ausgangslage mit 1000 Neuronen (drei Farbkanäle werden berücksichtigt) [Nie16]. 1000 verschiedene Objektklassen¹⁰ können mit diesem Netzwerk anhand von RGB-Bildern mit einer Auflösung von 224x224 Pixeln klassifiziert werden [KSH12].

Die enorme Anzahl an Neuronen moderner CNN und die damit verbundenen Rechenoperationen benötigten für die Echtzeitfähigkeit (vgl. Anforderung (5)) eine hohe Rechenkapazität. Weiterhin ist die Dauer des Trainings, beziehungsweise des Anlernens der CNNs, abhängig der Rechenleitung. Diese Rechenleistung empfiehlt sich mit einer GPU, anstatt mit einer CPU,

¹⁰Entspricht Anzahl der Ausgangsneuronen im Netzwerk

bereitzustellen. Deutlich wird der Unterschied an einem Vergleich zwischen Preis zu Rechenleistung, beispielsweise zwischen einer Nvidia TitanX¹¹ und der CPU des youBots (Intel Core i7-4790T¹²). Bei der Nvidia TitanX beträgt der Preis 0,12 USD pro GFLOP, bei dem i7-4790 hingegen 11,22 USD pro GFLOP.

Das Training von CNNs der Größe aus Bild 2.6 benötigt selbst auf Systemen mit der zuvor genannter Grafikkarte mehrere Wochen. Da das Training der CNNs vor dessen Einsatz durchgeführt wird, muss dieses Anlernen nicht zwingend auf der später eingesetzten Hardware stattfinden, sondern kann auf Großrechnern durchgeführt werden.

Nachdem Anlernen sind CNN einsatzbereit. Die Features der Eingabesequenz werden mit den angelernten Features des Netzwerkes, indem durch Faltung die Merkmale verglichen werden, abgeglichen. Normalerweise wird als Input die zweidimensionale RGB-Bildsequenz verwendet. Einige Ansätze verwenden zusätzlich das Tiefenbild einer Stereokamera, um Objekte dreidimensional zu erkennen. [TL16]. Allerdings sind diese Ansätze, wie beispielsweise die Implementierung von Eitel u.a. [ESRB15] derzeit noch nicht echtzeitfähig.

Die Ausgabe der CNN ist die Information zu welcher angelernten Klasse die Eingabefeatures mit Angabe der Wahrscheinlichkeit gehören.

2.3.3 Klassifizierung versus Detektion

Die bis hier beschriebene Faltungsarchitektur eignet sich zur Klassifizierung von Objekten. Für diese Studienarbeit wird jedoch eine Detektion der Objekte benötigt, sodass neben der Klasse die Region des Objektes erkannt werden muss.

Für die Bestimmung dieser Region gibt es verschiedene Ansätze. In [RHGS16] wird ein Ansatz zur Detektion von Objekten im zweidimensionalen Bild vorgestellt. Zwischen Sensorinput und der ersten Schicht des CNN wird eine Schicht zwischengeschaltet, die mögliche Regionen von Objekten in Form von Bounding Boxen¹³ vorschlägt. Die Schätzung basiert hierbei auf Merkmalen des Eingabebildes. Die Bounding Boxen werden vom CNN klassifiziert und dann in einer weiteren Neuen, vor der Ausgang befindlichen Schicht, validiert.

Ähnlich hierzu, wird in [RDGF16] ein Ansatz beschrieben, der allerdings das Eingabebild zunächst in ein Gitter zerlegt. Im Anschluss werden wie in [RHGS16] Bounding Boxen im Gesamtbild geschätzt. Parallel hierzu wird für jedes Gitterelement die Objektklasse über das CNN bestimmt. Abschließend werden die Bounding Boxen mit dem klassifizierten Gitter abgeglichen und mit statistischer Wahrscheinlichkeit ausgegeben (vgl. Bild 2.7).

¹¹Die Nvidia TitanX ist eine high-end Grafikkarte, Preis derzeit 1299 USD, Rechenleistung 11 TFLOPS, erschienen im August 2016 [L11]

¹²Der Intel Core i7-4790T ist ein Prozessor, Preis derzeit 303 USD, Rechenleistung 27 GFLOPS, erschienen im zweiten Quartal 2014 [L12]

¹³Eine Bounding Box verweist auf einen rechteckigen Bildausschnitt

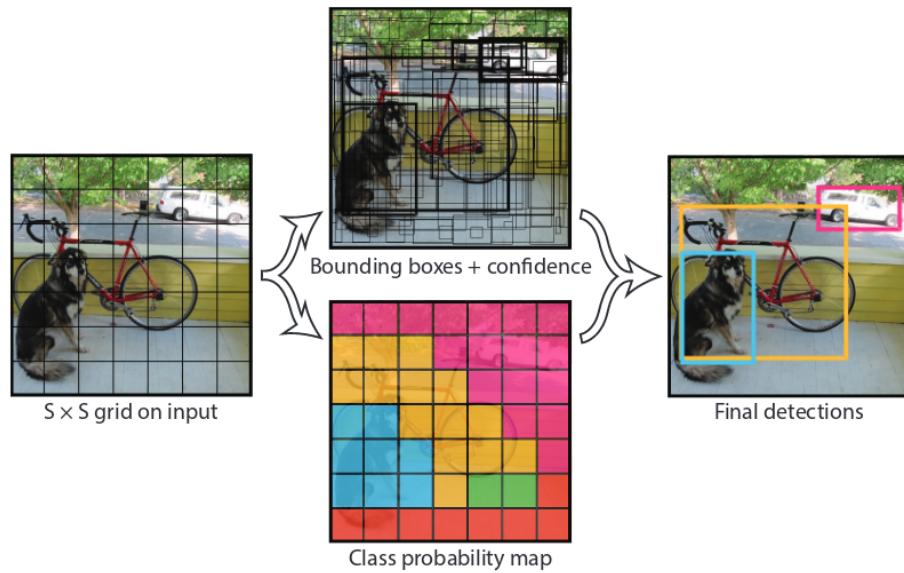


Bild 2.7: Beispielhafter Ablauf einer regionbasierten Objekterkennung mit einem CNN (Quelle: [RDGF16])

2.3.4 Wettbewerbe zur Objektklassifizierung und -erkennung

Für die Objektklassifizierung und -erkennung gibt es seit einigen Jahren weltweite Wettbewerbe. Die beiden größten sind die *ImageNet Large Scale Visual Recognition Challenge*¹⁴, kurz LSVRC und die *PASCAL Visual Object Classes Challenge*¹⁵, kurz VOC. Die Ziele der Wettbewerbe werden je nach Wettbewerbsklasse definiert. Standardmäßig wird eine möglichst hohe mittlere Genauigkeit (engl. *accuracy*), beziehungsweise Trefferquote, bei einer möglichst geringen Fehlerquote verlangt. Die Echtzeitfähigkeit wird meist nicht gewertet [NT15].

Bei der LSVRC steht eine Datenbank von 50.000 Bildern mit Label von 1000 verschiedenen Klassen für das Training der Netzwerke zur Verfügung. Das Netzwerk bekommt anschließend 1.2 Millionen noch nicht verwendete Bilder und muss je nach Wettbewerbsklasse entscheiden, was (Objektklassifizierung) und eventuell auch wo (Objektlokalisierung) etwas zu sehen ist. Das Ergebnis wird nach den beiden zuvor genannten Kriterien bewertet [NT15].

Ähnliche Kategorien werden bei der VOC angeboten, allerdings auf 20 Klassen (siehe Anhang A.1) beschränkt.

Die Wettbewerbe sind für diese Studienarbeit indirekt relevant. Zahlreiche Wissenschaftler verwenden aktuell die zuvor genannten Datensätze der Wettbewerbe zur Leistungsbewertung ihrer Implementierungen bezüglich der Objektlokalisierung. Besonders hilfreich für die Arbeit

¹⁴Umfangreiche Informationen zur LSVRC unter [L13]

¹⁵Umfangreiche Informationen zur VOC sind unter [L14] zu finden

ist dies, da die Implementierungen hierbei auch hinsichtlich der Schnelligkeit [vgl. Anforderungen in Abschnitt 1.4), anhand eines Referenzsystems, bewertet werden. Als Referenz wird die bereits erwähnte Nvidia TitanX als Recheneinheit und der VOC-Datensatz von 2007 für das Training verwendet. Diese Implementierungen sind frei verfügbar und werden bereits trainiert angeboten. Selbstverständlich können auch andere beziehungsweise weitere Daten für das Training verwendet werden (vgl. Anforderungen).

2.3.5 Performance aktueller Objektlokalisierungen

Was aktuell (Stand 12/2016) in der Objekterkennung möglich ist zeigen die Abbildungen in Bild 2.8. Dieses CNN mit der Bezeichnung YOLO9000 kombiniert über einen Entscheidungsbaum (Bild 2.9) zwei Datensätze: Den LSVRC-Datensatz und den COCO-Datensatz (*Common Objects in Context* [L15], Objektklassen im Anhang A.2). 9000 Klassen können regionbasiert bei 67 FPS¹⁶ mit einer NVIDIA TitanX erkannt werden¹⁷.

Leider ist bei dieser hohen Anzahl an Klassen die Genauigkeit (18 %) der Klassenzuordnung

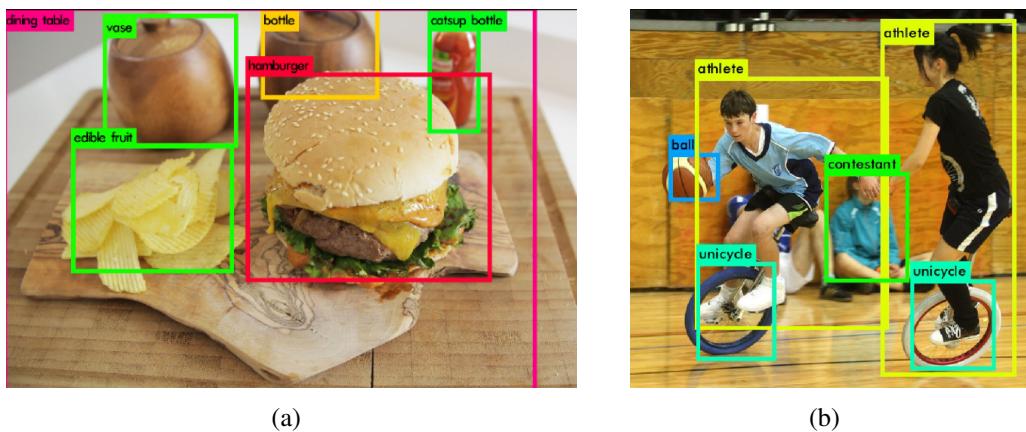


Bild 2.8: YOLO9000: Echtzeit state-of-the-art Deep Learning Objekterkennung von 9000 Klassen (Quelle: [RF16])

noch steigerbar. Von den gleichen Wissenschaftlern wurde dieses CNN ohne Entscheidungsbaum mit dem VOC07-Datensatz trainiert und erreicht mit diesem eine mittlere Genauigkeit von 76,8 %. Für die Studienarbeit relevante Implementierungen werden in Abschnitt 3.1, basierend

¹⁶Frames per Second: entspricht der Bildfrequenz

¹⁷Getestet in [RF16]

auf einem Vergleich zwischen der Leistungsfähigkeit unter dem genannten Referenzsystems, analysiert.

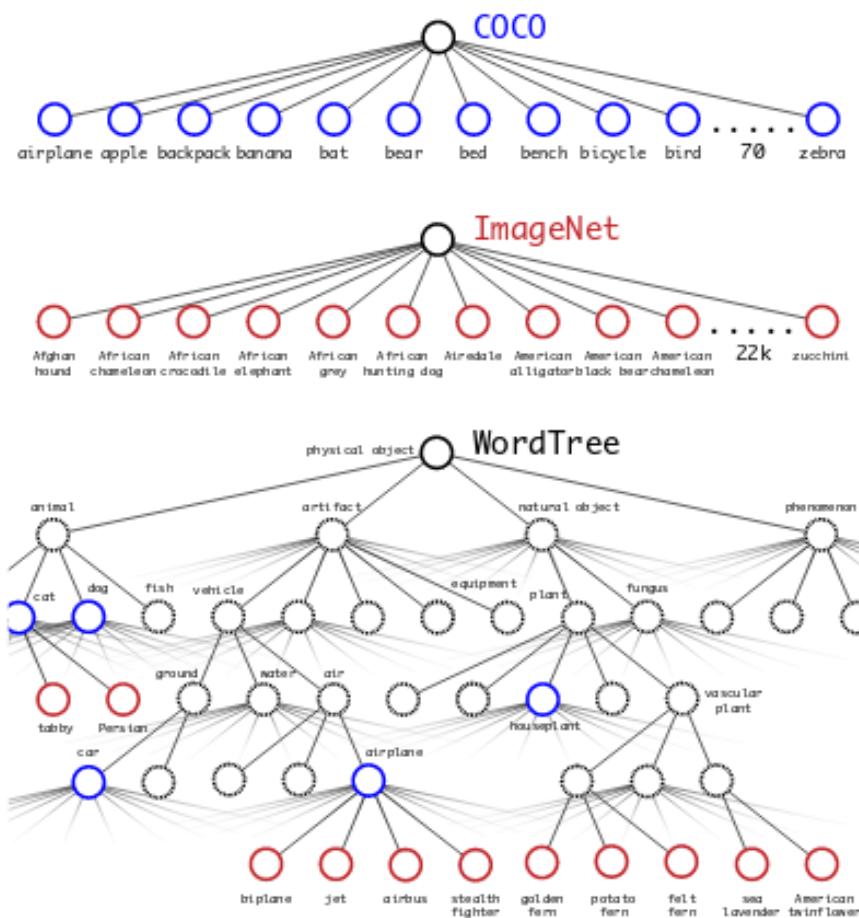


Bild 2.9: Entscheidungsbaum zur Kombination von Datensätzen (Quelle: [RF16])

2.4 RGB-D Kartierung

In diesem Abschnitt werden zuerst die Grundlagen von RGB-D-Kameras und danach die Grundlagen zur Kartierung mit diesem Kameratyp beschrieben. Das Ziel ist die Erstellung einer dreidimensionalen Umgebungskarte mit Farbinformationen.

2.4.1 RGB-D-Kamera

Herkömmliche digitale Farbkameras bilden einen Objektpunkt auf einem Flächensor als Bildpunkt mit roter, grüner und blauer Farbinformation ab. Das Lockkameramodell wird zur Berechnung herangezogen. Die Entfernung zwischen Kamera und Objektpunkt kann nicht erfasst werden.

Ein RGB-D-Kamera (*Red, Green, Blue - Depth*) ist ein Kamerasystem, das den Objektpunkt aus einer weiteren Positionen aufnimmt und dann über Triangulation die Tiefeninformation berechnet und dem Bildpunkt zuordnet. Derzeitige kostengünstige RGB-D-Kameras, wie die Intel Realsense SR300 des youBots, messen hierbei mit einer Infrarotkamera die Richtung zu einem Objektpunkt anhand eines Musters (siehe Bild 2.10). Das Muster wird von einer aktiven strukturellen Lichtquelle erzeugt [Jäh05]. Die SR300 verwendet als Lichtquelle einen Infrarotprojektor der versetzt zur Infrarotkamera angeordnet ist und phasenverschobene Streifen auf das Sichtfeld der Farbkamera projektiert [2.2]. An Objekten werden die Muster reflektiert und von der Infrarotkamera erfasst.

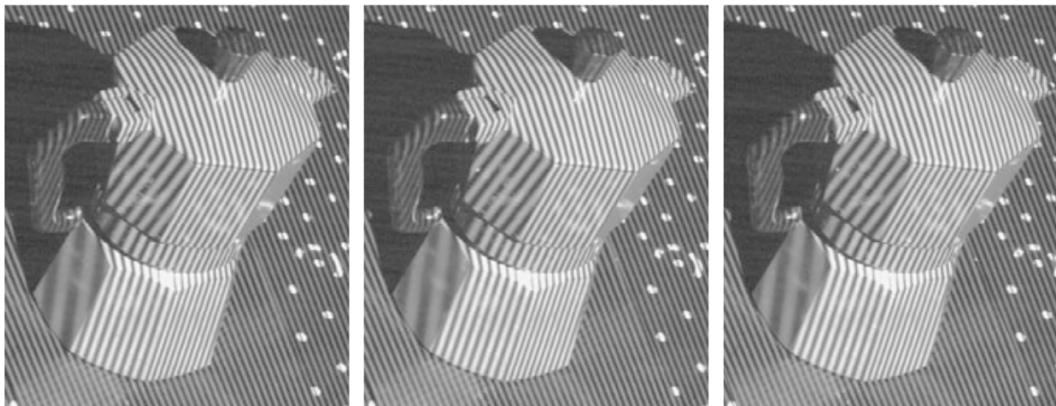


Bild 2.10: Aktive Triangulation durch phasenverschobene Streifenbilder. Abgebildet ist das Muster einer Phasenverschiebung von 0, 90 und 180 Grad (Quelle: [Jäh05])

Alle drei Sensoren sind parallel ausgerichtet. Der Infrarotprojektor und die Infrarotkamera sind jeweils außen und die RGB-Kamera mittig angeordnet.

Die Sensoren haben eine feste Brennweite und einen festen Fokus. Aus den Daten der RGB-Kamera wird ein Farbbild und aus den Daten der Infrarotkamera und -projektor ein Tiefenbild erzeugt. Über eine kamerainterne Software werden diese Daten fusioniert. Die Summe der erfolgreich zugeordneten Punkte wird als Punktwolke (engl. *Point Cloud*) bezeichnet. Somit erhält jeder Punkt sechs Parameter: einen Rot-, Grün- und Blauwert sowie die den Abstand zur Kamera in X-, Y- und Z-Richtung.

Die Ermittlung der Tiefe über das Infrarotmuster bedingt einige charakteristische Eigenschaften. Schwarze Objektpunkte absorbieren Infrarotstrahlen und können somit von der Infrarotkamera nicht erfasst werden. Transparente Objektpunkte und UV-Strahlung erschweren ferner die Zuordnung von Objektpunkten [Jäh05].

2.4.2 Kartierung

Für die Perzeption der Umgebung benötigen mobile Roboter eine Karte. Die Erstellung einer Karte wird über die kontinuierliche Aufnahme von Umgebungsinformation und Schätzung der Lage des mobilen Roboters erzielt, bekannt als *Simultaneous Mapping And Localization* (kurz SLAM).

Die Umgebungsinformationen können über Sensordaten, beispielsweise mit einer RGB-D-Kameras (3D) oder Laserscanner (2D) kontinuierlich aufgenommen werden (vgl. Abschnitt 2.4.1). Eine Transformation zwischen Sensor- und Umweltkoordinatensystem muss berechnet werden, um die Lageänderung des Roboters in der Karte zu berücksichtigen. Drei Translations- und drei Rotationsparameter sind hierfür nötig. Diese sechs Parameter können über die Verkettung einzelner Transformationen bestimmt werden und müssen für jede Lageänderung (*simultaneos localization*) des Roboters neu berechnet werden. Für die Berechnung der Transformation können die kinematischen Zusammenhänge zwischen Sensor und Roboterbasis, sowie die Lage des Roboters herangezogen werden. Alternativ kann die Transformation direkt über die Sensordaten anhand von optischen Merkmalen der Umgebung geschätzt werden. Jedes Verfahren beruht auf einer Schätzung. Generell gilt, durch Berücksichtigung zusätzlicher Informationen weiterer Sensoren, beispielsweise einer zweiten RGB-D-Kamera, kann die Genauigkeit gesteigert werden [End15].

Für Bewegungen mobiler Roboter ist weiterhin von Interesse, welche Regionen belegt sind und welche nicht. Hierzu wird ein zwei- oder dreidimensionales Belegungsgitter (engl. *occupancy grid*) herangezogen. Das Belegungsgitter verwendet die Informationen der Umgebungskarte und unterteilt hierbei in belegte, freie und unbekannte Felder [Ort16]. Aus der Kartierung mit einer RGB-D-Kamera kann somit aus der Punktwolke ein dreidimensionales Raumgitter mit Farbinformationen erzeugt werden. Diese Kartenart wird als *Octomap* bezeichnet, siehe Bild 2.11.



Bild 2.11: *Octomap* eines Arbeitsplatzes, erzeugt mit einer RGB-D-Kamera (Quelle: [End15])

Moderne Methoden zur Kartierung besitzen weiterhin die Fähigkeit, die Pixelpositionen in der Karte nachträglich automatisch zu aktualisieren. Nötig wird dies wenn beispielsweise nach einem 360 Grad Schwenk der Kamera die Pixelpositionen identischer Objektpunkte durch Messfehler abweichen. Hier spricht man von einer *loop-closure*-Erkennung, siehe [End15].

3 Entwurf

In diesem Kapitel wird ein Entwurf für die kombinierte Objektlokalisierung und Kartierung am Beispiel des youBots unter Verwendung vorhandener Ressourcen erarbeitet. Basierend auf den notwendigen Anforderungen (Abschnitt 1.4) werden Open-Source-Implementierungen vorgestellt, die zur Erfüllung der Aufgabenstellung beitragen können. Anhand eines Vergleichs werden Kennzahlen von relevanten Implementierungen gegenübergestellt. Darauf aufbauend wird ein Konzept erarbeitet, welches im anschließenden Kapitel 4 mit eigenem Code unter C++ vervollständigt wird.

3.1 Evaluation bestehender Ansätze

Eine vorhandene Implementierung zur kombinierten Objektlokalisierung und Kartierung kann zum derzeitigen Zeitpunkt (11/2016) nicht gefunden werden. In [TL16] werden Arbeiten beschrieben, die Teilaspekte aufgreifen. Die Wiedererkennung einer Umgebung mit Hilfe eines CNNs wird beispielsweise von von Arandjelovic u.a. [AGT⁺15] behandelt. Weiterhin werden in [TL16] Ansätze beschrieben, die Objekte mit einem CNN anhand der dreidimensionalen Daten erkennen. Hierzu gehört der zuvor beschriebene Ansatz von Eitel u.a. [ESRB15]. Hierbei kann die Objektposition in der Umgebung direkt ermittelt werden. Beide genannten Teilaspekte sind prinzipiell auf diese Studienarbeit übertragbar, jedoch widersprechen sie der Echtzeitanforderung aus Abschnitt 1.4.

Daher werden in diesem Abschnitt Teilaspekte vorgestellt und verglichen, die zur Lösung der Problemstellung (Abschnitt 1.2) unter Berücksichtigung aller Anforderungen (Abschnitt 1.4) wiederverwendet werden können. Die Objekterkennung mit einem Deep-Learning-Netzwerk (vgl. Abschnitt 2.3) und die Kartierung mit einer RGB-D-Kamera (vgl. Abschnitt 2.4) stellen hierbei die zwei größten Bausteine dar.

Zu beiden Teilaspekten existieren im Internet eine große Anzahl an Open-Source-Implementierungen. Diese werden in diesem Abschnitt jeweils in einer Gegenüberstellung der Leistungsmerkmale verglichen. Die Schnittstelle zwischen den beiden Bausteinen wird aufgrund des vergleichsweisen geringen Umfangs vorerst nicht berücksichtigt.

3.1.1 Vergleich von bestehenden Objekterkennungen

In Tabelle 3.1 ist das Top 5 Ergebnis einer umfangreichen Recherche zu echtzeitfähigen Objektlokalisierungen durch regionbasierte DNN. Alle Projekte greifen auf die Faltungsarchitektur der CNN zurück (vgl. Abschnitt 2.3). Die Kennzahlen aus Tabelle 3.1 wurden von den Dokumentationen der Implementierungen zusammengetragen, siehe Quellenangabe. Alle Projekte haben als Input ausschließlich die RGB-Bildsequenz. Echtzeitfähige Projekte zur Objektlokalisierung mittels Deep Learning und RGB-D Input konnten nicht gefunden werden.

An den Kennzahlen in Tabelle 3.1 ist zu erkennen, dass die Input-Auflösungen niedrig sind. Für das SSD- und das YOLO-V2-Package sind zusätzliche Kennzahlen aus einer höheren Eingabeauflösung angegeben. Die Bildfrequenz sinkt hierbei deutlich, während sich die mittlere Genauigkeit nur gering verbessert.

Weiterhin ist in Tabelle 3.1 an den Kennzahlen von YOLO-V1 und Fast-YOLO ein Trade-off zwischen mittlerer Genauigkeit und Bildrate zu erkennen. Die Netzwerkgröße wurde bei Fast-YOLO reduziert, um den Rechenaufwand zu reduzieren beziehungsweise, um die Bildfrequenz zu erhöhen (vgl. [RDGF16]).

Die Anzahl der Objektklassen ergibt sich aus dem jeweiligen Trainingsdatensatz (Vgl. VOC- und COCO-Objektklassennamen im Anhang A). Wie bereits in Abschnitt 2.3 beschrieben, können auch andere Datensätze mit speziellen Objektklassen über einen umfangreichen Lernprozess angelernt werden.

Die Performance der Packages wird in Tabelle 3.1 ebenfalls verglichen. Da der identische Trainingsdatensatz und die identische Recheneinheit verwendet wird, sind die Ergebnisse somit auch vergleichbar. Diese Referenz ist in den Quellen zu finden. Die Netzwerke wurden hierbei im Voraus mit dem VOC-Datensatz von 2007 trainiert. Die während der Objekterkennung erreichte Bildfrequenz (FPS) bezieht sich auf die Rechenleistung einer Nvidia TitanX. Somit sind die Ergebnisse vergleichbar. Die mittlere Genauigkeit (mAP) wird anhand der 20 Klassen des VOC-Datensatzes (Tabelle A.2) ermittelt, indem zunächst die mAP der einzelnen Objektklassen (siehe Quellen) und anschließend deren Mittelwert berechnet wird. Basierend auf diesen Erkenntnissen wird im Abschnitt 3.2 (Konzepterstellung) ein Package für die Objektlokalisierung ausgewählt.

¹Klassenname mit Angabe der Wahrscheinlichkeit der Zuordnung

²Zweidimensionale, rechteckige Bounding Box

³VGG16 ist eine 16-lagige CNN-Architektur von Karen Simonyan und Andrew Zisserman (2014). Einzelheiten unter [SZ15]

⁴ZF ist eine CNN-Architektur von Matthew D. Zeiler und Rob Fergus (2013). Einzelheiten unter [ZF13]

⁵*middle Accuracy Percentage*

Tabelle 3.1: Echtzeitfähige state-of-the-art (12/2016) Objekterkennungen durch *Convolutional Neural Networks* (regionbasiert)

Bezeichnung:	SSD	YOLO V1	YOLO V2	Fast YOLO	Faster R-CNN
Inputpixel:	300x300 (512x512)	448x448	448x448 (512x512)	448x448	1000x600
Inputsignal:	RGB	RGB	RGB	RGB	RGB
Output:	Class ¹ , BB ₂	BB, Class	BB, Class	BB, Class	BB, Class
Basisarchitektur:	VGG16 ³	VGG16	VGG16	VGG16	ZF ⁴
ROS-Anbindung:	nein	ja	ja	ja	nein
Trainingsdaten:	ILSVRC	VOC07 + VOC12	COCO	VOC07 + VOC12	VOC07 + VOC12
Objektklassen:	1000	20	80	20	20
Performancetest					
mAP ⁵ VOC07 Trainingsdatensatz:	74.30 % (76.80 %)	66.40 %	76.80 % (78.60 %)	52.70 %	62.10 %
FPS mit TitanX:	58 FPS (22 FPS)	45 FPS	67 FPS (40 FPS)	155 FPS	17 FPS
Quelle:	[LAE ⁺ 16]	[RDGF16]	[RF16]	[RDGF16]	[RHGS16]

3.1.2 Vergleich von bestehenden RGB-D-Kartierungen

Folgend werden echtzeitfähige Packages für die RGB-D Kartierung verglichen. In Tabelle 3.2 sind die Top-5 Ergebnisse einer umfangreichen Recherche zu sehen. Die Performance der Packages wird anhand der mittlere quadratisch-translatorischen Abweichung (RMSE⁶) und der Schnelligkeit (FPS) verglichen.

Ähnlich wie bei den CNN-Packages, gibt es auch bei RGB-D-Kartierungen Referenzdatensätze, an denen die Performance in den Dokumentationen der Packages verglichen wird. Allerdings gilt dies leider nicht für die Recheneinheit. Während einige Packages lediglich über eine CPU ausgeführt werden, verwenden andere Packages zusätzlich eine GPU. Somit sind die in Tabelle 3.2 angegebenen FPS-Werte nicht direkt vergleichbar, eine Tendenz kann jedoch unter Betrach-

⁶RMSE steht für *Root Mean Square Deviation*

tung der verwendeten Recheneinheit/en und Datensätze abgeleitet werden (vgl. Fußnoten in Tabelle 3.2).

Die angegebenen RMSE-Werte in Tabelle 3.2 beziehen sich auf RGB-D-Datensätze [L17] der TU München. Diese Datensätze wurden gewählt, da in [MAT16] die Packages hinsichtlich der Genauigkeit gegenübergestellt sind und somit einen direkten Vergleich zulassen. Hierzu wurde der arithmetische Mittelwert der RMSE-Werte aus Tabelle III in [MAT16] berechnet.⁷

In Hinblick auf die Umsetzung unter ROS wird in Tabelle 3.2 weiterhin angegeben ob eine Schnittstelle zu ROS besteht. Durch eine vorhandene Anbindung an ROS reduziert sich der Implementierungsaufwand für diese Arbeit.

Ebenfalls ist vermerkt ob ein Belegungsgitter (engl. *occupancy grid*) aus den Rohdaten erzeugt werden kann. Das Belegungsgitter ist unter anderem für die Pfadplanung, Navigation und Manipulation von Vorteil. Hierbei ist vermerkt, welche Regionen der Karte belegt und welche frei sind.

Zur Erfüllung der Aufgabenstellung dieser Studienarbeit ist ein Belegungsgitter nicht erforderlich, für Folgearbeiten hingegen schon. Für weitere Echtzeitapplikationen, die während der Kartierung ausgeführt werden sollen, ist dieses Features jedoch unerlässlich.

Tabelle 3.2: Echtzeitfähige state-of-the-art (12/2016) RGB-D-Kartierungen

Bezeichnung:	ORB-SLAM2	Elastic-Fusion	Kintinuous	DVO SLAM	RGBD-SLAM2
Belegungsgitter ⁸	2D	n.v.	n.v.	n.v.	3D
ROS-Anbindung:	ja	nein	nein	nein	ja
Unterstützte Recheneinheit:	CPU + GPU	CPU	CPU + GPU	CPU	CPU + GPU
Schnelligkeit:	10-20 FPS ⁹	22 FPS ¹⁰	15 FPS ¹¹	7,5 FPS ¹²	10-35 FPS ¹³
RMSE	18 mm	36 mm	44 mm	28 mm	56 mm
Quelle:	[MAT16]	[WSMG ⁺ 15]	[WKF ⁺ 12]	[KSC13]	[End15]

⁷Weitere RMSE-Kennzahlen sind in [KSC13] zu finden

⁸engl. *occupancy grid*

⁹Intel Core i7-4790, KITTI Datensatz 10 FPS, EuRoC Datensatz 20 FPS

¹⁰Intel Core i7-4930K und Nvidia GeForce GTX 780 Ti, ICL-NUIM Datensatz

¹¹Intel Core i7-2600 und Nvidia GeForce GTX 560 Ti, Datensatz siehe Quelle

¹²Intel Core i7-2600, TUM Datensatz

¹³Intel Core i7-2600, TUM Datensätze

Die vorgestellten Packages zur Objekterkennung (Tabelle 3.1) und zur Kartierung (Tabelle 3.2) werden im folgenden Abschnitt für die Konzepterstellung zur Auswahl einbezogen.

3.2 Konzepterstellung

In diesem Abschnitt wird ein Konzept, basierend auf einer modularen Struktur des Robot Operating System (Abschnitt 2.2), erarbeitet. Drei Packages - eines für die Implementierung der Objekterkennung mit einem CNN (Packagename: Objekterkennung), eines für die Implementierung der RGB-D-Kartierung (Packagename: Kartierung) und eines, um diese beiden Packages sinnvoll zu verbinden (Packagename: Segmentierung) - werden benötigt. Die Packages können über die in Abschnitt 2.2 vorgestellten Methoden von ROS kommunizieren.

Aus Abschnitt 2.3 geht hervor, dass bei künstlichen neuronalen Strukturen sehr viele Rechenoperationen durchgeführt werden müssen. Der youBot besitzt als Recheneinheit ausschließlich eine CPU (Intel Core i7-4790T). Diese stellt im Vergleich zum Referenzsystem aus Tabelle 3.1 mit einer Nvidia TitanX wenig Rechenleistung zu Verfügung. Vortests haben gezeigt, dass die exemplarische Implementierung des YOLO-V1-Packages ([RDGF16]) auf der CPU des youBots über 6 Sekunden pro Bild für die Berechnung benötigt. Parallel muss die RGB-D-Kartierung ausgeführt werden. Auch diese Implementierung benötigt nach Tabelle 3.2 eine hohe Rechenkapazität.

Die Rechenkapazität des youBots ist daher keinesfalls für die Echtzeitfähigkeit ausreichend, zusätzliche Rechenkapazität wird benötigt. Diese zusätzliche Rechenkapazität durch den youBot bereitzustellen widersprechen folgende drei Probleme:

1. Die Hardware mobiler Roboter, wie die des youBots, ist energieeffizient ausgelegt, sodass eine lange Akkulaufzeit zustande kommt¹⁴.
2. Bauraumbedingt kann keine leistungsstarke Grafikkarte, wie die Nvidia TitanX auf dem Mainboard montiert werden. Ein schnellerer Prozessor kann die benötigte Leistung derzeit nicht wirtschaftlich bereitstellen¹⁵.
3. Derzeitige mobile Grafikprozessoren, wie die Tegra-Serie¹⁶ von Nvidia, können nicht annähernd die Leistung einer TitanX zu Verfügung stellen.

¹⁴Der Stromverbrauch der Recheneinheit steigt ungefähr proportional zu den durchgeführten Rechenoperationen an.

¹⁵siehe Vergleich zwischen Prozessor und Grafikkarte in Abschnitt 2.3

¹⁶weiterführende Informationen unter [L18]

Die Lösung dieser Problemstellung wird über eine ausgelagerte Berechnung des CNN erzielt. Hierbei werden die Daten über Funk an eine externe Recheneinheit gesendet. Die Objekterkennung kann somit nur bei bestehender Funkverbindung durchgeführt werden. Dennoch ist die Objekterkennung nicht zwingend für den Betrieb eines mobilen Roboters notwendig, sodass Funkstörungen zu keiner Systeminstabilität führen können. Für die ausgelagerte Berechnung des CNN können die drei zuvor genannten Probleme umgangen werden. Aus Abschnitt 3.1 geht hervor, dass es zum derzeitigen Zeitpunkt noch keine mobilen, echtzeitfähigen Ansätze für die dreidimensionale Objekterkennung mittels CNN gibt. Daher ist es ausreichend die RGB-Bildsequenz vom youBot auf die externe Recheneinheit und die Ausgabe des CNN von der externen Recheneinheit zurück an den youBot zu senden. Der youBot kann über die in Abschnitt 2.1 vorgestellte WLAN-Schnittstelle kommunizieren. Die WLAN-Schnittstelle ermöglicht das Übertragen von Daten mit einer Geschwindigkeit von 300 Mbit/s.

Im Folgenden wird die benötigte Datenrate abgeschätzt, um zu überprüfen ob die Funkübertragung mittels WLAN realisierbar ist. Nach Abschnitt 3.1, beziehungsweise Tabelle 3.1, wird für ein CNN eine hohe Bildfrequenz bei geringer Auflösung verlangt. Aus dem Datenblatt der Kamera [L19] wird als geeignetes Datenformat des RGB-Sensors eine Auflösung von 640x480 (VGA, 4:3), sowie die höchste Bildfrequenz von 60 Hertz ausgewählt. Die Bildfrequenz muss nicht zwingend so hoch sein, um die Echtzeitanforderung aus Abschnitt 1.4 zu erfüllen. Mit folgender Beziehung kann die Datenrate für die RGB-Bildsequenz überschlagen werden:

$$\text{Auflösung} * \text{Farbtiefe} * \text{Bildfrequenz} = \text{Bits pro Sekunde}$$

Die Farbtiefe von digitalen RGB-Bildern beträgt 8 Bit pro Farbkanal. Bei der Übertragung von Daten unter ROS werden diese encodiert, beziehungsweise komprimiert versendet. Bei der Übertragung von Farbbildern reduziert sich hierbei die Datenmenge von 24 Bit auf 16 Bit pro Pixel. Die Datenrate, die vom youBot auf die externe Recheneinheit übertragen werden muss, beträgt somit bei 60 Bildern pro Sekunde mit einer Auflösung von 640x480 Pixeln ungefähr 295 Mbit/s.

Die Datenrate, die von der externen Recheneinheit auf den youBot übertragen werden muss, kann vernachlässigt werden, da lediglich wenige Parameter (siehe Abschnitt 4.1) zur Beschreibung jeder Bounding Box übertragen werden.

Das Verhältnis von maximaler Datenrate der WLAN-Schnittstelle zu Datenrate bei 60 Hertz ist 1.02 und entspricht somit dem Sicherheitsfaktor der Übertragung. Dieser Sicherheitsfaktor ist leider gering. Dennoch sollte die Übertragungsleistung ausreichend sein, um die folgende Operationen in Echtzeit zu ermöglichen. Das CNN kann somit ausgelagert werden. Schematisch ist die Auslagerung des Packages Objekterkennung in Bild 3.1 visualisiert.

Folgend werden die drei benötigten Packages (siehe Bild 3.1) hinsichtlich Ihrer Funktionalität beschrieben. Für das Package Objektlokalisierung kommen prinzipiell alle Packages aus

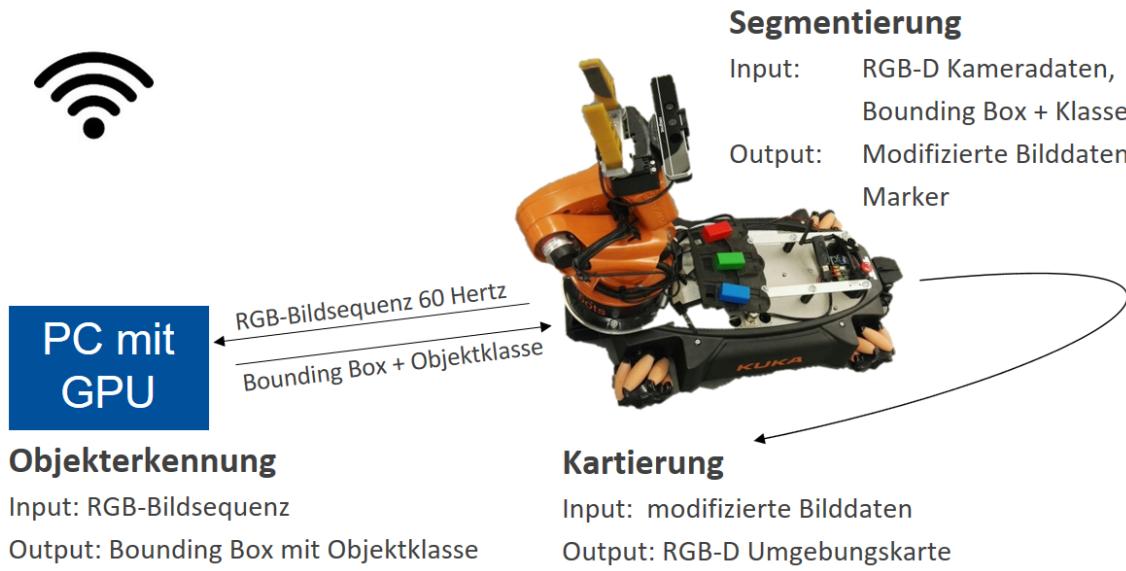


Bild 3.1: Konzept für die kombinierte Kartierung und Objektlokalisierung mit dem youBot bei ausgelagerter Berechnung des CNN

Tabelle 3.1 für die ausgelagerte Objektlokalisierung in Frage. Hierbei wurde die Performance anhand der Rechenleistung einer Nvidia TitanX bewertet. Diese Grafikkarte ist für ihre hohe Rechenleistung bekannt und kann für diese Studienarbeit verwendet werden. Die zu erwartenden Ergebnisse sind, durch die Verwendung der gleichen Recheneinheit, bis zu einer Bildfrequenz von 60 Hertz identisch zu denen in Tabelle 3.1. Darüber wird die Verarbeitung durch die Kamerafrequenz und die Bandbreite der Übertragung limitiert.

Die maximal mögliche Bildfrequenz ist somit 60 Hz. Diese Bildfrequenz sollte angestrebt werden, um die Echtzeitfähigkeit in jeder Situation zu gewährleisten.

3.2.1 Konzept der Objekterkennung

Unter Beachtung dieser Voraussetzungen und den in Abschnitt 1.4 beschriebenen Anforderungen wird das YOLO-V2 Package ausgewählt. YOLO-V2 kann im Vergleich zu den anderen Packages in Tabelle 3.1 mit einem mAP von 76,8 % am VOC-Datensatz von 2007 die höchste Genauigkeit erzielen. Trainiert mit dem COCO-Datensatz, kann YOLO-V2 einschließlich einer Schnittstelle zu ROS im Internet heruntergeladen werden. 80 Standardklassen können durch die Trainingsdaten des COCO-Datensatzes erkannt werden. Für erste Umsetzungen ist dies ausreichend. Alternativ trainierte Netzwerke von YOLO-V2 sind unter [L20] abrufbar. Somit können auch andere Objektklassen hinterlegt werden (vgl. Anforderung (3)).

Die zu erwartende Ausgabefrequenz von YOLO-V2 ist über 60 Hertz, sodass die maximale

Kamerafrequenz ausgenutzt werden kann. Die Input Bildauflösung von YOLO-V2 ist in X- und in Y-Richtung kleiner als die VGA-Auflösung der RGB-Kamera (vgl. Tabelle 3.1). YOLO-V2 berechnet aus der RGB-Bildsequenz für jedes erkannte Objekt eine zweidimensionale rechteckige Bounding Box mit Klassenangabe und deren Wahrscheinlichkeit (vgl. Abschnitt 2.3). Um diese Ergebnisse für die Objekterkennung in einer Karte zu nützen, ist eine Transformation in die dritte Dimension nötig.

3.2.2 Konzept der Segmentierung

Hierzu werden die Stereo Daten der Kamera benötigt. Die verwendete Kamera (Intel Realsense SR300) verfügt über einen RGB-Kamerasensor, eine parallel angeordneten Infrarotkamera und einen Infrarotprojektor. Die Kamerasoftware sucht übereinstimmende Objektpunkte zwischen Infrarotprojektor und -kamera. Hieraus wird ein Tiefenbild erzeugt. Dieses Tiefenbild besteht aus Grauwerten, jeder Wert steht hierbei für eine Entfernung zur Kamera. Weiterhin berechnet die Kamerasoftware aus dem Tiefenbild und dem RGB-Bild eine Punktwolke, sodass jeder erfolgreich zugeordnete Objektpunkt sechs Parameter enthält: X-, Y- und Z-Koordinate sowie den Rot-, Grün- und Blau-Wert.

Von Interesse für die Objektklassierung sind allerdings nur Punkte, die vom CNN erkannten Objekt stammen. Das Package Segmentierung hat daher die Aufgabe die Ausgabedaten des Packages Objekterkennung mit Hilfe von Kameradaten so aufzubereiten, dass die Objekte im Raum zugeordnet werden können. Diese relevante Region (engl. *Region of Interest*, kurz ROI) in der Punktwolke der Kamera wird in X- und Y-Richtung durch die Projektion der Bounding Box in Z-Richtung in Form eines Pyramidenstumpfs begrenzt. Innerhalb dieser Region muss sich das erkannte Objekt im Raum befinden. Veranschaulicht ist dies in Bild 3.2. Zu Beachten ist hierbei, dass das CNN lediglich die pixelbasierte Position der Bounding Box in Form von zwei 2D-Punkten ausgibt, folgend als $P_1(X_{min}, Y_{min})$ und $P_2(X_{max}, Y_{max})$ definiert. Die Punktwolke der RGB-D-Kamera verwendet hingegen die später für die Kartierung nötigen metrischen 3D-Werte.

Mögliche weitere Gegenstände, die im Vorder- oder Hintergrund innerhalb des Pyramidenstumpfes liegen, dürfen dem erkannten Objekt nicht zugeordnet werden. Neben der Einschränkung der Bounding Box sind daher weitere Methoden nötig, um das Objekt von der Umgebung zu segmentieren. Hierzu existieren verschiedene Methoden der Bildverarbeitung. In Abschnitt 4.2 wird eine geeignete Methode hierfür herausgearbeitet.

Nach der Zuordnung der Objektpunkte, können die Objekte von der Umgebung getrennt betrachtet werden. Drei Features, basierend auf den zugeordneten Objektpunkten, sollen in dieser Arbeit umgesetzt werden:

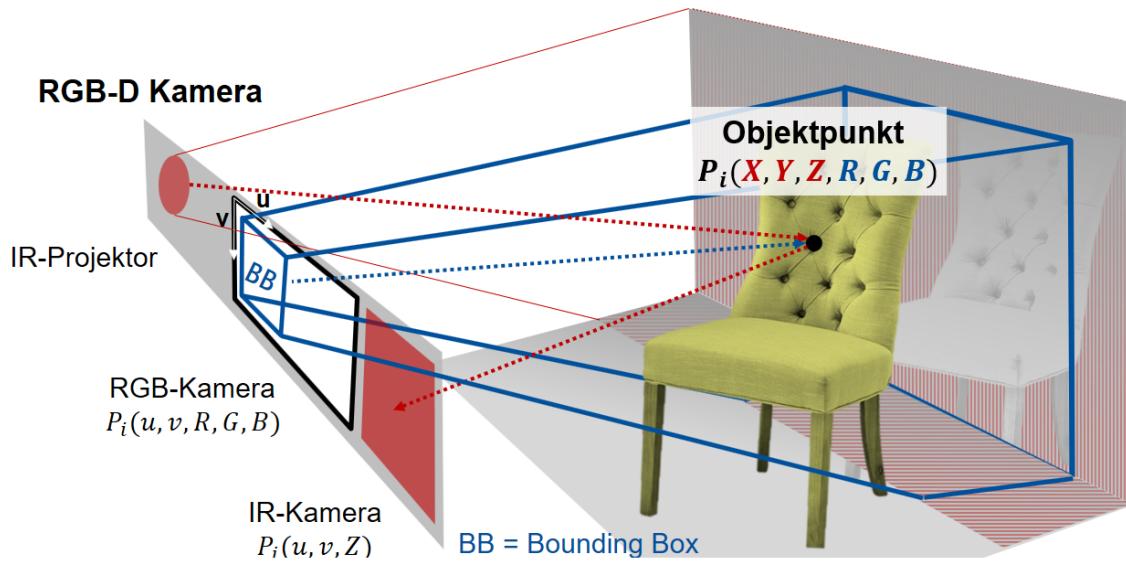


Bild 3.2: Konzept zur Transformation zwischen 2D und 3D

1. Erkannte Objekte sollen in der RGB-D-Kartierung sinnvoll als Marker gekennzeichnet werden.
2. Dynamische Objekte des CNNs sollen in der Kartierung nicht aufgenommen werden. Diese verändern im Laufe der Zeit ihre Position und haben somit einen negativen Einfluss auf die Qualität der Umgebungskarte.
3. Objekte die von der Objekterkennung erfasst wurden, aber außerhalb der Tiefenreichweite der Kamera sind, sollen als zweidimensionale Bounding Box an der maximalen Tiefenreichweite der Kamera in der Karte vermerkt.

Diese Features dienen der intelligenten Umgebungskarte (siehe Anforderung (4) in Abschnitt 1.4) und sollen einen Mehrwert gegenüber einer konventionellen Karte bieten.

3.2.3 Konzept der Kartierung

Für das Package Kartierung kommen prinzipiell alle Packages aus Tabelle 3.2 für die RGB-D-Kartierung in Frage. Um den Implementierungsaufwand gering zu halten, wird ein Package mit vorgefertigter ROS-Anbindung ausgewählt (ORB-SLAM und RGBD-SLAM). Durch die integrierte Octomap und den zu erwartenden höheren FPS-Wert wird das Package RGBD-SLAM-V2 ausgewählt. Dieses Package liefert nach Tabelle 3.2 im Vergleich zu ORB-SLAM etwas ungenauere Ergebnisse, sodass gegebenenfalls das Package abhängig von den Anforderungen der Anwendung nachträglich ausgetauscht werden muss.

RGBDSLAM-V2 und ORBSLAM-V2 benötigen als Input das RGB- und das Tiefenbild der Kamera. Da nur Punkte mit einem Z-Wert ungleich Null in der Kartierung berücksichtigt werden, ist es für die Ausblendung dynamischer Objekte ausreichend den Wert dieser Objektpunkte im Tiefenbild auf Null zu setzen. Somit wird imitiert, dass keine Tiefenzuordnung stattfinden konnte. Um keinen zeitlichen Versatz zwischen Farb- und modifizierten Tiefenbild zu bekommen, müssen beide zeitgleich in das Kartierungspackage geladen werden.

Das Package Kartierung kann daher das Package RGBDSLAM-V2 mit minimaler Modifikationen übernehmen.

Die Umsetzung dieses Konzeptes wird hinsichtlich der Implementierung im folgenden Kapitel beschrieben.

4 Implementierung

In diesem Kapitel wird die Implementierung der kombinierten Kartierung und Objektlokalisierung, die nach dem Konzept aus Abschnitt 3.2 aufbaut ist, knapp beschrieben. Unterteilt ist dieses Kapitel in drei Abschnitte, welche jeweils die Beschreibung der Implementierung eines der in Abschnitt 3.2 vorgestellten Packages - Objekterkennung, Segmentierung und Kartierung - beinhaltet. In Bild 4.1 ist die Struktur der gesamten Implementierung hinsichtlich Nodes und Topics zu sehen (Vgl. Bild 2.2). Veröffentlicht eine Node Messages über ein generiertes Topic, geht von dieser Node ein Pfeil aus. Werden Messages eines Topics von einer Node abonniert, zeigt die Pfeilspitze auf diese Node.

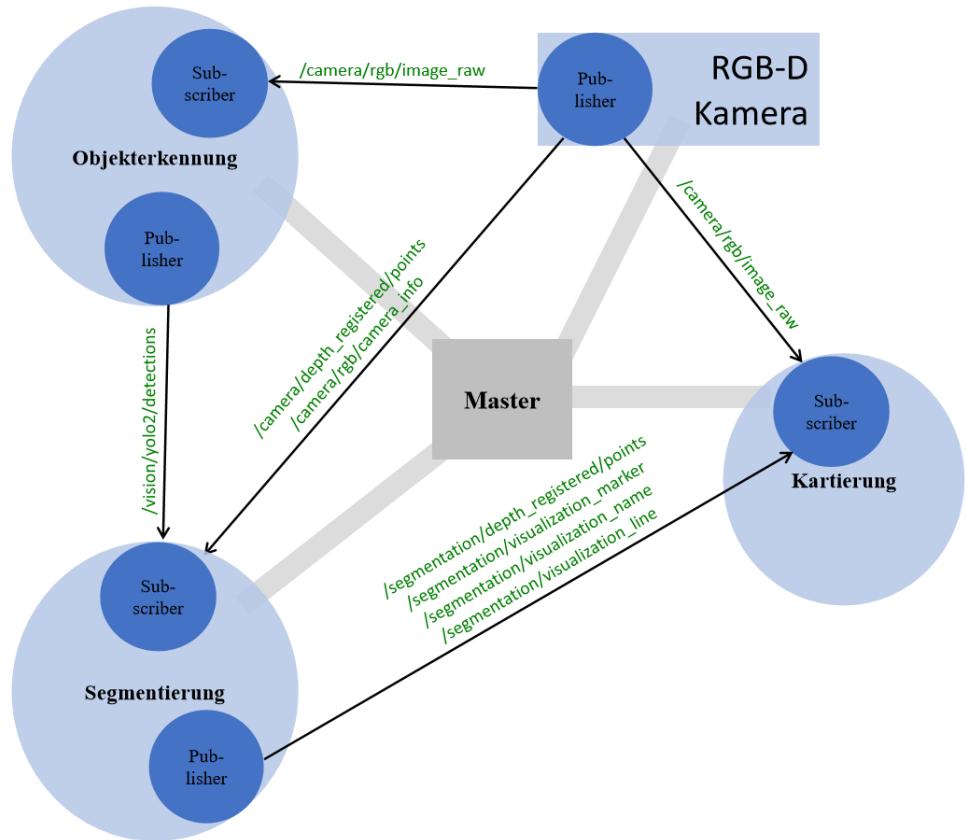


Bild 4.1: Schematische Kommunikation der Nodes (blau) über Topics (grün) unter dem ROS-Master

4.1 Implementierung Package Objekterkennung

Als Basis der Objekterkennung wird das YOLO-V2 Package von Joseph Redmon und Ali Farhadi der Universität Washington verwendet. Eine ausführliche Dokumentation für des Packages kann der Website [L20] entnommen werden. Für die Verwendung unter ROS muss dieses Package erweitert werden, um einen Datenaustausch zwischen den Packages zu ermöglichen. Hierfür wird das Package *ros_yolo2* [L21] verwendet. YOLO-V2 wird in *ros_yolo2* integriert. Eine Kommunikationsmöglichkeit von YOLO-V2 zu anderen Packages wird hierdurch hergestellt. Die Kommunikation von *ros_yolo2* gliedert sich wie folgt (Vgl. Bild 4.1):

Subscriber 1: Die RGB-Bildsequenz der Kamera des youBots wird als Input benötigt. Hierzu wird das Topic */camera/rgb/image_raw* der Kamera abonniert.

Publisher 1: Das Topic */vision/yolo2/detection* veröffentlicht die Ausgabedaten der Objekterkennung.

Listing 4.1: Ausgabedaten der Objekterkennung für jedes Objekt (Quelle [4.2])

```
1 `uint32 class_id` //Class number configured during training
2 `float32 confidence` //Confidence
3 `float32 x` //X position of the objects center relative to the image (
    between 0 = left and 1 = right)
4 `float32 y` //Y position of the objects center relative to the image (
    between 0 = top and 1 = bottom)
5 `float32 width` //Width of the object relative to the image (between 0
    and 1)
6 `float32 height` //Height of the object relative to the image (between
    0 and 1)
```

Über das Topics */vision/yolo2/detection* werden für jedes vom CNN erkannte Objekt die Informationen nach Listing 4.1 veröffentlicht. Der Schwellwert der Zuordnungswahrscheinlichkeit (engl. *confidence*), ab der eine Ausgabe generiert werden soll, kann beim Ausführen der Node über einen Wert zwischen Null und Eins festgelegt werden.

Bei der Ausgabe der Objekt-ID ist zu beachten, dass der Objektklassename durch eine Nummer (0-79) repräsentiert wird. Die Zuordnung hierzu und die Weiterverarbeitung dieser Daten findet im Package Segmentierung statt.

4.2 Implementierung Package Segmentierung

Das Package Segmentierung transformiert die Ausgabedaten des Packages Objekterkennung (siehe Listing 4.1) unter Zuhilfenahme der Kameradaten in die Eingabedaten des Packages Kartierung. Die Funktionsweise kann dem Abschnitt 3.2 entnommen werden. Das Package Segmentierung ist eigenständig unter C++ implementiert. Detaillierte Informationen zur Implementierung können einer detaillierten Beschreibung anhand von Codeausschnitten dem Anhang cha:node entnommen werden. Drei Features sollen nach Abschnitt 3.2 realisiert werden. Die Features greifen auf segmentierte Objektdaten zurück. Gängige Bildverarbeitungsmethoden werden mittels *OpenCV* [L22] (für zweidimensionale Daten) und der *Point Cloud Library* [L23], kurz PCL (für dreidimensionale Daten) für die Segmentierung angewendet. Drei Segmentierungsmethoden wurden implementiert. Um die Auswahl der Segmentierungsmethode zu begründen, werden alle drei knapp vorgestellt und dessen Eigenschaften beschrieben.

- a) *Region Growing*¹ in der Punktwolke der Kamera:

PCL bietet eine exemplarische Implementierung einschließlich Dokumentation für Region Growing in dreidimensionalen Daten mit automatischer Startpunktsetzung an [L24]. Die Nachbarschaftssuche gliedert sich hierbei in mehrere Schritte. Vorweg werden die Normalen aller Datenpunkte in der Punktwolke berechnet.

Der Winkel zwischen der Normalen des aktuellen Datenpunktes (zu Beginn der Startpunkt) und des Nachbardatenpunktes werden verglichen. Sofern der Winkel kleiner als ein definierter Schwellwert (engl. *threshold*) ist, wird der Nachbarpixel zur Region hinzugefügt. Diese Nachbarpixel werden anschließend hinsichtlich ihres Krümmungswertes (engl. *curvature*) untersucht. Ist dieser kleiner als der Schwellwert, wird dieser Nachbarpixel als neuer Suchwert aufgenommen. Es folgt wieder der erste Schritt bis keine Suchwerte mehr existieren, beziehungsweise die Datenmenge segmentiert ist. Der Ablauf lässt bereits erahnen, dass viele Rechenoperationen hierfür nötig sind.

Exemplarisch wurde diese Implementierung auf den RGB-D Kameradaten des youBots angewendet. Der Suchraum wurde auf die Punktwolke innerhalb der Bounding Box der Objekterkennung eingeschränkt, sodass nur die ROI berechnet werden muss. Jeder Datenpunkt in der Punktwolke besitzt hierbei sechs Parameter². Die Positionsparameter jedes Datenpunktes werden als Basis für die Berechnung benötigt. Durch diesen

¹Region Growing ist ein regionbasiertes Verfahren, um Daten zu segmentieren. Eine leere Menge wird definiert, diese Menge repräsentiert die segmentierte Region. Zu Beginn wird ein Startpunkt gesetzt und zur Menge hinzugefügt. Wenn der direkte Nachbarpunkt des aktuellen Punktes (zu Beginn der Startpunkt) ähnliche Merkmale aufweist, wird dieser mit in die Menge aufgenommen, sodass die Menge, beziehungsweise Region wächst (engl. *growing*). Als Merkmal kann beispielsweise bei einem Grauwertbild der Pixelwert verglichen werden. Für weitere Informationen wird auf [Jäh05] verwiesen.

²Drei Parameter für die Position (X, Y, Z) relativ zur Kamera und drei Parameter für die Farbinformation (Rot, Grün, Blau). Siehe Bild 3.2

enormen Rechenaufwand kann diese Segmentierungsmethode leider nicht echtzeitfähig auf dem youBot ausgeführt werden. Eine ausgelagerte Berechnung dieser Segmentierungsmethode, zusätzlich zur Objekterkennung, überschreitet die Bandbreite der WLAN-Schnittstelle (vgl. Sicherheitsfaktor in Abschnitt 3.2). Die Tiefeninformationen der Kamera müssten in diesem Fall zusätzlich in bidirektional übertragen werden. Da diese Segmentierungsmethode somit den Anforderungen aus Abschnitt 1.4 widerspricht, wird diese Methode nicht weiter verfolgt.

b) *Flood Fill*³ im Tiefenbild der Kamera:

Eine Methode mit geringerem Berechnungsaufwand als unter a) beschrieben, wird gesucht. Eine Berechnung im Zweidimensionalen reduziert gegenüber dem Dreidimensionalen grundsätzlich die Datenmenge und somit den Berechnungsaufwand. Daher wird geprüft, ob Flood Fill im Tiefenbild der youBot-Kamera gute Ergebnisse liefern kann. Das Tiefenbild wird repräsentiert durch ein Grauwertbild mit Einträgen zwischen Null⁴ und 256.⁵. Pro Pixel gibt es einen dieser Einträge, sodass die Datenmenge des Tiefenbildes geringer als die einer Punktfolge ist (vgl. Segmentierungsmethode a)). Die Anzahl der Pixel ist äquivalent zur Auflösung (VGA).

OpenCV beinhaltet eine fertig implementierte Flood-Fill-Funktion [L25] mit der, ausgehend von einem manuell gesetzten Startpunkt (engl. *seed point*), eine Region berechnet wird. Der Vergleich zwischen Nachbarpixeln basiert hierbei auf einem Vergleich der Grauwertdifferenz. Die Parameter hierfür können in der Flood-Fill-Funktion gesetzt werden.

Abhängig vom Objekt liefert diese Segmentierungsmethode im Tiefenbild qualitativ unterschiedliche Ergebnisse. Anhand Bild 4.2b ist erkennbar, dass ein Stuhl unter Umständen im Tiefenbild aus mehreren Regionen besteht. Drei Regionen - die Sitzfläche, die Rückenlehne und der Hintergrund - können innerhalb der Bounding Box segmentiert werden. Da lediglich die eine Region dem Objekt zugeordnet wird, kann es sein, dass geometrisch mehrteilige Objekte, wie dieser Stuhl, nur partiell erfasst werden. Selbes trifft auf die Methode unter a) zu. Weiterhin ist das Ergebnis der Flood-Fill-Funktion vom gewählten Startpunkt abhängig. In jedem Fall ist das Ergebnis des Stuhles unvollständig, oder sogar falsch wenn der Hintergrund erfasst wird.

Zusätzliche Bedingungen sind nötig, um die gewünschte Segmentierung zu erhalten. Zu

³Flood Fill ist nahezu identisch zu Region Growing. Der Unterschied besteht darin, dass nicht nur ein Nachbar, sondern alle Nachbarn des aktuellen Datenpunktes verglichen werden. Die Ausbreitungscharakteristik der Region unterscheidet sich hierdurch, das Ergebnis jedoch nicht.

⁴Entfernung zwischen Objektpunkt und Kamera ist gleich Null, oder es konnte keine Tiefenzuordnung stattfinden. Dieser Wert ist im Grauwertbild schwarz visualisiert

⁵Maximal detektierte Entfernung zwischen Objektpunkt und Kamera. Dieser Wert ist im Grauwertbild weiß visualisiert.

beachten ist, dass Bedingungen für alle Objektklassen (global betrachtet) einen Vorteil geben müssen. Eine Definition von objektklassenabhängigen Bedingungen, ist aufgrund der adaptiven Objekterkennung (vgl. Anforderung in Abschnitt 1.4), nicht erwünscht. Diese Segementierung mittels Flood-Fill-Funktion kann echtzeitfähig auf dem youBot implementiert werden und erfüllt somit die Echtzeitanforderung.

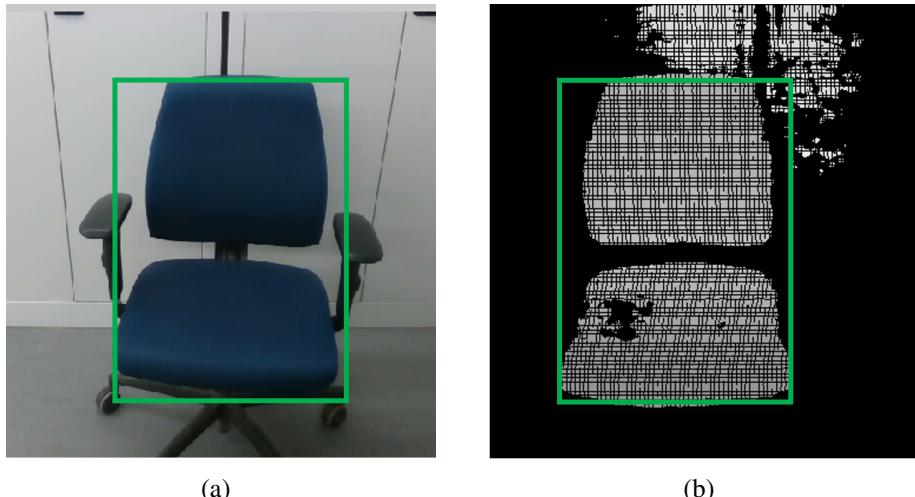


Bild 4.2: Farb- und Tiefenbild eines Drehstuhles

c) Histogramm im Tiefenbild der Kamera:

Eine weitere Methode ist die Segmentierung auf Basis eines eindimensionalen Histogramms des Tiefenbildes im Bereich der Bounding Box. Diese Methode ist nochmals effizienter als Methode b) hinsichtlich der Berechnung. Ein Histogramm repräsentiert die Häufigkeitsverteilung der Pixelwerte. In diesem Histogramm werden zwei Schwellwerte, die das Objekt in Z-Richtung vom Vorder- und Hintergrund trennen sollen, gesucht. Der Suchraum in X- und Y- Richtung wird bereits durch die Bedingungen der Bounding Box eingeschränkt. Somit kann der Suchraum für ein Objekt in allen Dimensionen eingeschränkt werden.

Bild 4.3 zeigt ein Histogramm, welches zum Tiefenbild des Drehstuhles aus Bild 4.2b im Bereich der Bounding Box gehört. Drei Maxima sind zwischen mittlerer und maximaler Kameradistanz zu erkennen. Die Fläche unter Kurve bis zur Abszisse im Bereich des globalen Maximums stammt von der senkrechten Rückenlehne des Drehstuhles. Die Entfernung zur Rückenlehne ist somit am häufigsten im Tiefenbild - Ausschnitt Bounding Box - vertreten. Die Sitzfläche ist durch die horizontale Anordnung an den abgeflachten Verlauf links (durch geringere Distanz) im Histogramm wiederzufinden. Die Häufigkeit

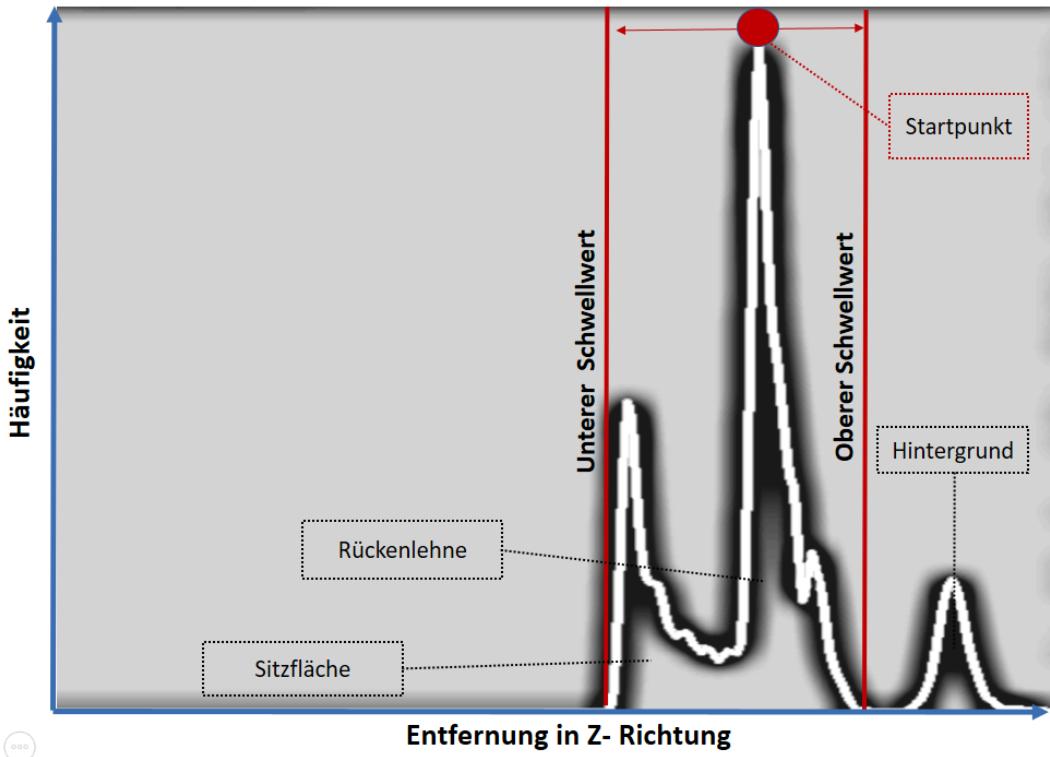


Bild 4.3: Histogramm aus Tiefenbild, Auschnitt Bounding Box Bild 4.2b. Horizontal ist die Entfernung zur Kamera (links nah, rechts fern), vertikal ist die Häufigkeit der Pixel mit identischer Entfernung aufgetragen

der Entfernung zur Wand ist rechts im Histogramm, mit einem Abstand zum Stuhl zu finden. Die Fläche unter der Kurve ist in diesem Bereich nur deshalb so gering, da sich das Histogramm auf die Bounding Box beschränkt.

Der Stuhl nimmt im Tiefenbild (Bild 4.2b) eine große Fläche im Vergleich zur Fläche seiner Bounding Box ein. Im Umkehrschluss bedeutet dies, dass sich das globale Maximum im Histogramm bei Objekten mit einem geringeren Verhältnis von Objektfläche im Tiefenbild zu Volumen der Bounding Box verschieben könnte (Bsp. Stuhl- und Tischbeine). Die eingesetzte Objekterkennung gibt solche filigranen Strukturen nicht zurück, sodass mit dieser Definition der Schwellwerte gute Ergebnisse erzielt werden können.

Jede Segmentierungsmethode hat somit ihre Stärken und Schwächen. Die Segmentierung über die Schwellwerte des Histogramms wird ausgewählt, da die Echtzeitanforderung erfüllt wird, die Berechnung den geringsten Rechenaufwand benötigt und die Ergebnisse gut sind.

Im Package Segmentierung wird die Segmentierungsmethode in der *segmentation_node* umgesetzt. Die Kommunikation zu anderen Packages gliedert sich wie folgt (vgl. Bild 4.1):

- Subscriber 1: Messages des Topics `/vision/yolo2/detection` werden abonniert, um die Parameter der Objekterkennung zu erhalten (siehe Listing 4.1).
- Subscriber 2: Die Punktwolke der Kamera wird durch auf dem Topic `/camera/depth_registered/points` abonniert, um die Objekte anhand der Bounding Boxen zu segmentieren.
- Publisher 1: Die modifizierte Punktwolke (ohne dynamische Objekte) wird durch auf dem Topic `/segmentation/depth_registered/points` veröffentlicht.
- Publisher 2: Die zum jeweiligen Objekt zugeordneten Punkte im Raum werden als Marker mit Objekt-ID über das Topic `/segmentation/visualization/marker` veröffentlicht.
- Publisher 3: Die zum jeweiligen Objekt gehörigen Objektnamen werden als Marker über das Topic `/segmentation/visualization/name` veröffentlicht.
- Publisher 4: Der Pyramidenstumpf der Objektregion wird als Marker über das Topic `/segmentation/visualization/line` veröffentlicht.

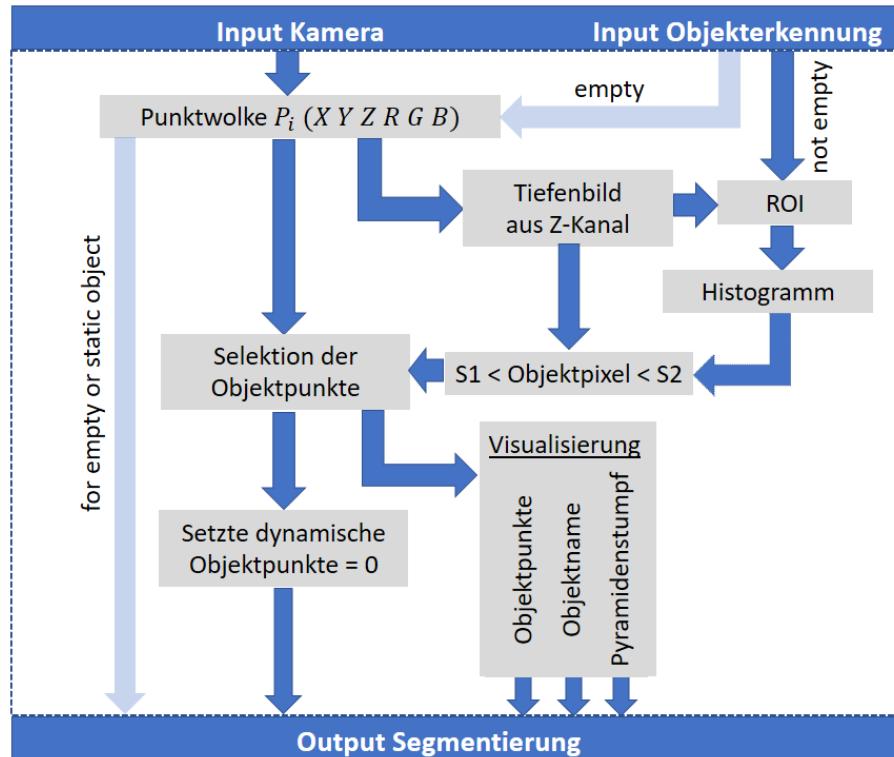


Bild 4.4: Datenverarbeitung innerhalb der `segmentation_node`

Die *segmentation_node* bekommt die Messages der abonnierten Topics als Input (siehe Bild 4.4). Die Punktwolke der Kamera wird zu einem Zeitpunkt t in die *segmentation_node* geladen, sofern die Objekterkennung zum gleichen Zeitpunkt mindestens ein Objekt detektiert hat. Aus dem Z-Kanal der Punktwolke wird anschließend in ein Tiefenbild erzeugt. Die Region der Bounding Box jedes Objektes wird selektiert. Ein Histogramm bestimmt in dieser Region einen oberen und einen unteren Schwellwert. Hiermit ist die Region of Interest dreidimensional durch einen Pyramidenstumpf eingeschränkt. Die Pixel des Tiefenbildes mit Einträgen zwischen den Schwellwerten werden im Tiefenbild und anschließend in der Punktwolke gekennzeichnet. Handelt es sich um ein dynamisches Objekt, werden die Objektpunkte aus der Punktwolke entfernt und die modifizierte Punktwolke über ein Topic versendet.

Parallel werden drei Markertypen erstellt. Ein Marker kennzeichnet die Objektpunkte, ein Marker visualisiert den Pyramidenstumpf der Objektregion und ein dritter Marker visualisiert den Objektnamen unter Angabe der Wahrscheinlichkeit. Die Marker werden über anschließend ebenfalls über Topics veröffentlicht.

4.3 Implementierung Package Kartierung

Das Package RGBDSLAM-V2 von Enderes u.a. [End15] wird nahezu vollständig implementiert für die Kartierung wiederverwendet. Lediglich die Punktwolke des Packages Segmentierung (Abschnitt 4.2) wird anstatt der Punktwolke der Kamera für die Kartierung verwendet (vgl. Bild 4.1).

5 Leistungsbewertung

Für die Leistungsbewertung der Implementierung wird eine exemplarische Umgebungskarte mit dem youBot aufgenommen. Diese Umgebungskarte wird hinsichtlich der Genauigkeit, der Zuverlässigkeit und der Fehleranfälligkeit untersucht.

5.1 Arena der Testumgebung

Als Testumgebung kommt eine für den youBot geeignete Indoor-Arena zum Einsatz (siehe Bild 5.1). Diese Arena muss der youBot mit Informationen aus der Umgebung und zusätzlich aus der Objekterkennung kartieren. Zur einfacheren Handhabung wird der youBot hierbei mit einem Joystick durch die Arena gesteuert.

Die Arena hat eine Fläche von ungefähr 15 m² und wird von einer weißen (vgl. Absorption



Bild 5.1: Übersicht der Arena

in Abschnitt 2.4) Bande begrenzt. Innerhalb der Arena werden einige Objekte des COCO-Datensatzes aus Tabelle A.1 platziert. Die Objekterkennung ist auf einem Rechner, der sich im gleichen WLAN-Netzwerk befindet, ausgelagert und muss die Objekte in der RGB-Bildsequenz des youBots erkennen. Die erkannten Objekte müssen vom youBot an der entsprechenden Stelle in der Karte markiert werden. Außerhalb der Arena befinden sich weitere diverse Objekte, welche ebenfalls innerhalb des Sichtfeldes der RGB-Kamera sein können. Hierbei kann es vorkommen, dass die Distanz zum Objekt größer als die Reichweite der Tiefenerkennung ist (vgl. Abschnitt 2.4.1).

Alle Objekte des COCO-Datensatzes sollen erfasst werden, jedoch sollen Objektpunkte von dynamischen Objekten nicht in die Karte integriert werden.

5.2 Vorbereitung und Konfiguration des youBots

Auf den youBot werden im Voraus die benötigten Packages geladen und kompiliert. Die Parameter der Kamerasensoren des youBots werden hinsichtlich der Bildformate und der Bildfrequenzen nach Abschnitt 3.2 konfiguriert. Die Tiefenreichweite der Realsense SR300 ist standardmäßig auf 200 cm begrenzt und wird daher maximiert. Exemplarisch wird das Gesamtsystem getestet. Hierbei treten folgende unerwünschte Eigenschaften auf:

- (a) Die Messung der Bildfrequenz beträgt nur 30 Hertz anstatt 60 Hertz, selbst direkt auf dem youBot.
- (b) Die Bildqualität nimmt durch die gesteigerte Tiefenreichweite der Kamera deutlich ab. Die rekonstruierte Punktfolge verrauscht.
- (c) Das Package RGBDSLAM-V2 kann aus den Kamera- und youBot-Daten keine zeitlich konstante Transformation zwischen Kamera und Umgebung berechnen. Es kommt zu hochfrequenten Positionsschwankungen, die das Ergebnis der Kartierung stark negativ beeinflussen. Vollständigerweise muss erwähnt werden, dass die Arena, gerade innerhalb des Sichtfeldes der Kamera, nur wenige optische Merkmale für die Bilderfassung von Features aufweist.

Diese Probleme können nicht kurzfristig behoben werden. Die Funktion der Implementierung wird von Problem (a) und (b) nicht beeinflusst. Das Problem (c) wird durch den Austausch des Packages RGBDSLAM-V2 umgangen. Der RGBDSLAM gegen einen 2D-SLAM (gmapping), zuzüglich einer Octomap ausgetauscht. Die Farbinformationen der Umgebung werden nicht berücksichtigt. Die Laserscanner des youBots werden für den 2D-SLAM verwendet und die RGB-D-Kamera des youBots für die Erstellung der Octomap. Der Informationsgehalt dieses Ansatzes ist äquivalent zum RGBDSLAM-V2, jedoch robuster, da der SLAM-Algorithmus

nicht von Bildfeatures der Umgebung abhängig ist (vgl. Abschnitt 2.4.2).

Ein alternatives Kameramodell kann nicht beschafft werden. Für den Anwendungsfall sind die Kameradaten der SR300 qualitativ ausreichend, sodass diese verwendet werden.

Folgende Parameter haben sich in Vortests bewährt und werden daher verwendet:

Der Schwellwert der Zuordnungswahrscheinlichkeit wird auf 65 % gesetzt. Die Filterbreite der Histogrammglättung wird bei dem Gaussian-Blur-Filter auf drei und bei dem Maximumfilter auf zehn gesetzt.

Die Würfelgröße der Octomap wird auf einen Zentimeter gesetzt.

5.3 Ergebnisse

Dieser Anschnitt beinhaltet die Ergebnisse der aufgenommenen Arena. Das Gesamtergebnis kann Bild 5.2 und Bild 5.3 entnommen werden. Zur grafischen Visualisierung der Arena wird RVIZ verwendet. Der youBot kann die Arena mit einer mittleren Frequenz von 26 Hertz aufnehmen und ist somit echtzeitfähig. Die Details der Ergebnisse werden in den folgenden Unterabschnitten, geordnet nach Packages beschrieben.

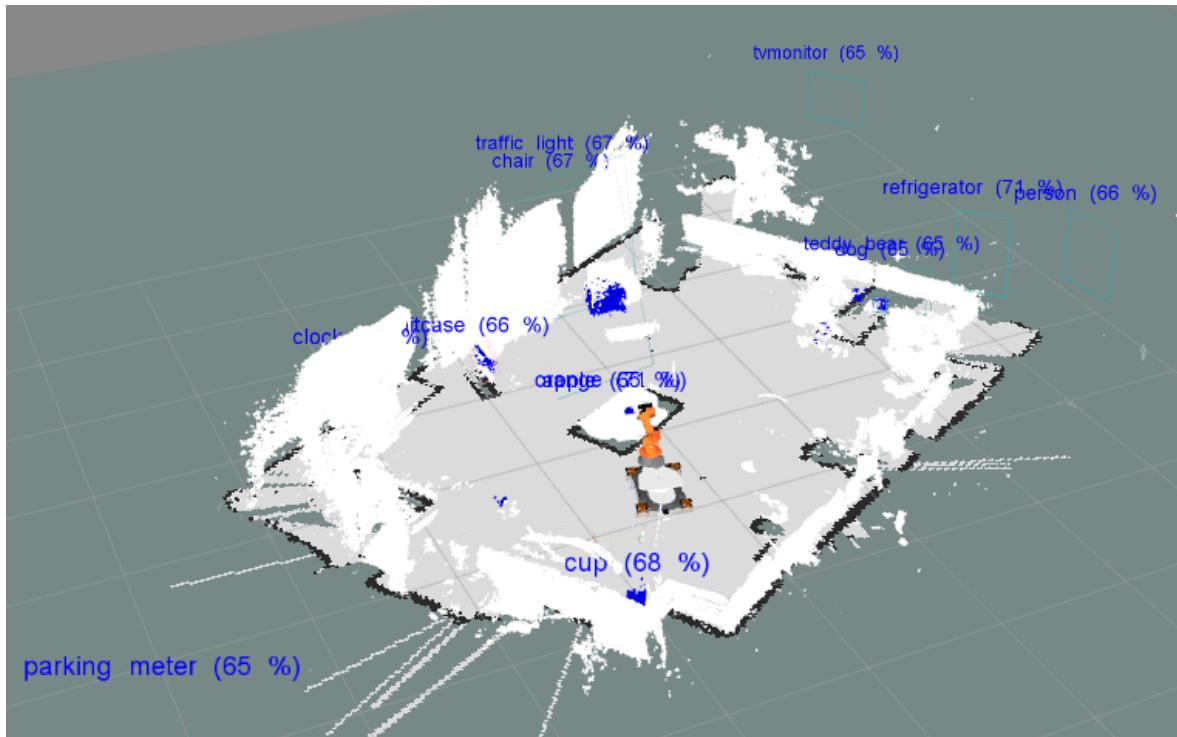


Bild 5.2: Gesamtergebnis der aufgenommenen Testarena

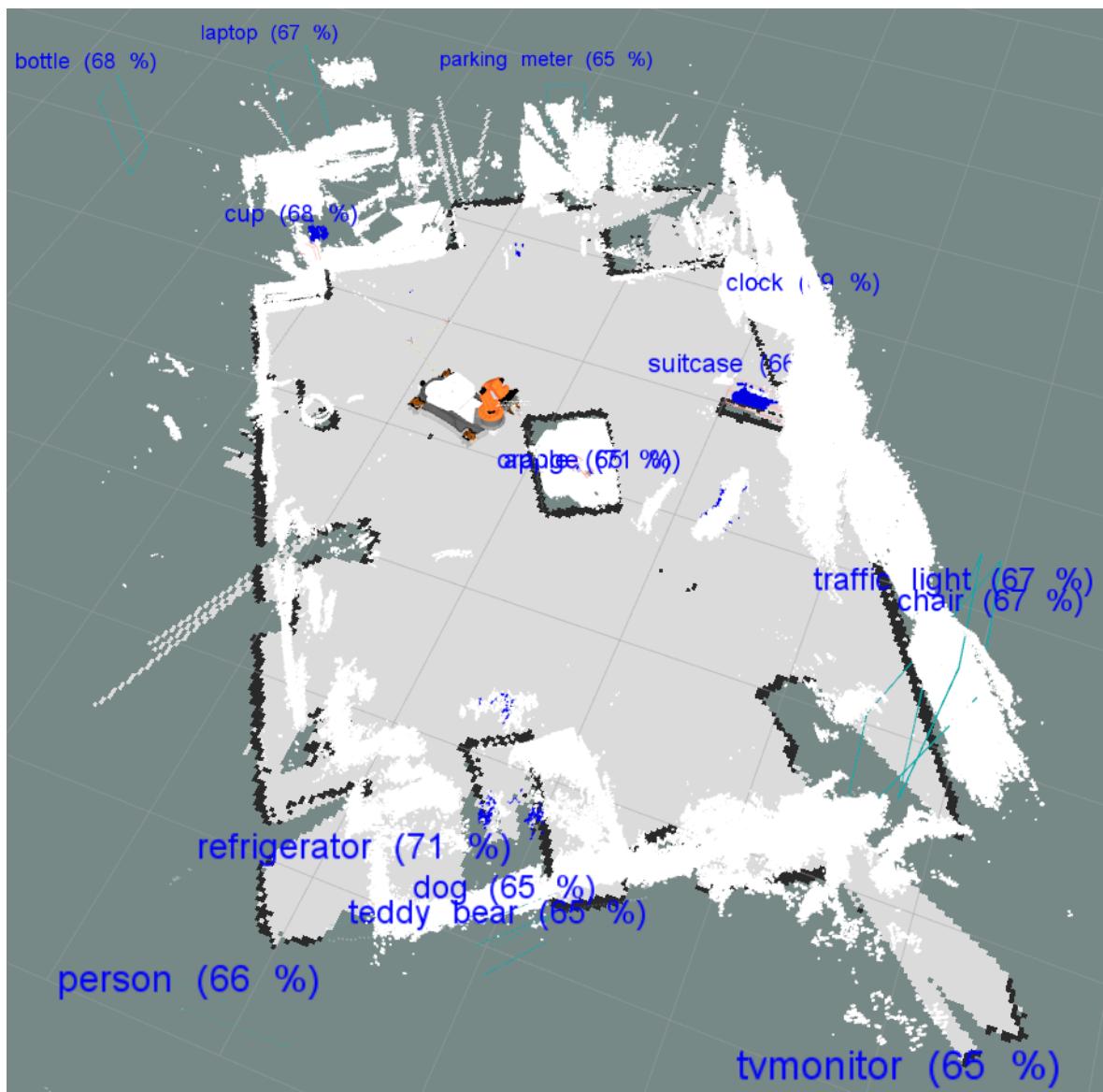


Bild 5.3: Gesamtergebnis der aufgenommenen Testarena

5.3.1 Ergebnisse und Auswertung der Objekterkennung

Die Zuordnung der Objektklasse ist teilweise nicht richtig, elf von 15 Objekten werden in der Arena bei einer mittleren Genauigkeit von 67 % korrekt zugeordnet (vgl. Bild 5.2 und Bild 5.3). Richtig erkannt wurden folgende Objekte: Stuhl (*chair*, 67 %), Tasse (*cup*, 68 %), Flasche (*bottle*, 68 %), Apfel (*apple*, 71 %), Uhr (*clock*, 69 %), Koffer (*suitcase*, 66%), Monitor (*tv-monitor*, 65%), Kühlschrank (*refrigerator*, 71 %), Person (*person*, 66 %), Teddybär (*teddy bar*, 65 %) und Laptop (*laptop*, 67 %).

Falsch erkannt wurden hingegen: Parkautomat (*parking meter*, 65 %), Hund (*dog*, 65 %), Orange (*orange*, 66 %) und Ampel (*traffic light*, 67 %).

Es ist zu beobachten, dass optisch ähnliche Gegenstände verwechselt oder doppelt zugewiesen werden. Der Apfel wird auch als Orange erfasst und eine rote Steckdose als Ampel. Weiterhin wird der Teddybär zusätzlich als Hund klassifiziert. Es wird deutlich, weder die Objektgröße, noch Informationen aus Nachbarbildern der Bildersequenz werden berücksichtigt.

Eine Anpassung des Schwellwerts der Zuordnungswahrscheinlichkeit wird nicht vorgenommen, da sich die Zuordnungswahrscheinlichkeit zwischen den falsch und richtig erkannten Objekten nicht signifikant unterscheidet. Strategien sind hierzu notwendig, um die Objekte über mehrere Bilder hinweg zu beobachten und anschließend zu entscheiden, um welches Objekt es sich höchstwahrscheinlich handelt.

5.3.2 Ergebnisse und Auswertung der Segmentierung

Die berechnete ROI ist als Pyramidenstumpf mit Farbwerten, abhängig der Zuordnungswahrscheinlichkeit abgebildet. Sofern das Objekt außerhalb der Tiefenreichweite der Kamera ist, wird das Objekt als 2D-Bounding Box in der Karte an der maximalen Tiefenreichweite vermerkt (siehe Kühlschrank und Monitor in Bild 5.3). Der Klassename und die Zuordnungswahrscheinlichkeit sind als Text oberhalb der ROI vermerkt.

Bei Objekten, die sich innerhalb der Teifenreichweite der Kamera befinden, werden die Objektpunkte innerhalb der ROI dem entsprechenden Objekt farblich zugewiesen. Die zugrundeliegende Histogrammsegmetierung über die Schwellwerte liefert abhängig von der Geometrie der Objekte qualitativ unterschiedliche Ergebnisse.

Problematisch erweisen sich Geometrien, die parallel zur Kameraachse verlaufen. Die Punktwolke ist bei diesen Flächen lückenhaft, sodass die Schwellwertsuche des Histogrammes zu vorzeitig beendet wird (vgl. Abschnitt 4.2). Ein Beispiel hierfür ist der in Bild 5.4 abgebildete Stuhl. Die Kameraachse des youBots befindet sich auf ähnlicher Höhe wie die Sitzfläche. Nur die vordere Kante der Stuhlfläche kann erkannt werden, sodass zwischen Sitzfläche und Rückenlehne eine Lücke entsteht.

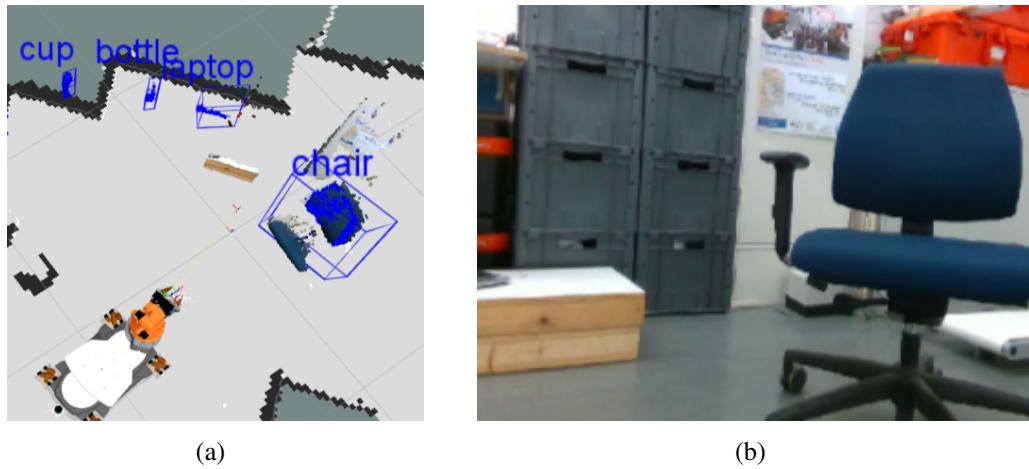


Bild 5.4: Fehlerhafte Segmentierung eines Stuhles

Objekte, wie die Tasse oder die Flasche, können hingegen vollständig in der Punktwolke zugewiesen werden (Bild 5.5). Die ROI in Form des Pyramidenstumpes schränkt die Objektrektion somit vollständig ein. Die zum Objekt gehörigen Punkte können erfolgreich zugewiesen werden.

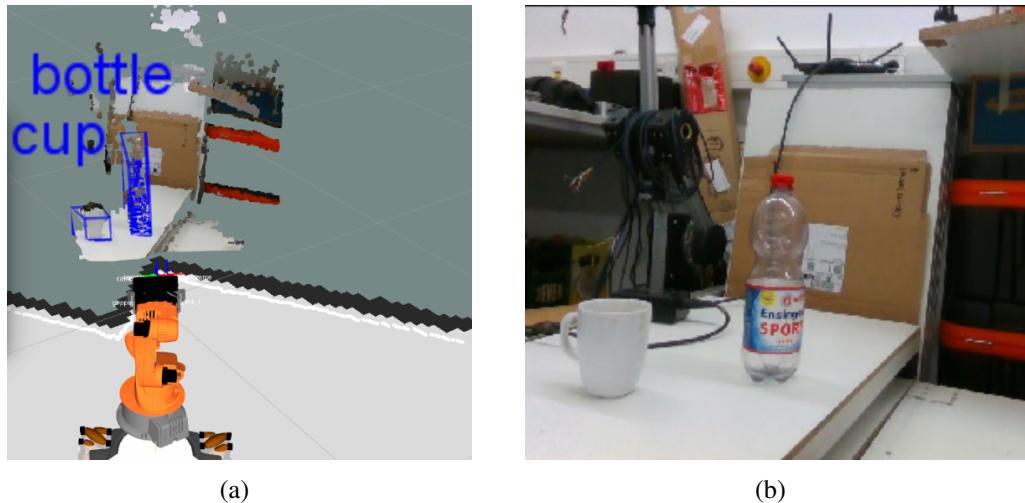


Bild 5.5: Erfolgreiche Segmentierung einer Tasse und einer Flasche

Objekte, die mehrfach von der Objekterkennung erkannt werden, sind leider nicht in der Karte vermerkt. Die Marker werden immer wieder durch die Objekterkennung aktualisiert. Hier ist

eine Strategie nötig, die beispielsweise auf Basis der Distanz zwischen zwei Objekten einen weiteren Marker der gleichen Objektklasse hinzufügt.

Weitere Ergebnisse der Segmentierung, jedoch nicht aus der Testarena, sind im Anhang B. Das Feature zur Verdeckung dynamischer Objekte aus der Karte kann der beiliegenden DVD entnommen werden.

5.3.3 Ergebnisse und Auswertung der Kartierung

Die Kartierung liefert in der Testarena gute Ergebnisse (siehe Bild 5.2). Die Begrenzungen der Arena werden von den Laserscannern des youBots erfasst. Durch die weiße Bande werden die Laserstrahlen reflektiert. Eine Erfassung der Position ist über den 2D-SLAM somit möglich (siehe Bild 5.2 und Bild 5.3). Die Octomap wird aus den Positionsdaten des 2D-SLAMs und nicht aus den Bilddaten erzeugt. Hierbei entstehen teilweise Positionsabweichungen, siehe Ecke der Arena links unten in Bild 5.3. Die Würfel der Bande stimmen in dieser Region nicht mit der Begrenzung der Laserscanner überein.

6 Fazit

Die Erkenntnisse dieser Studienarbeit werden in diesem abschließenden Kapitel zusammengefasst. Weiterhin wird ein kurzer Ausblick für zukünftige Arbeiten gegeben.

6.1 Zusammenfassung

Eine intelligente Umgebungskarte kann mit einem mobilen Roboter, auf dem eine RGB-D-Kamera montiert ist, in Echtzeit generiert werden. Hierbei werden sowohl die Position, als auch die Geometrie von Objekten ermittelt und in die Umgebungskarte integriert. Das Ergebnis wird aus einer Kombination von Objekterkennung, Segmentierung und Kartierung erzielt.

Die Objekterkennung klassifiziert und detektiert Objekte (Objektklassen nach Tabelle A.1) mit einer mittleren Genauigkeit von über 75 %. Sie basiert auf einer Methode des maschinellen Lernens, einem Faltungsnetzwerk (CNN) des Deep Learnings. Aus den Informationen des Farbbildes der Kamera berechnet ein ausgelagertes Faltungsnetzwerk für jedes erkannte Objekt eine Bounding Box und sendet diese an den Roboter. Die Projektion der Bounding Box auf Basis des Lochkameramodells in den Raum schränkt die *Region of Interest* zweidimensional ein. Mit Hilfe eines Histogrammes werden im Tiefenbild der Kamera zwei Schwellwerte für die Beschränkung der dritten Dimension berechnet. Die *Region of Interest* ist somit in Form eines Pyramidenstumpes gegeben. Die Position und die Geomtrie der erkannten Objekte wird über Marker in der abschließenden Kartierung visualisiert. Drei Features werden umgesetzt:

1. Dynamische Objekte sind in der statischen Karte unerwünscht und werden daher ausgebendet.
2. Drei Markertypen werden unterschieden: Die Objektpunkte, die Region des Objektes und der Objektname einschließlich Zuordnungswahrscheinlichkeit.
3. Objekte die vom Faltungsnetzwerk erkannt wurden, aber für eine dreidimensionale Erfassung außer Reichweite der Kamera sind, werden als zweidimensionale Bounding Box an der maximalen Kamerareichweite im Raum vermerkt.

Die Schwierigkeit der Arbeit beruht auf den komplexen Zusammenhängen verschiedener Problemstellungen. Eine besondere Herausforderung sind Methoden aus dem Deep Learning. Diese benötigen eine sehr hohe Rechenkapazität und können aktuell (01/2017) - auch mit

modernsten Recheneinheiten - nur im zweidimensionalen echtzeitfähige Ergebnisse erzeugen. Hieraus ergibt sich das Problem der Rekonstruktion vom Zwei- ins Dreidimensionale. Es fehlen Informationen, um eine zuverlässige Zuordnung der Objektpunkte zu gewährleisten. Verschiedene Segmentierungsmethoden wurden hierzu validiert, um die Objektpunkte von der Umgebung zu trennen.

6.2 Ausblick

Durch die erfolgreiche Implementierung des kombinierten Ansatzes wurde das Ziel der Arbeit eine intelligente Umgebungskarte zu erstellen erreicht. Dennoch gibt es Forschungsbedarf, um die Ergebnisse zu verbessern.

Wie bereits erwähnt, ist Deep Learning ein relativ neuer Forschungstrend. Betrachtet man die Entwicklung der Faltungsnetzwerke in den letzten Jahren, ist eine stetige Steigerung der Genauigkeit und der Effizienz zu erkennen. Für den sinnvollen Einsatz in der mobilen Robotik müssten Faltungsnetzwerke für die dreidimensionale Objekterfassung optimiert werden. Drei Möglichkeiten werden knapp vorgestellt:

1. Kurzfristige Verbesserungsmöglichkeit: Bildübergreifende Objekterkennung

Aktuell wird für jedes Farbbild die Objekterkennung neu angewendet, ohne Informationen zwischen den Nachbarbildern der Bildsequenz auszutauschen. Strategien sind für die Kartierung der Umgebungskarte erforderlich, um dynamische Objekte robust in jedem Bild der Bildsequenz zu erkennen und zu verdecken.

Weiterhin können durch eine geeignete Strategie - beispielsweise durch Betrachtung einer festen Objektregion - Doppelzuweisungen in der Kartierung vermieden und gleichzeitig die Genauigkeit gesteigert werden.

2. Mittelfristige Verbesserungsmöglichkeit: Modifikation der Detektion im Bild

Die Objekterkennung gibt die Objektkontur anstatt der Bounding Box aus. Hiermit wäre direkt eine eindeutige Zuordnung der Objektpunkte im Dreidimensionalen möglich, die Segmentierung der Objektpunkte entfällt. Die Detektion der YOLO-Packages (vgl. Abschnitt 2.3, Bild 2.7) kann hierzu modifiziert werden, indem das klassifizierte Gitter nicht mit den vorgeschlagenen Bounding Boxen, sondern mit deren zugrunde liegenden Objektkontur im Bild verglichen und ausgegeben wird.

3. Langfristige Verbesserungsmöglichkeit 1: 3D-Dateninput in das Faltungsnetzwerk

Da zweidimensionale Daten die Umgebungsinformationen bezüglich des Informationsgehaltes einschränken, wird als weitere Möglichkeit eine Berechnung über dreidimensionale Umgebungsdaten empfohlen. Der voraussichtliche Berechnungsaufwand ist hierbei, um

einiges Höher, sodass dies gerade für mobile Robotersysteme, eher eine langfristige Alternative ist (siehe auch [ESRB15]).

4. Langfristige Verbesserungsmöglichkeit 2: Trainieren von Teilnetzen

Neue Deep-Learning-Techniken erlauben das nachträgliche Training von Teilnetzen [Jäs16]. Um die Genauigkeit zu steigern und / oder neue Objektklassen anzulernen wird die Technik des *Tracking-Learning-Detection*-Tracker von Zedal Kalal u.a. [KMM10] empfohlen. Auf Basis eines initialisierten Bildausschnittes - die Bounding Box der Objekterkennung - werden in der fortlaufenden Bildsequenz neue Bilder des selben Objektes generiert. Die Daten könnten anschlieSSend für das Training eines Teilnetzes verwendet werden.

Mit dieser kurz- mittel- und langfristigen Möglichkeit zur Verbesserung der Ergebnisse können durchaus brauchbare Anwendungen im Bereich der mobilen Robotik abgeleitet werden.

Die in dieser Arbeit entstandene Implementierung könnte beispielsweise auf einen Service-roboter adaptiert werden, der als Haushaltshilfe (ausgelagerte Objekterkennung) oder als Wegbegleiter (interne Objekterkennung) für blinde Menschen nützlich sein kann. Die visuelle Wahrnehmung könnte somit ein Stück weit von einem mobilen Roboter übernommen werden.

Anhang

A Objektklassen der Objekterkennung

Tabelle A.1: COCO-Datensatz Objektklassen

0	person	20	elephant	40	wine glass	60	diningtable
1	bicycle	21	bear	41	cup	61	toilet
2	car	22	zebra	42	fork	62	tvmonitor
3	motorbike	23	giraffe	43	knife	63	laptop
4	aeroplane	24	backpack	44	spoon	64	mouse
5	bus	25	umbrella	45	bowl	65	remote
6	train	26	handbag	46	banana	66	keyboard
7	truck	27	tie	47	apple	67	cell phone
8	boat	28	suitcase	48	sandwich	68	microwave
9	traffic light	29	frisbee	49	orange	69	oven
10	fire hydrant	30	skis	50	broccoli	70	toaster
11	stop sign	31	snowboard	51	carrot	71	sink
12	parking meter	32	sports ball	52	hot dog	72	refrigerator
13	bench	33	kite	53	pizza	73	book
14	bird	34	baseball bat	54	donut	74	clock
15	cat	35	baseball glove	55	cake	75	vase
16	dog	36	skateboard	56	chair	76	scissors
17	horse	37	surfboard	57	sofa	77	teddy bear
18	sheep	38	tennis racket	58	pottedplant	78	hair drier
19	cow	39	bottle	59	bed	79	toothbrush

Tabelle A.2: VOC-Datensatz Objektklassen

0	aeroplane	5	bus	10	diningtable	15	pottedplant
1	bicycle	6	car	11	dog	16	sheep
2	bird	7	cat	12	horse	17	sofa
3	boat	8	chair	13	motorbike	18	train
4	bottle	9	cow	14	person	19	tvmonitor

B Weitere Abbildungen von Ergebnissen

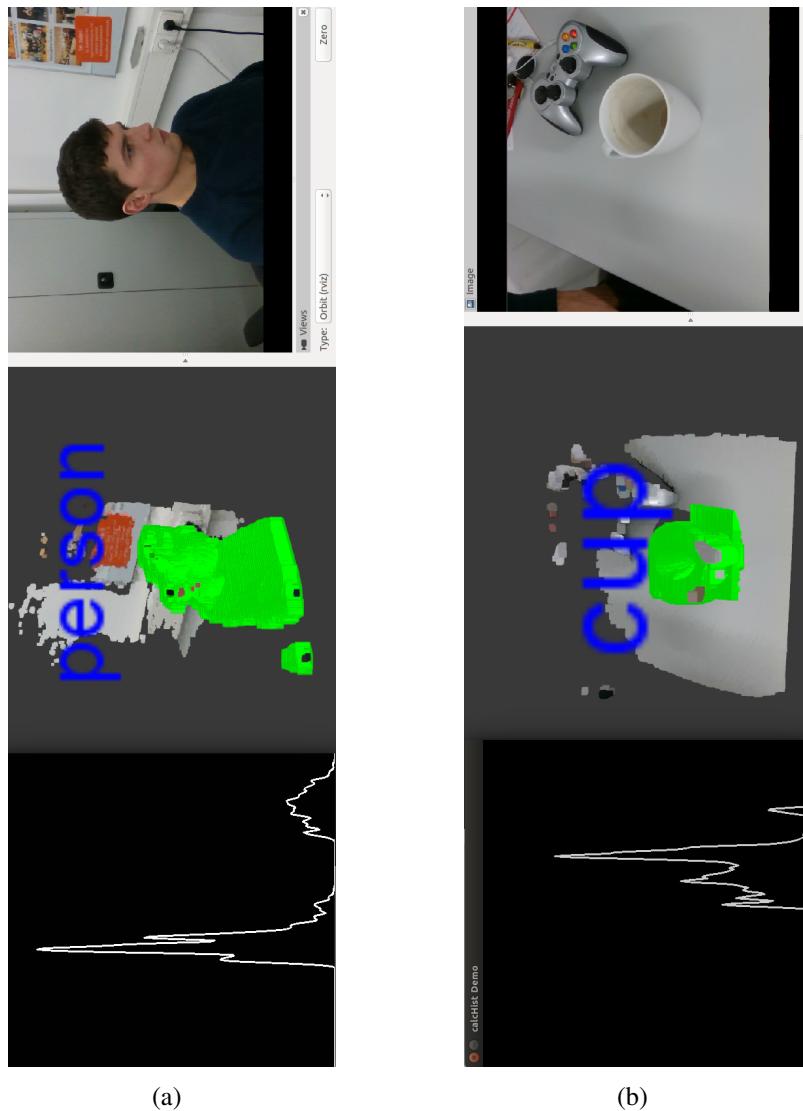


Bild B.1: Vortests zur histogrammbasierten Segmentierung

Bild B.1a und Bild B.1b zeigen Ergebnisse aus der Testphase der histogrammbasierten Segmentierung einer Person und einer Tasse.

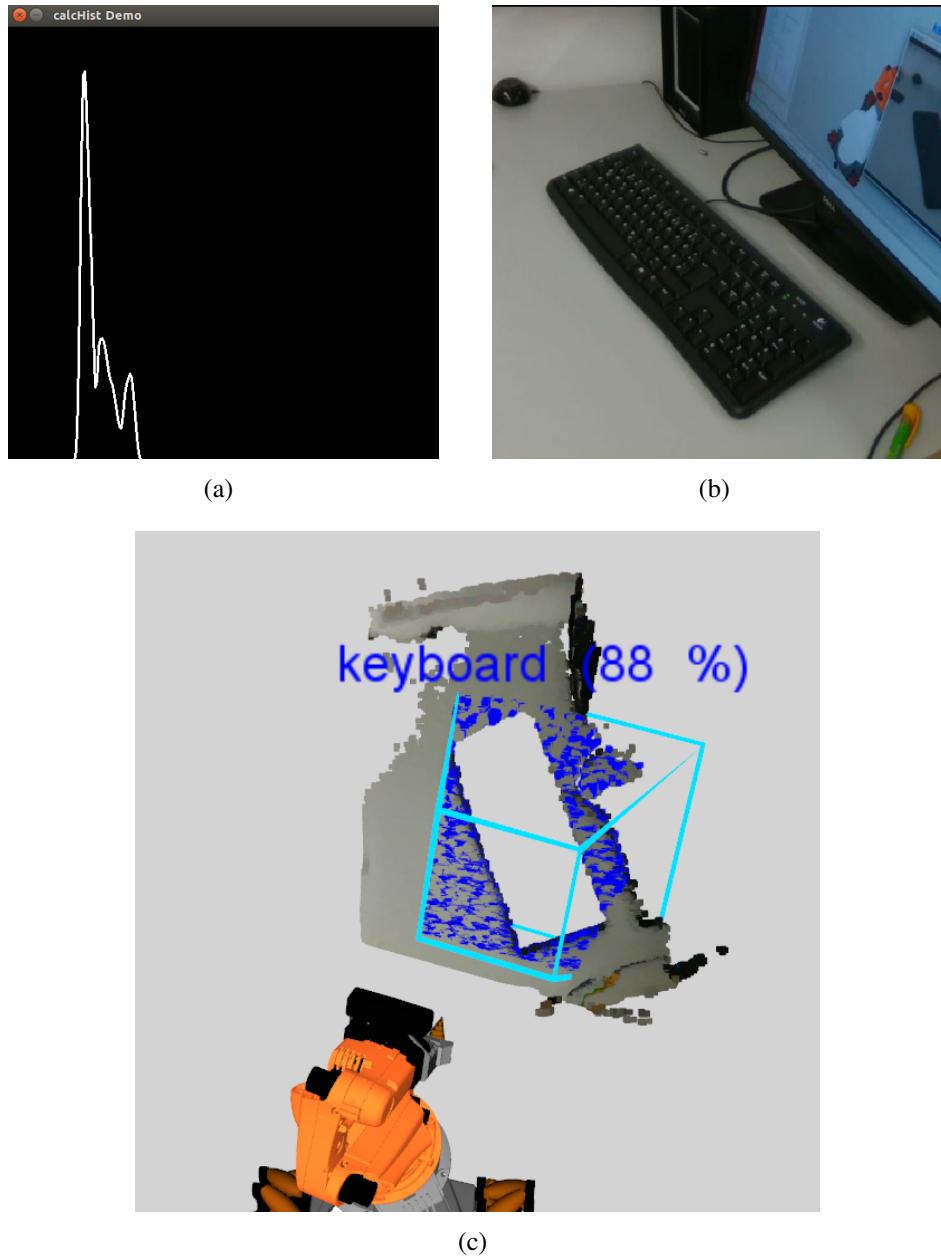


Bild B.2: Absorption schwarzer Flächen am Beispiel einer schwarzen Tastatur

In Bild B.2c ist die eine schwarze Tastatur im Farbbild zu sehen. Im Tiefenbild kann die Entfernung nicht berechnet werden, da die Infrarotstahlen des IR-Projektors der SR300 von der schwarzen Fläche absorbiert und nicht gespiegelt werden (vgl. Abschnitt 2.4.1). Somit entsteht in der Punktewolke (Abbildung B.2c) eine Lücke. Die histogrammbasierte Segmentierung scheitert.

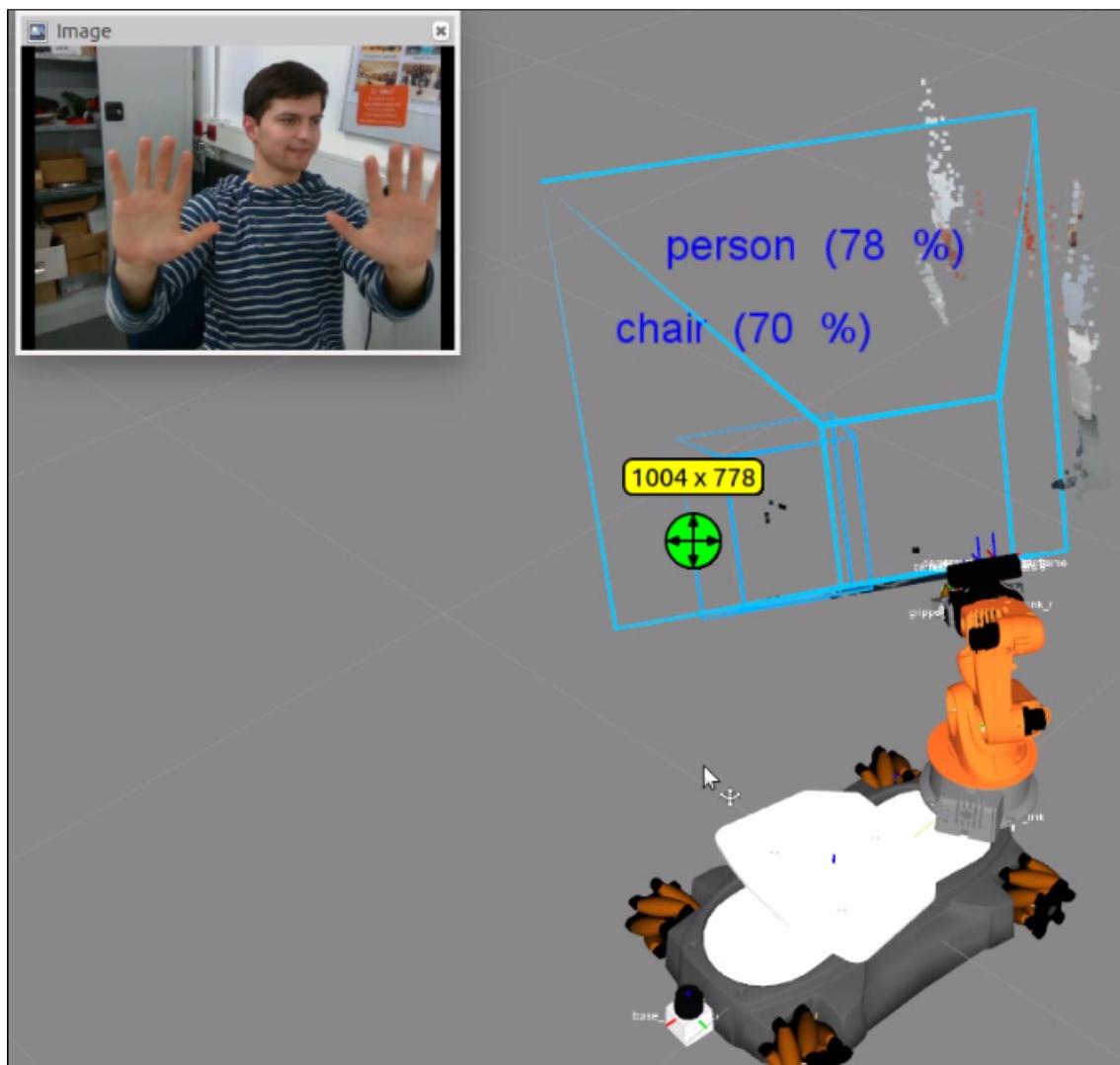


Bild B.3: Bild aus Videosequenz, Verdeckung dynamischer Objekte

In Bild B.3 ist der Ausschnitt einer Videosequenz zu sehen, die der beigelegten DVD entnommen werden kann. Hierbei wird die zu sehende Person (dynamisches Objekt, vgl. Abschnitt 3.2) in der Punktwolke verdeckt, sodass diese Punkte nicht in die Kartierung mit aufgenommen werden.

C Codeausschnitte der *segmentation_node*

Folgend wird die Implementierung unter C++ der *segmentation_node* anhand von Codeausschnitten beschrieben. Die gesamte Implementierung dieser Studienarbeit kann der beiliegenden DVD entnommen werden.

Im Konstruktor werden die Publisher und Subscriber definiert. Im Subscriber der Punktwolke wird das in Abschnitt 4.2 beschriebene Konzept der Segmentierung umgesetzt (siehe Codeausschnitt C.1).

Eingangs werden die Messages umgewandelt, sodass die Daten decodiert vorliegen (vgl. Abschnitt 3.2). Aus den Z-Kanal der Punktwolke wird eine Matrix erzeugt, ihre Einträge werden auf Werte zwischen 0 und 255 skaliert. Ein 8 Bit Grauwertbild entsteht. Mit einer For-Schleife werden nun zwischen Zeile 120 und 465 für jede von der Objekterkennung empfangene Bounding Box Operationen ausgeführt.

Für das Feature zur Verdeckung dynamischer Objekte wird abschließend entweder eine modifizierte Punktwolke (Zeile 472) oder die Input-Punktwolke ((Zeile 478) über das Topic */segmentation/depth_registered/points* an die Kartierung ausgegeben.

Codeausschnitt C.1: *segmentation_node*: Subscriber 2

```

90 void SegmentationNode::cloud_segment (const sensor_msgs::PointCloud2::ConstPtr& cloud_in)
91 {
92     if (!arrayBB.detections.empty()) // just do if object detection
93         found_object
94     {
95         // load and encoding Pointcloud from ROS message
96         pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud_pcl (new pcl::PointCloud<pcl::PointXYZRGB>);
97         pcl::fromROSMsg (*cloud_in, *cloud_pcl);
98
99         // Create empty matrix & pointer with size of z-channel
100        cv::Mat depth_image(cloud_pcl->height, cloud_pcl->width, CV_32FC1);
101        float* depth_ptr = depth_image.ptr<float>(0);
102
103        // fill z-values from pointcloud into pointer
104        float maxDistance = 0.0;
105        for (uint i = 0; i < cloud_pcl->points.size(); ++i)

```

```
105     {
106         depth_ptr[i] = cloud_pcl->points[i].z;
107
108         if (cloud_pcl->points[i].z>maxDistance) // get max Distance [m]
109             in z-direction
110         {
111             maxDistance=cloud_pcl->points[i].z;
112         }
113
114     // transform distance values [m] in gray scale 0-255
115     cv::Mat depth_image_8bit;
116     depth_image.convertTo(depth_image_8bit, CV_8U, 255/maxDistance);
117
118     // loop for each detected Bounding Box
119     for (uint i=0; i<arrayBB.detections.size();++i)
120     {
121         //histogram-based segmentation and marker definition
122
123
124
125     }
126
127     //publish modified point cloud 2
128     sensor_msgs::PointCloud2 cloud_out;
129     pcl::toROSMsg(*cloud_pcl, cloud_out);
130     cloud_out.header = cloud_in->header;
131
132     //publish modified PointCloud2 as sensor msgs
133     pub_segm_cloud_.publish (cloud_out);
134 }
135
136 else
137 {
138     //publish original PointCloud2 as sensor msgs
139     pub_segm_cloud_.publish (cloud_in);
140
141     ROS_INFO("no object detected");
142 }
143 }
```

In der For-Schleife in Zeile 30 (Codeausschnitt C.1) werden nun für jedes erkannte Objekt der Objekterkennung Operationen ausgeführt.

Die ROI im Tiefenbild kann über die Ausgabedaten der Objekterkennung - siehe Parameter

aus Codeausschnitt 4.1 - bestimmt werden. Innerhalb der ROI im Tiefenbild (cv::Mat roi) wird anschließend das Histogramm für die Schwellwertsuche erzeugt.

Für die Berechnung der Schwellwerte in Z-Richtung wird das Histogramm über die OpenCV Funktion *calcHist* [L26] zwischen den Werten eins und 255 im Tiefenbild berechnet. Pixel mit dem Eintrag Null werden somit ausgeschlossen. Dies ist notwendig da Pixel, die mangels fehlender Zuordnung auf null gesetzt sind, das Histogramm verfälschen. Die notwendigen Schritte zur Erstellung des Histogramms können dem Codeausschnitt C.2 entnommen werden.

Codeausschnitt C.2: *segmentation_node* Subscriber 2: Histogramm

```
159 Point BB_topleft (x1,y1), BB_downright (x2,y2); // set vertices of ROI
160 Mat roi(depth_image_8bit, Rect(BB_topleft,BB_downright)); // Select ROI
    in depth image
161
162 /* OpenCV calc-Hist:
163  * void calcHist(const Mat* images, int nimages, const int* channels,
164  InputArray mask, OutputArray hist, int dims, const int* histSize,
165  const float** ranges, bool uniform=true, bool accumulate=false )
166 */
167
168 // parameter initialization
169 int histSize = 256;
170 float range[] = {1, 255} ;
171 const float* histRange = {range};
172 Mat depth_hist;
173
174 calcHist( &roi, 1, 0, Mat(), depth_hist, 1, &histSize, &histRange, true
175 , false );
176
177 // prepare depth histogram for drawing (size 500x500) and normalize
178 int hist_w = 500; int hist_h = 500;
179 int bin_w = cvRound( (double) hist_w/histSize );
180 Mat histImage( hist_h, hist_w, CV_8UC1, 1 );
181 normalize(depth_hist, depth_hist, 0, histImage.rows, NORM_MINMAX, -1,
182 Mat() );
183
184 ...
185 // draw histogram
186 for( int i = 1; i < histSize; i++ )
187 {
188     line( histImage, Point( bin_w*(i-1), hist_h - cvRound(depth_hist.at<
189     float>(i-1)) ) , Point( bin_w*(i), hist_h - cvRound(depth_hist.at<
190     float>(i)) ), Scalar( 255 ), 2, 8, 0 );
```

```
186 }
```

Das Histogramm wird vor der Berechnung der Schwellwerte mit Filtern bearbeitet (siehe Codeausschnitt C.2). Die Parameter der Filter sind auf dem Parameterserver hinterlegt, sodass diese einfach angepasst werden können. Zum einen ist dies erforderlich, da die Kamera für einzelne Tiefenwerte innerhalb einer Objektregion keine Treffer liefert. Zum anderen erfordert dies die geometrische Beschaffenheit der Objekte. Eine zu hohe Filterbreite kann allerdings dazu führen, dass Regionen von Objekt und Hintergrund im Histogramm verschmelzen. Auf die Histogrammerstellung folgt die Schwellwertsuche. Ausgehend vom globalen Maximum (*index_maxhist*), wird die Kurve des Histogramms in beide Richtungen abgetastet. Der Code zur Berechnung der unteren Schwelle kann dem Codeausschnitt C.3 entnommen werden. Der erste Kontakt zwischen Kurve und Ursprung der Ordinate wird jeweils als Schwellwert auf der Abszisse festgelegt. Es wird somit angenommen, dass die maximale Häufigkeit im Histogramm zum Objekt gehört.

Codeausschnitt C.3: *segmentation_node* Subscriber 2: Schwellwertsuche in Histogramm

```
225 hist_lower_border=index_maxhist; // set startpoint at global max
226 bool lower_border=false,
227 while(lower_border!=true) //search while border was not found
228 {
229     if(depth_hist.at<float>(hist_lower_border)<=2.0)
230     {
231         lower_border=true; // finish: exit loop
232     }
233     if(lower_border==false)
234     {
235         hist_lower_border = hist_lower_border -1; // move to lower
236             index in hist
237     }
238     ...// same for upper border
```

Mit Kenntnis beider Schwellwerte werden die Einträge der ROI modifiziert. Jeder Eintrag, der einen Wert größer als der untere Schwellwert und kleiner als der obere Schwellwert hat, wird durch den Wert *valRoi* ersetzt. Die Objektpunkte sind somit im Tiefenbild gekennzeichnet.

ROS beinhaltet Marker zur Visualisierung [L27] unter RVIZ [L28]. Codeausschnitt C.1 zeigt die Implementierung der Objektpunkte als Marker. In Zeile 266-276 werden zunächst grundlegende Einstellungen des Markers *points* festgelegt. Jeder Objektpunkt wird hier als blauer Würfel mit einer Kantenlänge von 3 cm und definiert.

In einer For-Schleife wird jeder Punkt im Array des Tiefenbildes überprüft, ob dieser den Wert

valRoi hat. Bei einem Treffer (Zeile 281) wird die eingangs im Codeausschnitt C.1, Zeile 95 erstellte Punktwolke an der entsprechenden Position des Tiefenbildes aufgerufen und der Eintrag von X-, Y- und Z-Koordinate ausgelesen. Diese Punkte (XYZ) werden in einem weiteren Array gesammelt und abschließend als Marker Message auf dem Topic */segmentation/visualization/points* veröffentlicht. Der Code für die Visualisierung des Klassennamens ist mit Ausnahme der nicht benötigten For-Schleife nahezu identisch. Die Position des Klassennamens wird nicht auf jedem Punkt, sondern nur einmal oberhalb des Objektschwerpunktes gesetzt. Zuvor muss jedoch die Datei der Objektnamen geladen und der Name über die Objekt-ID ausgelesen werden.

Codeausschnitt C.4: *segmentation_node* Subscriber 2: Visualisierung Objektpunkte

```
266 visualization_msgs::Marker points; // initialize marker
267 points.header.frame_id = cloud_in->header.frame_id; //set frame id
268 points.header.stamp = ros::Time::now(); //set time stamp
269 points.action = visualization_msgs::Marker::ADD;
270 points.pose.orientation.w = 1.0;
271 points.id = objectclass; //set point id
272 points.type = visualization_msgs::Marker::POINTS; //set type of marker
273 points.scale.x = 0.03; // set size of cube in x
274 points.scale.y = 0.03; // ..and same y
275 points.color.b = 1.0f; // set green color
276 points.color.a = 0.6; // set transparency
277
278 array.assign(depth_image_8bit.datastart, depth_image_8bit.dataend);
279 for (int k=0;k<cloud_pcl->points.size();++k) // go through point cloud
280 {
281     if(array[k]==valRoi) // if point is from object
282     {
283         //push back point (x,y,z) in Array
284         geometry_msgs::Point p;
285         p.x = cloud_pcl->points[k].x;
286         p.y = cloud_pcl->points[k].y;
287         p.z = cloud_pcl->points[k].z;
288         points.points.push_back(p);
289     }
290 }
291 pub_points_.publish(points); //publish array points as visualization
marker
```

Die dreidimensionale Einschränkungen aus Bounding Box und Schwellwerte der Segmentation wird ebenfalls als Marker erfasst. Hierzu werden Kamerainformationen vom Topic */camera/rgb/camera_info* verwendet, um die Eckpixel der Bounding Box über das Lockka-

meramodell in den Raum zu projizieren (vgl. Codeausschnitt C.5.) Die projizierten Punkte haben eine Entfernung von einem Meter. Hieraus und mit Kenntnis der Schwellwerte des Histogramms werden die Eckpunkte des Pyramidenstumpfes berechnet. Äquivalent zum ersten Punkt in Zeile 389-392 werden alle acht Eckpunkte bestimmt. Abschließend mit einer Line entsprechend verbunden und als Marker Message über das Topic */segmentation/visualization_line* veröffentlicht.

Codeausschnitt C.5: *segmentation_node* Subscriber 2: 3D Projektion der Bounding Box über Lochkameramodell

```
378 // project pixel form BB in 3D points via pinhole model
379 pinmodel.fromCameraInfo(camerainfo);
380 Point BB_topright (x2,y1), BB_downleft (x1,y2);
381
382 // rect BB in z=1m
383 cv::Point3d P_topleft = pinmodel.projectPixelTo3dRay(BB_topleft);
384 cv::Point3d P_downright = pinmodel.projectPixelTo3dRay(BB_downright);
385 cv::Point3d P_topright = pinmodel.projectPixelTo3dRay(BB_topright);
386 cv::Point3d P_downleft = pinmodel.projectPixelTo3dRay(BB_downleft);
387
388 // vertices front rect
389 geometry_msgs::Point p1;
390 p1.x= P_topleft.x*z1;
391 p1.y= P_topleft.y*z1;
392 p1.z= P_topleft.z*z1;
393
394 ...
```

Literaturverzeichnis

- [Ade13] ADEN, Simon: Entwicklung eines effizienten Kommunikationsprotokolls zum globalen Informationsaustausch zwischen mobilen Robotern und externen Sensoren. (2013)
- [AGT⁺15] ARANDJELOVI, Relja ; GRONAT, Petr ; TORII, Akihiko ; PAJDLA, Tomas ; SIVIC, Josef: CNN architecture for weakly supervised place recognition. (2015). goo.gl/LJ7MK8
- [Ben09] BENGIO, Yoshua: Learning Deep Architectures for AI. (2009). <https://arxiv.org/pdf/1312.6098.pdf>
- [Bro15] BROWN, Larry: Deep Learning for Image Classification. (2015). goo.gl/OSXDCY
- [End15] ENDRES, Felix: *Robot Perception for Indoor Navigation*, Albert-Ludwigs-Universität Freiburg, Diss., 2015. <https://goo.gl/GTN2kj>. – Code http://felixendres.github.io/rgbdslam_v2/
- [Ert13] ERTEL, Wolfgang: *Grundkurs Künstliche Intelligenz, 3. Auflage*. Springer-Verlag, 2013
- [ESRB15] EITEL, Andreas ; SPINELLO, Jost Tobias S. ; RIEDMILLER, Martin ; BURGARD, Wolfram: Multimodal Deep Learning for Robust RGB-D Object Recognition. (2015). goo.gl/ZwG25q
- [Jäh05] JÄHNE, Bernd: *Digitale Bildverarbeitung, 6. Auflage*. Springer-Verlag, 2005
- [Jäs16] JÄSCHKE, Robert: *Skript zur Vorlesung Künstliche Intelligenz*. Institut für Verteilte Systeme, Leibniz Universität Hannover, April 2016
- [KMM10] KALAL, Zdenek ; MIKOŁAJCZYK, Krystian ; MATAS, Jiri: Tracking-Learning-Detection. (2010)
- [Kou16] KOUBAA, Anis: *Robot Operating System (ROS)*. Springer-Verlag, 2016

- [KSC13] KERL, Christian ; STURM, Jürgen ; CREMERS, Daniel: Dense Visual SLAM for RGB-D Cameras. (2013). goo.gl/QcMmt6. – Code <http://vision.in.tum.de/data/software/dvo>
- [KSH12] KRIZHEVSKY, Alex ; SUTSKEVER, Ilya ; HINTON, Geoffrey: ImageNet Classification with Deep Convolutional Neural Networks. (2012). goo.gl/VK3yLm
- [LAE⁺16] LIU, Wei ; ANGUELOV, Dragomir ; ERHAN, Dumitru ; SZEGEDY, Christian ; REED, Scott ; FU, Cheng-Yang ; BERG, Alexander C.: SSD: Single Shot MultiBox Detector. (2016). <https://arxiv.org/pdf/1512.02325v5.pdf>
- [MAK12] *KUKA youBot - Betriebs- und Montageanleitung.* : KUKA youBot - Betriebs- und Montageanleitung, 2012
- [MAT16] MUR-ARTAL, Raul ; TARDOS, Juan D.: ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. (2016). <https://128.84.21.199/pdf/1610.06475v1.pdf>. – Code https://github.com/raulmur/ORB_SLAM2
- [Neh02] NEHMZOW, Ulrich: *Mobile Robotik - Eine praktische Einführung*. Springer-Verlag, 2002
- [Nie16] NIELSEN, Michael: Neural Networks and Deep Learning. (2016). goo.gl/DZYUj8
- [NT15] NIEVES, Christopher A. ; TAYLOR, Camillo J.: Caffe Framework on the Jetson TK1: Using Deep Learning for Real Time Object Detection. (2015). <https://goo.gl/d3QwNw>
- [Ort16] ORTMAIER, Tobias: *Skript zur Vorlesung Robotik II*. Institut für Mechatronische Systeme, Leibniz Universität Hannover, April 2016
- [PMB13] PASCANU, Razvan ; MONTUFAR, Guido ; BENGIO, Yoshua: On the number of response regions of deep feedforward networks with piecewise linear activations. (2013). <https://arxiv.org/pdf/1312.6098.pdf>
- [RDGF16] REDMON, Joseph ; DIVVALA, Santosh ; GIRSHICK, Ross ; FARHADI, Ali: You Only Look Once: Unified, Real-Time Object Detection. (2016). <https://arxiv.org/pdf/1506.02640v5.pdf>

-
- [RF16] REDMON, Joseph ; FARHADI, Ali: YOLO9000: Better, Faster, Stronger. (2016). goo.gl/AZRfFs
 - [RHGS16] REN, Shaoqing ; HE, Kaiming ; GIRSHICK, Ross ; SUN, Jian: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. (2016). <https://arxiv.org/pdf/1506.01497.pdf>
 - [Sch14] SCHAAL, Stefan: Roboter werden selbstständig. In: *Jahresbericht Max-Planck-Gesellschaft* (2014). goo.gl/Sr43pe
 - [SZ15] SIMONYAN, Karen ; ZISSERMAN, Andrew: Very Deep Convolutional Networks for Large Scale Image Recognition. (2015). <https://arxiv.org/pdf/1409.1556.pdf>
 - [TL16] TAI, Lei ; LIU, Ming: Deep-learning in Mobile Robotics - from Perception to Control Systems: A Survey on Why and Why not. (2016). <https://arxiv.org/pdf/1612.07139.pdf>
 - [WKF⁺12] WHELAN, Thomas ; KAESZ, Michael ; FALLON, Maurice ; JOHANNSSON, Hordur ; LEONARD, John ; McDONALD, John: Kintinuous: Spatially Extended KinectFusion. (2012). <https://goo.gl/df7j7n>. – Code <https://github.com/mp3guy/Kintinuous>
 - [WSMG⁺15] WHELAN, Thomas ; SALAS-MORENO, Renato F. ; GLOCKER, Ben ; DAVISON, Andrew J. ; LEUTENECKER, Stefan: ElasticFusion: Real-Time Dense SLAM and Light Source Estimation. (2015). <http://www.thomaswhelan.ie/Whelan16ijrr.pdf>. – Code <https://github.com/mp3guy/ElasticFusion>
 - [ZF13] ZEILER, Matthew D. ; FERGUS, Rob: Visualizing and Understanding Convolutional Networks. (2013). <https://arxiv.org/pdf/1311.2901v3.pdf>

Linksammlung

Stand 03.02.2017 - Google URL-Shortener

- [L1] goo.gl/jbj2AL
- [L2] goo.gl/DcuV2H
- [L3] goo.gl/IoETPT
- [L4] goo.gl/BH8C1Y
- [L5] goo.gl/ue85Pz
- [L6] goo.gl/wmBKHP
- [L7] goo.gl/8bGECS
- [L8] goo.gl/kXNEbC
- [L9] goo.gl/LU6XR7
- [L10] goo.gl/Xci1uk
- [L11] goo.gl/61gdhu
- [L12] goo.gl/8xR09R
- [L13] goo.gl/sm0kpG
- [L14] goo.gl/f7UC24
- [L15] goo.gl/IgxjpZ
- [L16] goo.gl/6jlVsw
- [L17] goo.gl/qTj31O
- [L18] goo.gl/eF7C1w
- [L19] goo.gl/KUFJ5E

- [L20] goo.gl/y14cE3
- [L21] goo.gl/400Pbz
- [L22] goo.gl/pkFPFs
- [L23] goo.gl/0L3JgF
- [L24] goo.gl/94qw6B
- [L25] goo.gl/puWZsS
- [L26] goo.gl/AjDO33
- [L27] goo.gl/D7W4UU
- [L28] goo.gl/BA1UiH