Master Thesis

Mike Steven Zander

Tiefendaten unterstütze Bildauswertung für die Realisierung einer Mensch-Roboter-Kollaboration

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Mike Steven Zander

Tiefendaten unterstützte Bildauswertung für die Realisierung einer Mensch-Roboter-Kollaboration

Master Thesis eingereicht im Rahmen der Masterprüfung
im Studiengang Automatisierungstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Jochen Maaß
Zweitgutachter: Prof. Dr.-Ing. Jörg Dahlkemper

Abgegeben am 15. September 2017

Mike Steven Zander

Title of this paper
Implementation of a human-machine-collaboration by depth data mediated image evaluation

Keywords
image processing, Human-robot collaboration, HRC-System, openCV, OGRE3D, UR5, Kinect, Time-of-flight, Image moment, Cluster, k-Means, Collision detection

Abstract
Within the Project "Smart Productions" of the HAW Hamburg, a system was designed to determine and recognize the position of one human and multiple transport boxes in a defined working area. With this approach, the basis for a human-robot collaboration was established. The purpose of this thesis is to investigate the theory of the used image processing algorithms, the needed coordinate transformation between the environment and the image and the process of depth value measurements. Furthermore, two approaches for the correction of images made by an unevenly placed camera will be applied and the environmental influences on the range images will be investigated and characterized. Additionally, the object recognition will be established using the depth and color sensors by the image moments and the registration of distinct properties of the objects. The recognition of humans will be done by using clusters in the depth image and classification of convex hulls by adjusting for the robot's position. Eventual collisions between the human and the robot will be detected by an overlap in image contours and result in the termination of the robot's movement. Additionally, the results of the classification and object recognition by the OGRE-3D engine will be visualized, which is integrated fully in a QT-framework based user interface.

Mike Steven Zander

Thema der Masterthesis
Tiefendaten unterstützte Bildauswertung für die Realisierung einer Mensch-Roboter-Kollaboration

Stichworte
Bildverarbeitung, Mensch-Roboter-Kollaboration, MRK-System, openCV, OGRE3D, UR5, Kinect, Time-of-flight, Bildmoment, Cluster, k-Means, Kollisionserkennung

Kurzzusammenfassung
Im Zuge des Projektes „Smart-Production" der HAW Hamburg gilt es ein System zu entwickeln, mit dessen Hilfe die Position eines Menschen und mehrerer Transportboxen in einem definierten Arbeitsumfeld erkannt werden. Auf diese Weise soll die Grundlage für eine Mensch-Roboter-Kollaboration (MRK) geschaffen werden. Diese Thesis behandelt dabei die Theorie der umgesetzten Bildverarbeitungsalgorithmen, die notwendigen Koordinatentransformationen zwischen der Umgebung und dem Bild sowie das Verfahren der Tiefenwertmessung. Darüber hinaus werden zwei Ansätze für die Korrektur der schräg platzierten Kamera behandelt und vorhandene Fremdeinflüsse auf das Tiefenbild untersucht und beschrieben. Anschließend erfolgt die Umsetzung der Objekterkennung unter Verwendung des Tiefen- und Farbsensors durch die Hu-Momente sowie der Erfassung definierter Merkmale. Die Erkennung des Menschen erfolgt in dem Tiefenbild durch Clustern und Klassifizieren von konvexen Hüllen, indem ein Abgleich mit der Roboterposition stattfindet. Kollisionen zwischen Mensch und Roboter werden durch eine Schnittmenge der Konturen erkannt und stoppen die Bewegung des Roboters. Zusätzlich werden die Ergebnisse der Klassifizierung und Objekterkennung durch die OGRE-3D-Engine visualisiert, die vollständig in eine QT-Framework basierende Benutzeroberfläche integriert ist.

Danksagung

An dieser Stelle möchte ich mich bei allen Unterstützern während des Bearbeitungszeitraumes bedanken, die zu dem Erfolg der Thesis beigetragen haben.
Bedanken möchte ich mich besonders bei Herrn Prof. Jochen Maaß für die Bereitstellung des Themas und der kollegialen und hilfsbereiten Unterstützung während der Durchführung der Thesis. Ein weiterer Dank geht an Herrn Prof. Jörg Dahlkemper für die Funktion als Zweitprüfer. Für die praktische Unterstützung möchte ich mich bei dem Arbeitsteam von Herrn Maaß, bestehend aus Karin Volkmann, Sven Berding und Alexei Konowalow, bedanken. Weiterhin danke ich Dr. Bernhard Goetze für die Bereitstellung der Unterlagen für die Korrektur der Schräglage und Frau Prof. Annabella Rauscher-Scheibe für die Unterstützung zu dem Expectation-Maximization-Algorithmus. Für die Unterstzützung während des gesamten Studiums, geht ein besonderer Dank an meine Eltern, Roswita Zander und Ralph Zander, sowie an Janice Raabe.

Inhaltsverzeichnis

Abbildungsverzeichnis **v**

Tabellenverzeichnis **vii**

1 Einleitung **1**
1.1 Problemstellung . . . 2
1.2 Aktueller Stand . . . 3

2 Arbeitsumfeld **4**
2.1 Technischer Aufbau . . . 4
2.2 Kinect-Sensor . . . 5
2.2.1 „Time of Flight“-Funktionsprinzip . . . 5
2.3 UR5-Roboter . . . 7
2.4 Transportobjekte . . . 8
2.5 Auswahl der Programmierumgebung und Bildverarbeitungsbibliothek . . . 9
2.6 Auswahl des 3D-API . . . 10
2.6.1 OGRE-Engine . . . 11

3 Anforderungsanalyse **13**

4 Schräglagenkorrektur **15**
4.1 Lösungsansatz durch Minimierung der Standardabweichung . . . 15
4.2 Lösungsansatz durch Minimierung der Fehlerquadrate . . . 16
4.3 Gegenüberstellung und Validierung . . . 17

5 Koordinatentransformation **19**
5.1 Rotation und Translation . . . 19
5.2 Homogene Koordinaten im zweidimensionalem Raum . . . 20
5.3 Lochkameramodell . . . 21
5.4 Intrinsische Kameraparameter . . . 23
5.5 Extrinsische Kameraparameter . . . 24
5.6 Transformation zwischen Pixel- und Weltkoordinaten . . . 25
5.7 Verzeichnung . . . 26

6 Datenerfassung und Aufbereitung **27**
6.1 Untersuchung des Sensors . . . 27
6.1.1 Tiefenbild . . . 27
6.1.2 Coordinate-Mapper . . . 29
6.1.3 Farbbild . . . 30
6.2 Merkmalsextraktion zur Detektion der Transportboxen im Tiefenbild . . . 31
6.2.1 Ansatz über die konvexe Hülle und Hu-Momenten . . . 35
6.2.2 Ansatz über die Hough-Transformation . . . 37
6.2.3 Gegenüberstellung und Validierung . . . 38
6.3 Merkmalsextraktion zur Detektion der Transportboxen im Farbbild . . . 42
6.3.1 HSV-Farbraum . . . 42
6.3.2 Vorsegmentierung . . . 45
6.3.3 Differential . . . 45
6.3.4 Kreuzkorrelationsfunktion . . . 46
6.3.5 Gegenüberstellung und Validierung . . . 46
6.4 Merkmalsextraktion zur Detektion der menschlichen Gliedmaßen . . . 49
6.4.1 Vorsegmentierung . . . 50
6.4.2 Ansatz über die Ortsinformation der Objektkontur . . . 51
6.4.3 Ansatz über Clustern mit dem K-Means-Algorithmus . . . 52
6.4.4 Ansatz über Clustern mit dem Expectation-Maximization-Algorithmus . . . 53
6.4.5 Gegenüberstellung und Validierung . . . 55
6.5 Klassifizierung und Kollisionserkennung . . . 57
6.5.1 Robotermodell . . . 57
6.5.2 Klassifizierung der Cluster . . . 57
6.5.3 Kollisionserkennung . . . 59
6.5.4 Validierung . . . 61

7 Praktische Umsetzung **67**
7.1 Entscheidungsgrundlage für die Verwendung der untersuchten Algorithmen . . . 67
7.2 Schnittstellen . . . 68
7.2.1 Schnittstelle zu der Kinect . . . 68
7.2.2 Schnittstelle zu der OGRE-Engine . . . 69
7.2.3 Schnittstelle zu dem UR5-Roboter . . . 70
7.3 Programmstruktur . . . 72
7.3.1 Die Klasse „framesource“ . . . 74
7.3.2 Die Klasse „configurator“ . . . 75
7.3.3 Die Klasse „corrector“ . . . 76
7.3.4 Die Klasse „armestimator“ . . . 77
7.3.5 Die Klasse „collisiondetect“ . . . 78
7.3.6 Die Klasse „objectestimator“ . . . 79
7.3.7 Die Klasse „renderengine“ . . . 80
7.3.8 Die Klasse „robotcontroll“ . . . 81

8 Durchführung und Validierung **82**
8.1 Konfiguration 82
8.2 Greifen und Bewegen einer Box 84
8.3 Kollisionserkennung 86
8.4 Performance 87

9 Ergebnis **88**

10 Ausblick **90**

Literatur **91**

11 Anhang **94**

Abbildungsverzeichnis

1 *Projektumgebung mit der Kinect (grün), dem Arbeitsbereich (blau), dem Sicherheitsbereich (rot), sowie dem UR5 Roboter (gelb).* . . . 4
2 *Signalverlauf des ausgehenden (Light) und eingehenden Lichts (Return) und die erzeugte Ladung (A und B). [JS14, Seite:50]* . . . 6
3 *Signalverlauf des ausgehenden Lichts (Radiated IR Signal), des eingehenden und verzögerten Lichts (Reflected IR Signal) sowie die Ladungen Q_1 und Q_2 in Abhängigkeit von Zeit. [HLCH12, Seite:3]* . . . 6
4 *Die durch den Roboter zu transportierende Box.* . . . 8
5 *OGRE-Schema und Schnittstelle zwischen dem QT-Framework. In Anlehnung an [Sch05, Abbildung 2].* . . . 12
6 *Grafische Darstellung der Schräglagenkorrektur über den Ansatz der geringsten Standardabweichung.* . . . 15
7 *Tiefenbildaufnahme vor der Korrektur.* . . . 17
8 *Tiefenbildaufnahme nach der Korrektur durch den Ansatz der geringsten Standardabweichung.* . . . 17
9 *Tiefenbildaufnahme nach der Korrektur durch den Ansatz der kleinsten Fehlerquadrate.* . . . 18
10 *Schematische Darstellung der zwei Punkte P_{Z1} und P_{Z2} durch die homogenen Koordinaten (rot), in Abhängigkeit zu der jeweiligen Ebene in Z (In Anlehnung an [Kla10, Abbildung 2.17]).* . . . 20
11 *Modell einer Lochkamera, mit der Gegenstandsweite g, sowie der Bildweite b (In Anlehnung an [Jäh12, Abbildung 3.3])* . . . 21
12 *Modell einer Abbildung durch eine Linse (In Anlehnung an [Jäh12, Abbildung 3.3]).* . . . 22
13 *Schematische Darstellung der intrinsischen Kameraparameter.* . . . 23
14 *Schematische Darstellung der extrinsischen Parameter.* . . . 24
15 *Tonnenförmige- (rechts) und kissenförmige (links) Verzeichnung eines Rechtecks (gestrichelt). In Anlehnung an [Jäh12, Abbildung 3.12].* . . . 26
16 *Aufnahmen im Tiefenbild.* . . . 27
17 *Gegenüberstellung der technischen Zeichnung des Testobjektes und der Aufnahme im Tiefenbild.* . . . 28
18 *Transformation durch den im Kinect SDK v. 2.0 enthaltenen Coordinate- Mapper.* . . . 29
19 *Aufnahme einer Box im Farbbild.* . . . 30
20 *Gegenüberstellung verschiedener Farben im Farb- und IR-Bild.* . . . 30
21 *Schräglagen korrigierte Aufnahme des Tiefenbildes in dem sich beide Transportboxen, klein (mittig) und groß (rechts unten) befinden.* . . . 31
22 *Aufnahme des Tiefenbildes unter Anwendung der Schwellwertgrenze von* 35 mm. . . . 32
23 *Vergrößerte und farblich unterteilte Transportbox, nach Anwendung des Canny-Algorithmus.* 32
24 *Coordinate-Mapper korrigierte Aufnahme des Tiefenbildes unter Anwendung der Schwellwertgrenze von* 15 mm *und des Canny- sowie Sobel-Operators.* . . . 33
25 *Anwendung des Algorithmus aus Gleichung 15 auf das COOM korrigierte und orignale Bild, mit den Algorithmenparametern $min = 10$ und $max = 100$.* . . . 34
26 *Fehlerhaft klassifizierte Objekte.* . . . 39
27 *Gegenüberstellung der Kanäle des RGB- und des HSV-Farbraumes einer Box mit roter Markierung an den Rändern.* . . . 42
28 *Visualisierung des HSV-Raumes. Die untere Farbskala ist in Grad aufgetragen (Normale Skala (oben) / Skala in OpenCV (unten)) [wik].* . . . 43
29 *Gegenüberstellung der HSV-Anteile H und S sowie dem Faktor V.* . . . 44
30 *Darstellung der Auswertung einer Box im Farbbild mithilfe des Ansatzes über die Korrelation.* . . . 45
31 *Darstellung der Vorsegmentierungsschritte in Reihenfolge der Durchführung a,b.* . . . 50
32 *Ortsprofile der Kontur eines menschlichen Armes für drei unterschiedliche Situationen.* 51

33 *Farbliche Gegenüberstellung der Clusterschritte i des K-Means (Mit den Initialisierungensvariablen PP_CENTERS und INITIAL_LABELS) - und des EM- Algorithmus.* . . . 55
34 *Das im Farbbild eingezeichnete Modell des Roboters (Bezogen auf das Weltkoordinatensystem) sowie die durch den Roboter verursachten Cluster (*$1\,px \hat{=} 1\,\mathrm{mm}$ *für die eingezeichneten Rechtecke).* . . . 58
35 *Das im Farbbild eingezeichnete Modell des Roboters, die durch den Roboter verursachten Cluster und die durch einen Menschen verursachten Cluster (*$1\,px \hat{=} 1\,\mathrm{mm}$ *für die eingezeichneten Rechtecke).* . . . 58
36 *Darstellung der klassifizierten und vergrößerten Konturen des Menschen (rechts/rot), des Roboters (links/blau) sowie der Überlagerung beider Konturen (unten/gelb) durch eine Kollision.* . . . 60
37 *Gegenüberstellung verschiedener Situationen, in denen der Roboter in den Arbeitsbereich fährt.* . . . 62
38 *Gegenüberstellung verschiedener Situationen, in denen der Roboter über den Boxen stationiert ist.* . . . 64
39 *Messaufbau für die Erfassung der Kollisionen unter variierender Roboter-Geschwindigkeit und -Beschleunigung sowie unterschiedlichen Konturbreiten.* . . . 65
40 *Verarbeitungskette der Kinect für das Empfangen und Aufbereiten der Frames (in Anlehnung an [Kle16, Abbildung 13]).* . . . 68
41 *Sequenzdiagramm der vollständigen Applikation.* . . . 72
42 *UML- Klassendiagramm der Klasse „framesource“.* . . . 74
43 *UML- Klassendiagramm der Klasse „configurator“.* . . . 75
44 *UML- Klassendiagramm der Klasse „corrector“.* . . . 76
45 *UML- Klassendiagramm der Klasse „armestimator“.* . . . 77
46 *UML- Klassendiagramm der Klasse „collisiondetection“.* . . . 78
47 *UML- Klassendiagramm der Klasse „objectestimator“.* . . . 79
48 *UML- Klassendiagramm der Klasse „RenderEngine“.* . . . 80
49 *UML- Klassendiagramm der Klasse „robotcontroll“ mit Beziehung zu der Klasse „ProgrManager“ von Herrn Frank Hoch [Hoc16].* . . . 81
50 *Konfigurationsprozess* . . . 82
51 *Greif und Validierungsvorgang.* . . . 84
52 *FPS des umgesetzten Systems unter diversen Belastungen.* . . . 87

Tabellenverzeichnis

1 *Geometrische Maße der beiden vorhandenen Transportboxen in Anlehnung an Abbildung 4a.* . . . 8
2 *Gegenüberstellung diverser Framework- und Software-Lösungen. In Anlehnung an [DK16, Abbildung 16].* . . . 9
3 *Gegenüberstellung der High-Level Grafik APIs. += unterstützt, - = nicht unterstützt, o = nur durch Erweiterungen oder Umwege unterstützt. In Anlehnung an [RRA10, Seite: 335].* . . . 11
4 *Messung der Hu-Momente auf Basis der konvexen Hülle für beide Größen der Transportboxen in unterschiedlicher Ausrichtung und Position.* . . . 37
5 *Gegenüberstellung der Zuverlässigkeit der Algorithmen zur Erkennung der Boxen.* . . . 38
6 *Gegenüberstellung der Genauigkeit der Algorithmen zur Erkennung der Boxen im Tiefenbild.* 40
7 *Gegenüberstellung der umgesetzten Algorithmen im Tiefenbild.* . . . 41
8 *Gegenüberstellung der Genauigkeit der Algorithmen zur Erkennung der Boxen im Farbbild für die erste Testsituation.* . . . 47
9 *Gegenüberstellung der Genauigkeit der Algorithmen zur Erkennung der Boxen im Farbbild für die zweite Testsituation.* . . . 47
10 *Gegenüberstellung der umgesetzten Algorithmen im Farbbild.* . . . 48
11 *Messungen mit den erfolgreich verhinderten Kollisionen unter variierter Geschindigkeit, Beschleunigung ($v_R = a_R$) und Konturbreite.* . . . 66
12 *Durchschnittlicher Abstand zwischen den seitlichen Markierungen sowie durchschnittliche Ausrichtungswinkel der Box im Weltkoordinatensystem.* . . . 85
13 *Auflistung der erfolgreichen Transportbewegungen aller acht Boxen, sowie deren Größe.* 85
14 *Ergebnisse der Kollisionserkennung, aus der Beobachtung von 157 Transportaufgaben.* . 86
15 *Zuordnung der Belastungssituationen und der Zeitmessung aus Abbildung* 52. . . . 87

Abkürzungsverzeichnis

AOI Area Of Interest

API Application Programming Interface

COOM Coordinate- Mapper

EM Expectation-Maximization

FAR False acceptance rate

FPS Frames per second

FRR False reject rate

HSV Hue Saturation Value

IR Infrarot

MRK Mensch-Roboter Kollaboration

OGRE Object-Oriented Graphics Rendering Engine

RGB Rot Grün Blau

SDK Software Development Kit

TOF Time-of-Flight

1 Einleitung

Die Industrialisierung und damit verbundene Automatisierung findet ihren Ursprung in der Mitte des 18. Jahrhunderts und ist mitlerweile wesentlicher Bestandteil der Fertigungsprozesse in Unternehmen. Im Kontext der Industrie 4.0 besteht heute der Bedarf nach einem mit dem Menschen kollaborierenden Robotersystem (MRK-System). Die Vorteile bestehen vor allem in dem variablen Automatisierungsgrad sowie dem Zugriff auf die stärksten Ressourcen von Mensch und Maschine. Hierzu gehört die Problemlösungskompetenz des Menschen sowie die konstante Fertigungsqualität der Roboter. Die Einsatzgebiete liegen vor allem in Bereichen mit erhöhten Flexibilitätsanforderungen, d.h mit variierenden Aufgabenstellungen und Roboterpositionen. Der Einsatz kollaborierender Roboter ermöglicht zudem den Verzicht auf sicherheitskritische Elemente wie Sicherheitsbereiche und Zäune. Weiterhin können statische Programmabläufe im Kontext der Industrie 4.0 durch dynamische Abläufe ersetzt werden und eine Vielzahl neuer Möglichkeiten eröffnen. Die Anbindung an ein Cloud-System ermöglicht darüber hinaus ein groß interagierendes Netz, in dem unter Verwendung von „Machine-learning“-Algorithmen bestimmte Verfahren an gleiche oder ähnliche MRK-Systeme übermittelt werden können. Die Systeme wären dann in der Lage auf Grundlage der eigenen Arbeitsumgebung und mit dem Wissen vernetzter Roboter eigenständig Prozesse zu planen und auszuführen [DKML16, Seite: 7-8]. MRK-Systeme ermöglichen zusätzlich wirtschaftlich attraktive Automatisierungsmöglichkeiten für kleinere Produktmargen, da eine statische Automatisierung zu teuer wäre [MDHMN$^+$16, Seite: 9]. In der vorliegenden Arbeit erfolgt die Umsetzung von Teilgebieten der Industrie 4.0, indem der kollaborationsfähige Roboter UR5 der Firma Universal Robots auf der Informationsgrundlage eines „Time of Flight“-Sensors (TOF-Sensor) mit einem Menschen kollaboriert. Das Ziel dieser Arbeit besteht darin, mit einer „Kinect v.2“-Kamera der Firma Microsoft die Position der Arbeitsobjekte und die des Menschen zu erfassen, um eine Kollision zu unterbinden. Dies beinhaltet die Umsetzung der Objekt- und Personenerkennung, der Schätzung von verdeckten Objekten sowie die Ansteuerung des Roboters.

1.1 Problemstellung

Es soll ein MRK-System entstehen, in dem der Roboter an einem Arbeitsplatz agiert, der gleichzeitig von einem Menschen genutzt wird. Das System besteht aus einer Arbeitsfläche in Form eines Tisches auf dem der UR5-Roboter befestigt ist. Die Kamera wird auf horizontaler Tischebene über der Arbeitsfläche platziert. Die zu bewegenden Transportboxen befinden sich auf der Tischfläche, deren Struktur statisch ist. Für die Realisierung des MRK-Systems benötigte dieses die Informationen über die Position von relevanten Objekten sowie von Mensch und Roboter. Während letzteres über die Schnittstelle des UR5-Roboters zur Verfügung steht, werden die anderen Informationen durch einen Kinect v.2 Sensor extrahiert. Der Sensor unterliegt diversen Fremd- und Fehlereinflüssen, die für die Positionserkennung zu berücksichtigen sind. Um eine Verwechslung mit irrelevanten Objekten auszuschließen, müssen die Merkmale der relevanten Objekte analysiert und in geeigneter Form in einen Bildverarbeitungsalgorithmus implementiert werden. Damit ein Greifen durch den Roboter sicher zu realisieren ist, muss die Erkennung der korrekten Objekte hinreichend genau sein. Gleiches gilt für die Genauigkeit zur Umrechnung der Objektposition im Bild in metrische Daten innerhalb des Arbeitsfeldes. Zusätzlich sind Verdeckungen zu erwarten, die durch den Menschen oder den Roboterarm verursacht werden und das darunterliegende Sichtfeld verdecken. Für eine einfache und schnelle Analyse der Tiefeninformationen wird darüber hinaus ein Tool mit entsprechender Eigenschaft benötigt. Dieses soll Tiefeninformationen durch eine „Render-Engine“ zugänglich machen und die Ergebnisse der Objekterkennung veranschaulichen. Dabei ist eine Schnittstelle zwischen dem Roboter, der Kamera und der 3D-Engine zu bilden.
Für eine sichere Bewegung des Roboters werden folgende Dinge benötigt (In Anlehnung an [Ebe03, Seite: 23]) :

- Wissen über die Umgebung in der die Kollaboration stattfindet.
- Wissen über Position und Form des Roboters sowie den kritischen Elementen der Person im Arbeitsfeld.
- Die Fähigkeit eine Kollision zu erkennen und zu vermeiden.

1.2 Aktueller Stand

Bereits heute werden MRK-Systeme für die Umsetzung diverser Aufgaben in der Industrie verwendet. Der Leichtbauroboter (LBR) „iiwa“ der Firma Kuka z.B unterstützt neben einem Mitarbeiter die Montage von Kegelrädern oder Karosserieteilen des Autoherstellers BMW. Diese Systeme verwenden keinen TOF-Sensor als Informationsgrundlage, sondern arbeiten redundante und feste Arbeitsabläufe ab. Das in der Dissertation von Herrn Thorsten Gecks vorgestellte System an der Universität Bayreuth verwendet hingegen mehrere Farb- und Tiefenbildsensoren, um den Arbeitsbereich eines Roboters nach Objekten zu scannen und so eine Roboter-Mensch-Koexistenz und -Kooperation zu sichern [Gec11]. Hierzu wird die dynamische sogenannte „online Trajektorie“ auf Informationsgrundlagen eines Kameranetzwerkes gebildet, indem das Differenzbildverfahren „Change Detection“ verwendet wird. Dabei erfolgt eine Unterteilung der Bildinformationen in den Hinter- und Vordergrund der Umgebung. Im Gegensatz zum dynamischen Hintergrund der zuvor erwähnten Dissertation, existiert in der vorliegenden Arbeit ein statischer Hintergrund, da die Tischstruktur keine Änderung im Tiefenbild erfährt und keine irrelevanten Arbeitsabläufe innerhalb des Arbeitsumfeldes stattfinden.
Eine weitere Dissertation, die sich mit dem Thema beschäftigt, ist die Arbeit von Herrn Jens Bernshausen an der Universität Siegen [Ber11]. In dieser Arbeit wird ein Tiefenbildsensor auf einem mobilen Roboter montiert und eine Selbstlokalisierung anhand definierter Bezugspunkte sowie der Differenzbildung des Tiefenbildes durchgeführt. Bestandteil der Arbeit ist eine Schätzung der z-Ebene auf Grundlage einer 3D-Punktwolke (nach [Goe17]), die ebenfalls für das System dieser Thesis relevant ist. Eine mit der Aufgabenstellung dieser Thesis identische Abhandlung konnte nicht gefunden werden, jedoch ähnliche Problemstellungen. So existieren Lösungen zur Erkennung der Hand eines Menschen im Tiefenbild. Zusätzlich sind die Fremdeinwirkungen und Sensoreigenschaften des Tiefenbildes der Kinect v.2 sehr ausführlich behandelt und in den Arbeiten erläutert [CGMS15, WS16, JS14]. Darüber hinaus verfügt die OpenCv-Bibliothek eine Vielzahl an bekannten Bildverarbeitungsalgorithmen, die der Merkmalsextraktion dienen. Diese sind in der einschlägigen Literatur wie [Jäh12] erläutert . Daher existieren bereits viele Verfahren zur Approximation von Objekten wie Kreise, Rechtecke oder Linien in einem Bild. Die Merkmalsextraktion der Transportboxen ist aufgrund ihrer nicht vollständig rechteckigen Struktur eine für diese Aufgabenstellung individuelle Problematik.
Aus den Dissertationen von Dirk Ebert an der TU Kaiserslautern [Ebe03] und Nils Hofemann an der Universität Bielefeld [Hof06] sind für diese Thesis relevante Informationen zu beziehen. Das in [Ebe03] umgesetzte System untergliedert sich in die Teilgebiete Robotermodellierung, Bildverarbeitung, Kollisionserkennung und Bahnplanung. Die Zielsetzung ist eine konfliktfreie Trajektorienplanung, in einem mit Hindernissen versehenem Arbeitsumfeld unter Verwendung mehrerer stationär platzierten Farbbildkameras, zu realisieren. Es findet dabei die globale Kollisionserkennung ohne Distanzberechnung statt. Das bedeutet, dass alle (globalen) Hindernisse auf eine Kollision (Schnittmenge) mit dem Roboter geprüft werden und die Kollisionsfrage binär beantwortet wird. Eine Kollisionsfrage ist zu verneinen, sofern eine der Kameras keine Schnittmenge zwischen Roboter- und Hindernissilhouette erfasst [Ebe03, Seite:74]. Dies geschieht auf Grundlage der als Gitter modellierten Umgebung, indem die Silhouette aller Objekte im Bild durch das Differenzbildverfahren erfasst wird. Anschließend erfolgt eine Einteilung der Gitteranteile in Vorder- sowie Hintergrund unter Verwendung eines linearen Klassifikators. Für eine Kollisionsvermeidung wird eine „Bounding-Box“ über die Trajektorie und entsprechend der möglichen Armwinkel des Roboters verschoben, um die infrage kommende Trajektorie auf eine mögliche Kollision mit dem Vordergrund zu prüfen.

2 Arbeitsumfeld

2.1 Technischer Aufbau

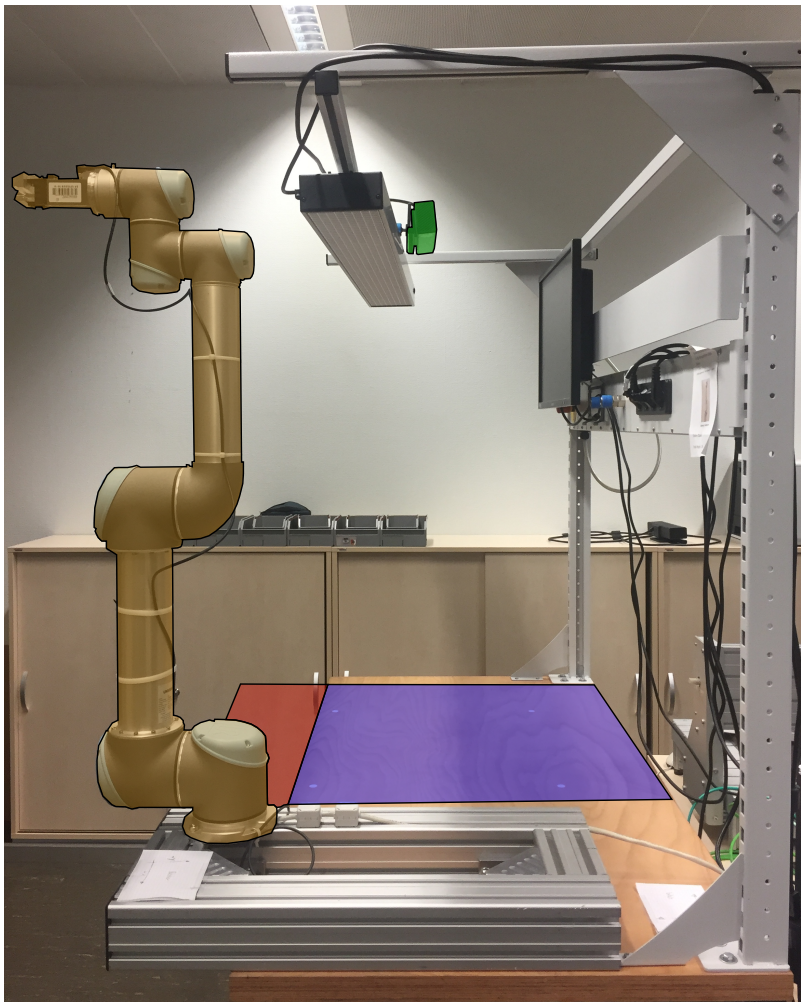


Abbildung 1: Projektumgebung mit der Kinect (grün), dem Arbeitsbereich (blau), dem Sicherheitsbereich (rot), sowie dem UR5 Roboter (gelb).

Abbildung 1 zeigt die vorhandene Arbeitsumgebung, in der die Problemstellung gelöst werden soll. Zu sehen ist der UR5-Roboter (gelb) in der Home-Position, der Arbeitsbereich (blau), in dem die Transportboxen durch den Roboter und durch den Menschen bewegt werden sollen, die Kinect (grün) sowie der Sicherheitsbereich (rot). Der Sicherheitsbereich beschreibt den Bereich, in dem sich der Benutzer aufhalten wird und aus dem Bereich heraus in den Arbeitsbereich einwirkt. Zu erkennen ist, dass der Roboter auf einem Alu-Profil installiert ist und nicht direkt auf der Tischplatte. Weiterhin befindet sich links neben der Kinect eine Leuchtstofflampe, die den Arbeitsbereich unter der Netzfrequenz von 50 Hz ausleuchtet. Die Kinect wird so positioniert, dass sie auf den Arbeitsbereich gerichtet ist. Es ist zu erkennen, dass der Roboter während der Arbeiten in dem Arbeitsbereich das Sichtfeld der Kinect beeinträchtigen wird und darunterliegende Objekte verdeckt werden.

2.2 Kinect-Sensor

Der Kinect-Sensor ist ein Tiefenbildsensor, der seinen Ursprung im Consumer-Bereich, vorwiegend in der digitalen Unterhaltung der XBox 360 Konsole der Firma Microsoft, findet. Aufgrund seines geringen Preises handelt es sich um eine attraktive Alternative gegenüber Industrielösungen. Der Sensor verfügt über zwei Bildsensoren:

Infrarot Die Infrarotkamera wird zur Erfassung der Entfernung verwendet und besitzt eine Auflösung von 512x424 Pixeln sowie eine Bildwiederholungsrate von 30 $\frac{\mathrm{B}}{\mathrm{s}}$. Unter schlechten Lichtverhältnissen wird die Rate auf 15 $\frac{\mathrm{B}}{\mathrm{s}}$ reduziert.

Farbe Die Farbbildkamera weist eine Auflösung von 1920x1080 Pixeln auf und bei ausreichender Helligkeit eine Bildwiederholungsrate von 30 $\frac{\mathrm{B}}{\mathrm{s}}$. Unter schlechten Lichtverhältnissen wird die Rate ebenfalls auf 15 $\frac{\mathrm{B}}{\mathrm{s}}$ reduziert.

In der vorliegenden Arbeit findet der Sensor der zweiten Generation Anwendung, der gegenüber der ersten Version nicht mit einem IR-Muster, sondern nach dem TOF-Prinzip arbeitet.

2.2.1 „Time of Flight"-Funktionsprinzip

Das „Time of Flight" (TOF)-Prinzip basiert auf der Messung der Dauer, die ein Lichtstrahl zwischen Aussenden und Eintreffen auf den Sensor benötigt. Diese Zeit variiert je nach Distanz zu einem Objekt und erlaubt einen Rückschluss auf dessen Entfernung. Dabei werden die folgenden zwei Verfahren unterschieden:

Puls-Verfahren Innerhalb einer definierten Frequenz werden rechteckige Infrarotsignale versendet und die Zeit bis zum Eintreffen des reflektierten Lichts ermittelt.

Moduliertes Licht Es wird die Phasendifferenz eines in der Stärke sinusoidal oder trapezförmig variierendes und dauerhaft ausgestrahltes Licht beim Emittieren und Empfangen gemessen.

Die Kinect in der zweiten Version verwendet das TOF-Prinzip mit dem modulierten Licht. Mit einer Frequenz von f wird hierzu ein trapezförmiges Infrarotsignal versendet, dessen Wegdauer d anhand der ausgehenden und eingehenden Phasenlage bestimmt wird.

$$d = \frac{c}{2f} \cdot \frac{t_d}{2\pi}$$

Dabei ist die Laufzeit des Lichtes t_d von der Ladungsdifferenz der Kanäle A und B des Sensors abhängig:

$$t_d = arctan(A - B) \tag{1}$$

Die Kanäle A und B werden dabei invers angesteuert und sind von der Frequenz des Taktgebers abhängig. Dieses Prinzip ist der Abbildung 2 zu entnehmen.

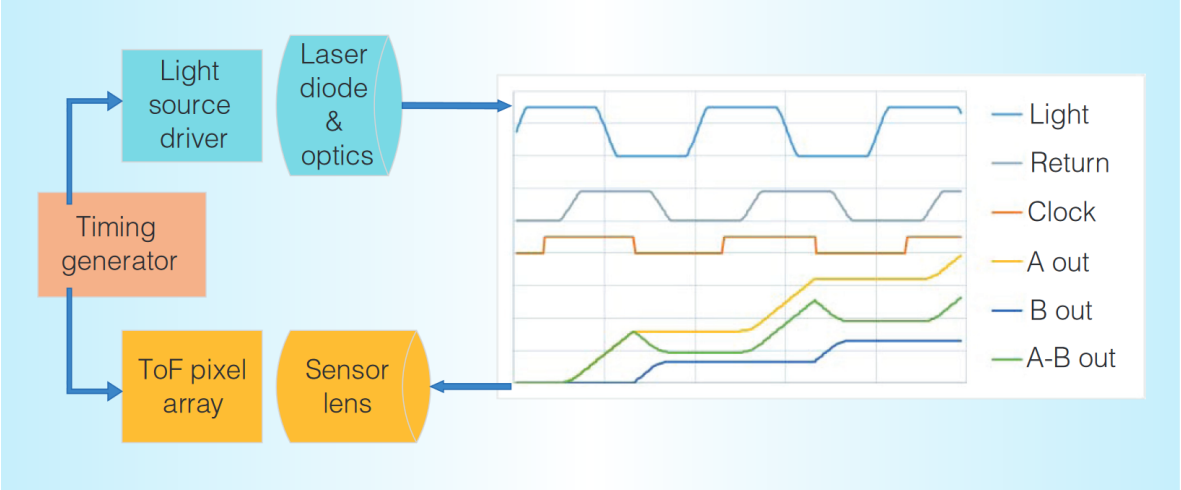


Abbildung 2: Signalverlauf des ausgehenden (Light) und eingehenden Lichts (Return) und die erzeugte Ladung (A und B). [JS14, Seite:50]

Der Abbildung 2 sind die Signalverläufe für das ausgehende Licht (Light), das durch das Objekt reflektierte und in der Amplitude gedämpfte Licht (Return) sowie die beiden Kanäle A und B, deren Ladungsaufnahme von dem Taktsignal (Clock) abhängig ist, dargestellt. Hierbei erfolgt die Aufnahme von Photonen während eines positiven Clocksignals von Kanal A und während der Nullphase von Kanal B. Das selbe Verfahren ist in Abbildung 3 dargestellt, während hierbei die Ladungsträger verdeutlicht werden. Anhand der Ladungsdifferenz zwischen beiden Phasen ist nach Gleichung 1 auf die Laufzeit rückzuschließen.

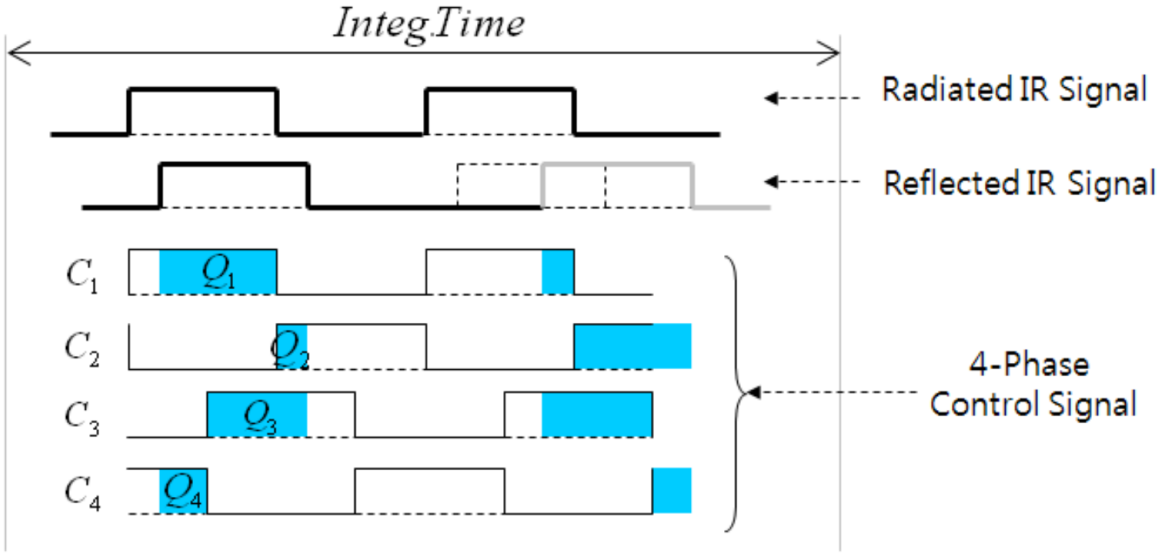


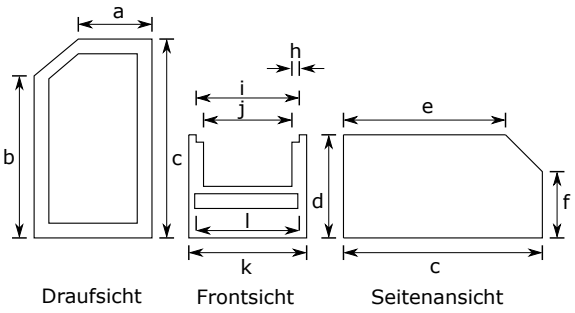
Abbildung 3: Signalverlauf des ausgehenden Lichts (Radiated IR Signal), des eingehenden und verzögerten Lichts (Reflected IR Signal) sowie die Ladungen Q_1 und Q_2 in Abhängigkeit von Zeit. [HLCH12, Seite:3]

2.3 UR5-Roboter

Für den Transport der Arbeitsobjekte findet der Roboter UR5 der Firma Universal Robots Anwendung. Dieser kann bis zu 5 kg Traglast bewegen und besitzt eine Reichweite von 850 mm. Er weist einen Freiheitsgrad von sechs auf, in welchen eine Rotationsgeschwindigkeit von $\frac{180°}{s}$ erreicht wird [Rob05]. Eine wichtige Eigenschaft stellt die Sicherheitsfunktion „Force Sensitivity“ dar, die eine Zertifizierung des Roboters nach „EN ISO 13849:2008 PL d“ erlaubt und dem Roboter das Arbeiten in der Nähe von Personen erlaubt [Hoc16, Seite: 5]. Die Sicherheitsfunktionen bestehen aus der Geschwindigkeitsreduzierung, die durch externe Sensoren ausgelöst wird, wenn definierte Objekte den Sicherheitsbereich des Roboters betreten. Zusätzlich erfolgt eine Überwachung der Motorströme, um die Kräfte der Gelenke ohne zusätzliche Sensoren zu erfassen. Über die Widerstandskraft wird eine Kollision erkannt und eine entsprechende Kollisionsstrategie ausgeführt [dMG+17, Seite: 103]. Weitere Robotersysteme mit Zertifizierung für eine Roboter-Mensch-Kollaboration sind die Roboter „Baxter“ und „Sawyer“ der Firma Rethink Robotics, der Leichtbauroboter „iiwa“ der Firma Kuka sowie der 2016 vorgestellte Roboter „Franka Emika“ [dMG+17, Seite: 103].
Die Ansteuerung des UR5-Roboters erfolgt im Allgemeinen durch eine grafische Benutzeroberfläche, die auf dem Touch-Panel des Roboters ausgeführt wird. Durch die Masterthesis von Frank Hoch an der HAW Hamburg [Hoc16], besteht zusätzlich die Möglichkeit online Trajektorien durch eine Skript-Injektion via TCP/IP zu nutzen. Grundlegende Ansteuerungen wie „Power on“ oder das Durchführen zuvor definierter Programme mithilfe des „teach“-Verfahrens können ebenfalls über TCP/IP durch den „Dashboard“-Server und dem „Secondary Client Interface“ des Roboters genutzt werden.

2.4 Transportobjekte

Der Roboter soll zu einem späteren Zeitpunkt diverse Aufgabenstellungen mit Transportobjekten umsetzen. Bei den Transportobjekten handelt es sich um zwei unterschiedlich große und in ihrer Form identische Transportboxen, die in Abbildung 4a beschrieben sind. Die geometrischen Maße sind der Tabelle 1 zu entnehmen. Für einen Transport der Boxen verfügt der Roboter über ein entsprechendes Werkzeug, um eine Box im Bereich der Länge l zu fixieren. Die Aufnahme in Abbildung 4b zeigt das Werkzeug mit einer fixierten Box. Auffällig ist der schmale Greifbereich, der einen genauen Greifwinkel voraussetzt, um eine Box ohne Schäden zu greifen.



(a) Vereinfachtes Konstruktionsschema der Transportboxen.

(b) Aufnahme des Werkzeugs (links) mit einer fixierten großen Box (rechts).

Abbildung 4: Die durch den Roboter zu transportierende Box.

Größe	Große Box in [mm]	Kleine Box in [mm]
a	148	100
b	220	143
c	245	175
d	102	102
e	216	134
f	65	65
g	245	166
h	6	6
i	168	117
j	156	105
k	173	122
l	156	105

Tabelle 1: Geometrische Maße der beiden vorhandenen Transportboxen in Anlehnung an Abbildung 4a.

2.5 Auswahl der Programmierumgebung und Bildverarbeitungsbibliothek

Lösung	OpenCv	Kommerz. SW	Matlab	ImageJ
Fkt. Umfang	++	++	+	+
Performance	++	++	+	-
Grafische Oberfläche	-	++	o	O
Support	o	++	++	o
Interaktives Testen	-	-	++	+

Tabelle 2: Gegenüberstellung diverser Framework- und Software-Lösungen. In Anlehnung an [DK16, Abbildung 16].

Für die Verarbeitung von Bildinformationen stehen diverse Framework- und Software-Lösungen zur Verfügung die in Anlehnung an [DK16, Abbildung 16] in Tabelle 2 gegenübergestellt werden. Ihr kann entnommen werden, dass eine kommerzielle Software die meisten Vorteile aufweist. Diese ist jedoch mit Lizenzgebühren verbunden und erschwert die Fortführung der Arbeit in anderen Gruppen, da diese ebenfalls eine Lizenz benötigen. Wenige Vorteile bietet das Java Programm „ImageJ“, das sowohl durch den Support als auch der Performance wenig punkten kann. Die Realisierung über Matlab erfordert ebenfalls Lizenzgebühren, sofern die Software nicht im Rahmen der Lehre angewendet wird. Weiterhin sind zu dem Standardpaket von Matlab diverse Zusatzpakete nötig, die weitere Lizenzgebühren verlangen. Daher wurde das OpenCv-Framework als Bildverarbeitungssoftware ausgewählt. Dieses System bietet eine hohe Performance bei gleichzeitig hohem Funktionsumfang. Zudem ist das Framework durch eine „3-clause BSD License“ geschützt, die eine Weitergabe der Software erlaubt, ohne dafür Gebühren zu verlangen.

OpenCV-Framework

Das OpenCv-Framework ist eine „Open-source“-Bibliothek, die der Auswertung und Verarbeitung von Bildinformationen dient und ursprünglich durch Intel im Jahr 1999 entwickelt wurde. Die Zielsetzung bestand darin, eine portierbare, optimierte und kostenlose Software anzubieten, die in der Lage war den damaligen neuen Anforderungen in der Bildverarbeitung gerecht zu werden. Bereits im Jahr 2012 besaß die Bibliothek mehr als 2500 optimierte Funktionen und wurde von mehr als 2,5 Millionen Menschen heruntergeladen. [CAP+12, Seite: 1725] Daher existiert heute eine Vielzahl an Literatur zur Verwendung der Software. Nach zwei größeren Updates in der Version 2.0 im Jahr 2009 und 3.0 im Jahr 2015, verfügt die Software heute in der Version 3.2 über Funktionen zur Objekt- und Menschenerkennung, Objektverfolgung 3D-Rendering und Stereobild-Verarbeitung [tea]. Die Bibliothek besitzt jedoch keine Möglichkeit eine grafische Oberfläche zu realisieren und muss um eine weitere Software ergänzt werden, wozu sich das QT-Framework anbietet.

QT-Framework

Das QT-Framework findet seinen Ursprung im Sommer 1990, in dem Haarvard Nord und Eirik Chambe-Eng mit Ultraschallbildern arbeiteten. Hierzu benötigten Sie eine für Unix-, Macintosh-, und Windows funktionierende Grafikoberfläche, die auf dem Schema der objektorientierten Programmierung basieren sollte. 1995 erschien dessen Umsetzung QT in der Version 0.90 für Mac und Linux sowie ein Jahr später mit der Unterstützung für Windows[Wela]. Das C++ basierende Framework zeichnet sich heute durch seine objektorientierte und einfache Umsetzung anspruchsvoller Benutzeroberflächen aus. Weiterhin ist die Verwendung für nicht kommerzielle Anwendungen kostenlos und kann mit weiteren Hochsprachen wie Python, SQL, Perl oder OpenGL kombiniert werden. Das Framework arbeitet nach einem Signal-Empfänger-Prinzip, bei dem Objekte definierte Signale an Empfänger anderer Objekte senden können. Eine Kommunikation zwischen zwei Objekten ist daher sehr einfach für unterschiedliche Datentypen zu realisieren. In Kombination mit dem QT-Creator als Entwicklerumgebung können Benutzeroberflächen

einfach per „Drag and Drop“ erstellt werden und über das Signal-Empfänger-Prinzip miteinander agieren. Seit 2012 existiert QT in der Version 5.0 und wird von mehr als 500.000 Entwicklern genutzt [Wela].

2.6 Auswahl des 3D-API

Um ein geeignetes 3D-Applikations Interface zu ermitteln ist im ersten Schritt eine Anforderungsanalyse zu erstellen. Die Beurteilung erfolgt anschließend auf Grundlage der für die Anwendung relevanten Kriterien. Für eine grafische Darstellung der Tiefenbildinformationen muss die API (Application Programming Interface) eine Punktwolke visualisieren können. Weiterhin sollen folgende Objekte farblich hervorgehoben werden:

Zylinder Zur Darstellung der Roboterelemente und der menschlichen Gliedmaßen.

Kugeln Zur Visualisierung der Gelenke des Roboters.

Linien Zur Darstellung der geplanten Trajektorien.

Quader Zur Darstellung der Grenzbereiche. Beispielsweise für den durch die Person verursachten Sperrbereich.

Weitere Auswahlkriterien sind die Komplexität der Implementierung, Interaktionsmöglichkeiten, die Geschwindigkeit und die vorhandene Literatur. Da für die Umsetzung in jeden Fall eine Kompatibilität zu QT vorhanden sein muss, wird die Auswahl auf die von QT unterstützen API´s ([Welb]) begrenzt. Als weiteren Entscheidungshilfen dienen die Ergebnisse der Federal University of São Carlos [RRA10]. In deren Analyse aus dem Jahr 2010, sind Graphic Engines mit den folgenden Mindesteigenschaften gegenübergestellt:

- Stabiles Release vorhanden
- Keine Lizenzgebühren
- Source Code steht öffentlich zur Verfügung
- Windows- und Linux-Unterstützung
- Nicht nur für einen speziellen Spiele-Typ geeignet
- Dokumentation und Forum sind vorhanden

Die für die Anwendung relevanten Eigenschaften sind in Tabelle 3 zusammengefasst. Es ist anzumerken, dass Teile der Informationen um Aktuellere ergänzt wurden.

API	OGRE3D	Irrlicht	OSG	Panda 3D	QT3D
Splines	+	+	o	+	-
Klasse für Geometrische Objekte	o	+	+	-	-
GUI	+	+	o	+	+
DirectX	+	+	-	o	+
OpenGL	+	+	+	+	+
Algorithmus zur Wegfindung	+	-	-	-	-
Punktwolke	o	+	+	o	-

Tabelle 3: Gegenüberstellung der High-Level Grafik APIs. += unterstützt, - = nicht unterstützt, o = nur durch Erweiterungen oder Umwege unterstützt. In Anlehnung an [RRA10, Seite: 335].

Die Gegenüberstellung zeigt, dass OGRE3D (Object-Oriented Graphics Rendering Engine) die meisten und QT3D die geringsten relevanten Funktionen bietet. Da QT3D erst seit QT 5.5 (Veröffentlichung 2015) Bestandteil des Framework ist, handelt es sich um die jüngste Grafik API und mit dem Update durch QT5.8 (2017) gleichzeitig um die Aktuellste. Irrlicht bietet ebenfalls eine Vielzahl an Funktionen und soll in Betracht gezogen werden. Panda3D und OSG befinden sich im Mittelfeld, während OSG häufig in technischen Programmen Anwendung findet, handelt es sich bei Panda3D um eine Spiel-Engine. Aus diesem Grund werden im ersten Schritt OGRE3D und Irrlicht aufgrund der Vielzahl an Funktionen sowie OSG aufgrund des technischen Hintergrundes näher betrachtet. Ein weiterer grundlegender Entscheidungspunkt ist die Dokumentation und Community der APIs, da sie eine schnelle Einarbeitung und Problemlösung ermöglichen. Recherchen der drei APIs haben gezeigt, dass sich die OGRE3D-Community gegenüber der OSG-Community minimal aktiver zeigt, jedoch über eine sehr gute Dokumentation verfügt [Alf, scr, Que]. Einige Foreneinträge der deutschen Irrlicht-Community sind aus dem Jahr 2016, die meisten allerdings noch älter [1], weswegen die Community als gering aktiv einzustufen ist, während sich die englische Community weiterhin aktiv zeigt und eine englische, wenngleich knappe Dokumentation existiert [Que]. Im Weiteren wird OGRE3D aufgrund der ausführlichen Dokumentation und dem aktiven Forum sowie OSG durch den technischen Hintergrund umgesetzt. Die Umsetzung der der OSG-Engine hat gezeigt, dass während der Installation eine Vielzahl an Problemen auf dem Entwicklungsrechner entstehen. Aus diesem Grund fiel die Entscheidung aus Zeitgründen für die OGRE-Engine aus. Durch eine vollständige Dokumentation kann diese innerhalb weniger Wochen und ohne Vorkenntnisse vollständig in eine QT-GUI integriert werden. Die Größe des Darstellungsfensters ist dabei vollständig durch die QT-Umgebung skalierbar und eine Interaktion durch Tastatur- und Mauseingaben realisierbar. Es ist anzunehmen, dass Irrlicht und OSG ebenfalls für die Umsetzung der Visualisierung geeignet sind, jedoch mit der gut dokumentierten OGRE-Engine schnellere Erfolge zu erwarten sind.

2.6.1 OGRE-Engine

OGRE ist eine in C++ geschriebene, objektorientierte und szenen-basierte 3D-Engine. Ihren Ursprung findet sie im Jahre 1991 in dem DIMClass-Projekt des Gründers Steve Sinbad und befindet sich seit 2015 in der Version 2.1. Sie ist ab der Version 1.7 unter der MIT-Lizenz (vorher LGPL) geschützt und ist „open source“ geschützt. Voraussetzung für die Verwendung ist, dass in der Software ein definierter Lizenztext enthalten ist.
Das Szenen basierte System ist wie folgt strukturiert und in das QT-Framework integriert:

[1] Überprüft im Juli 2017. Der Reiter „Aktive Themen“ hat in den letzten 6 Monaten keine Einträge gefunden.

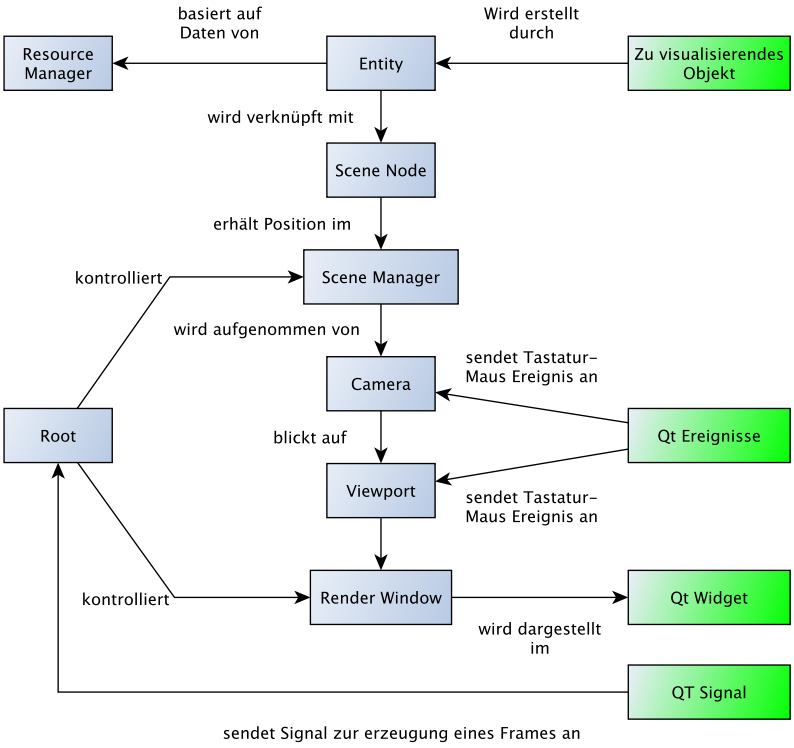


Abbildung 5: OGRE-Schema und Schnittstelle zwischen dem QT-Framework. In Anlehnung an [Sch05, Abbildung 2].

Blaue Elemente definieren die OGRE-Engine und Grüne das QT-Framework bzw. den allgemeinen Programmanteil. Die OGRE-Struktur basiert dabei auf einem „root“, der die Verwaltung der Engine übernimmt und in jedem Programm einmalig ist. Soll ein Objekt visualisiert werden, erfolgt der Eintrag einer „Entity“, welche für jedes zu rendernde Objekt erstellt wird. Ihre visuellen Eigenschaften basieren dabei auf den Einträgen im Resource-Manager, der z.B. Texturen und Darstellungen der Oberflächen verwaltet. Im Anschluss erfolgt die Verknüpfung mit einer „Scene Node“. Einer „Scene Node“ können dabei ein oder mehrere Objekte angehängt werden um eine Szene zu beschreiben. Die Position der Szene wird durch den Szenen-Manager vorgegeben, dem ebenfalls mehrere Szenen zugeordnet werden können. Für die Darstellung werden eine oder mehrere Kameras platziert, deren Sichtfeld und Ausrichtung durch den Viewport bestimmt werden. Mit einem Signal an das „root“-Objekt erzeugt der Render ein Frame, dessen Darstellung im zuvor verknüpften QT-Widget erfolgt. Tastatur- und Mausereignisse werden vom QT-Framework erfasst, um im Anschluss Änderungen an der Kamera und dem Viewport vorzunehmen.

3 Anforderungsanalyse

Die Auswahl der zu untersuchenden Algorithmen basiert auf der Anforderungsanalyse an das System. In der Kollaboration von Maschine und Mensch steht die Sicherheit der Person an vorderster Stelle, weshalb eine schnelle Reaktionszeit, optimalerweise in Echtzeit, von dem System erwartet wird. Durch eine hohe Reaktionszeit kann die Geschwindigkeit des Roboters erhöht werden, da die Kollisionswahrscheinlichkeit mit einer schnellen Reaktionszeit abnimmt[Ebe03, Seite:4]. Da die Kinect mit einer maximalen Bildwiederholrate von $30\,\frac{\mathrm{B}}{\mathrm{s}}$ arbeitet, definiert diese das maximal zu erreichende Ziel. Für den Anwender bedeutet dies, dass das System alle $33,\overline{3}\,\mathrm{ms}$ auf eine Veränderung im Bild reagieren kann, wenn die Programmkette vor Eintreffen des nächsten Frames abgeschlossen ist. Darüber hinaus muss die Positionsbestimmung des Greifpunktes einer Kiste hinreichend genau erkannt werden. Im Farbbild beträgt die Abweichung eines Pixel auf Höhe der Griffposition einer Box bereits mehr als drei Millimeter. Sofern zu hohe Abweichungen für das Greifen vorliegen, kann ein langsames Heranfahren an die Griffposition realisiert werden, indem ein Verschieben der Kiste toleriert wird. Für den Greifvorgang ebenfalls von Bedeutung ist die korrekte Erfassung der Ausrichtung einer Box. Da der Griffpunkt sehr schmal ist, muss das Werkzeug des Roboters die Box im korrekten Winkel greifen. Andernfalls kann es zu einer Beschädigung an der Box kommen. Um eine Kollision zwischen Roboter und Mensch zu verhindern, wird die Position der Gliedmaßen in der Arbeitsumgebung erfasst und mit der Position des Roboters abgeglichen. Bei der Erfassung der Person ist daher sicherzustellen, dass alle Teile erkannt werden und die Positionsschätzung hinreichend genau ist. Hinreichend bedeutet, dass eine Vergrößerung der menschlichen Körperteile zu einem kollisionsfreien Stopp des Roboters führt, sobald eine Schnittmenge zwischen Beiden entsteht. Die Genauigkeit kann daher geringer sein, wenn diese durch eine Vergrößerung der Armbereiche kompensiert wird. In diesem Fall wird der Arbeitsbereich des Roboters eingeschränkt. Zusätzlich werden fremde Objekte in der Arbeitsumgebung vorerst ignoriert. Gleichzeitig wird eine Möglichkeit geboten, diese nachträglich in der Kollisionsvermeidung zu beachten. Durch die Arbeiten des Roboters im Sichtfeld der Kamera ist zu erwarten, dass die Objekte auf dem Arbeitsplatz verdeckt werden. In diesen Situationen ist eine Positionserfassung nicht möglich, weshalb die Box weiterhin als „existierend“ angenommen wird, solange das Sichtfeld bedeckt ist. Entgegen der zwei vorhandenen Ansätze steht für diese Aufgabenstellung nur ein Sensor zur Verfügung. Das bedeutet, dass ein Eintreten des Menschen in den Arbeitsbereich sicher erkannt werden muss, selbst wenn der Roboter das Sichtfeld vollständig verdeckt. Dieser Teil ist durch den technischen Aufbau softwareseitig nicht zu lösen, weswegen das Arbeitsfeld des Roboters hinreichend eingeschränkt werden muss, um ein Eintreten in den Arbeitsbereich durch den Menschen zu erkennen. Für die Darstellung der genannten Prozesse werden der Roboter, die Boxen und der Mensch visualisiert und gemeinsam mit der Punktwolke dargestellt. Eine exakte Visualisierung mit einer hohen Genauigkeit ist nicht erforderlich, da sie nur der Veranschaulichung dient. Die vorliegende Arbeit umfasst keine Planung der Roboter-Trajektorie, die eine Kollisionsvermeidung mit Objekten und Mensch beinhaltet. Aus diesem Grund soll der Roboter unter Annahme einer bevorstehenden Kollision seinen Arbeitsauftrag pausieren bis dieser sicher weiter ausgeführt werden kann. Zusätzlich ist davon auszugehen, dass durch die fehlende Kollisionsvermeidung eine weitere folgende Arbeit entstehen wird, in der genaue Positionen von den Boxen und der Person von Vorteil sind, da die Trajektorienplanung effizienter gestaltet werden kann. Sprünge in der Positionsbestimmung sind für weitere Arbeiten ebenfalls von Nachteil. Weiterhin ermöglicht ein ausführlich kommentierter Code und eine gute Dokumentation die schnelle Einarbeitung in das existierende Programm.

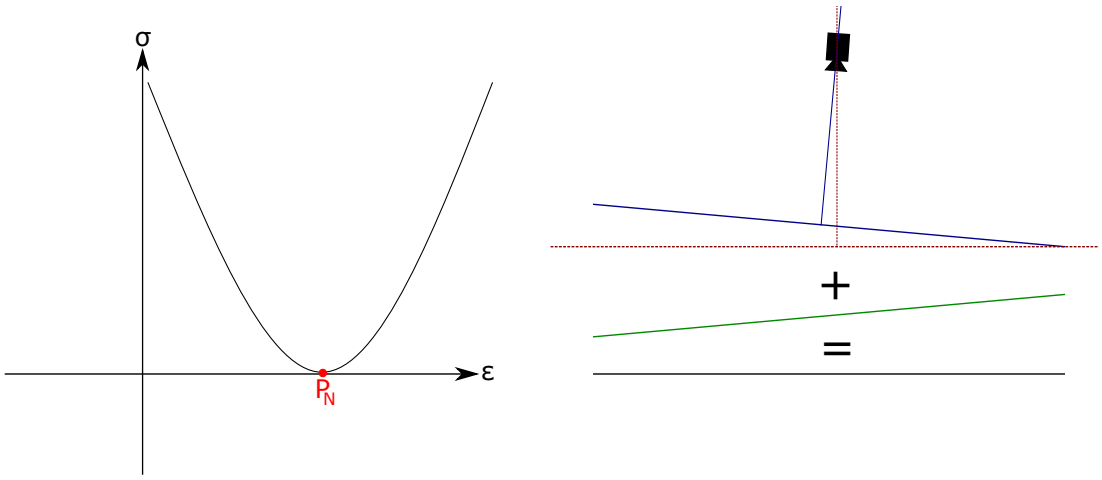
Im Wesentlichen lassen sich somit die folgenden Anforderungen an das System definieren:

1. Performant, um eine schnelle Reaktionszeit und eine höhere Robotergeschwindigkeit zu erlauben.
2. Sicher, damit eine Schnittmenge zwischen dem Roboter und einer Person rechtzeitig zu einem Stopp des Robotors führt.
3. Genaue Erkennung der Ausrichtung der Griffposition um die Boxen nicht zu beschädigen.
4. Erkennung der außenliegenden Kanten einer Box für eine zukünftige Kollisionsvermeidung zwischen dem Roboter und der Box.
5. Variabler technischer Aufbau.
6. Gute Dokumentation und Ausarbeitung.

4 Schräglagenkorrektur

Eine exakt horizontale Montage des Sensors über den Arbeitsbereich ist ohne hinreichend genaue Einmessung nicht zu realisieren. Da bereits geringe Abweichungen gegenüber der horizontalen Ebene, in Abhängigkeit der Entfernung zum Arbeitsbereich, zu einer Abweichung in der Tiefeninformation führen (vgl. Abbildung 6b), wird eine Korrektur der Schräglage durchgeführt.

4.1 Lösungsansatz durch Minimierung der Standardabweichung



(a) Verlaufsschema der Funktion $\sigma(\varepsilon) = f(x)$ mit der Varianz σ, dem Maß der Schrägheit ε und dem Wendepunkt der Funktion P_N.

(b) Kamera, die auf den Boden (rot- -) zeigt und durch die Schräglage (blau–) versetzte Tiefenwerte erfasst. Eine Addition mit der invertierten Ebene (grün–), erzeugt die planare Ebene (schwarz –).

Abbildung 6: Grafische Darstellung der Schräglagenkorrektur über den Ansatz der geringsten Standardabweichung.

Da die Arbeitsfläche als planare Ebene existiert, kann über diese Information eine Schräglage korrigiert werden. Als Messgröße wird dabei die Standardabweichung σ eines mittleren Bildausschnittes verwendet. Die Standardabweichung repräsentiert dabei ein Maß an Varianz in den Pixelwerten und ist definiert durch [DK16, Seite: 139]:

$$\sigma = \sqrt{\sum_{Z=Z_u}^{Z_0} (z - \bar{z})^2 \cdot h(z)}$$

mit dem mittleren Grauwert des Bildes $\bar{z}$, der sich aus allen Pixeln zusammensetzt, dem Grauwert eines bestimmten Pixels z und dem normalisierten Histogramm $h(z)$. Besteht nun zwischen dem mittleren Grauwert $\bar{z}$, welcher den mittleren Tiefenwert repräsentiert, gegenüber dem aktuell zu untersuchenden Pixel eine große Differenz bzw. eine starke Unebenheit, resultiert aus $(z - \bar{z})^2$ ein hoher Wert und somit eine höhere Varianz. Ist die Tiefeninformation in z nahe dem mittleren Wert (Es ist kaum eine Unebenheit vorhanden), resultiert eine geringe Varianz. Eine Varianz von Null sagt aus, dass eine planare Ebene vorhanden ist, da alle Tiefenwert-Informationen in dem Bildausschnitt identisch bzw. nah beieinander liegen. Das Optimierungskriterium ist daher die Standardabweichung, mit dem Optimum Null.
Die Schräglagenkorrektur muss sowohl in x- als auch in y-Richtung erfolgen und kann durch die

Subtraktion einer invertierten Matrix mit identischer Schräglage erfolgen (vgl. Abbildung 6b). Um eine identische und invertierte Matrix mit der Schräglage zu finden, wird im ersten Schritt die Steigung zwischen zwei Punkten in x- und y-Richtung erfasst und auf dessen Grundlage eine Matrix mit der Größe des Messbereichs erzeugt. Die Subtraktion der invertierten Matrix von dem Original erzeugt eine Korrektur in beide Richtungen. Vor und nach der Korrektur wird die Standardabweichung σ der Matrix erfasst und die Differenz gebildet. Nach einer Division mit ε stellt sie die Steigung der Funktion $\sigma(\varepsilon) = f(\varepsilon)$ dar (vgl. Abbildung 6a). Dabei repräsentiert ε das Maß der Schrägheit. Wird auf Grundlage der Steigung die Schräglage der nächsten Iteration in entsprechender Richtung verändert, bewegt sich der Algorithmus in das Minimum der Funktion $\sigma(\varepsilon)$ (In Abbildung 6a der Punkt P_N) und somit in die geringste Standardabweichung, die eine planare Ebene beschreibt.

4.2 Lösungsansatz durch Minimierung der Fehlerquadrate

Ein bekannter Ansatz ist die Achsen-parallele Approximation, aus der die Ebenengleichung in der Koordinatenform

$$ax + by + cz = 0 \tag{2}$$

approximiert wird. Voraussetzung ist, dass der Schwerpunkt der Punktwolke im Koordinatenursprung liegt, sodass der Mittelwert in alle drei Richtungen Null ist [Goe17, Lemma 1]:

$$\bar{x} = \bar{y} = \bar{z} = 0$$

Für den Abstand d zwischen dem Tiefenwert der Ebene, die approximiert werden soll $z_e = f_e(x, y)$ und dem Tiefenwert eines Pixels der Kinect z, resultiert [Goe17, Seite: 4]:

$$d_i = z_i - f_e(x_i, y_i)$$

Je näher sich die Approximation der Ebenengleichung den Werten der Kinect annähert, desto geringer ist der Abstand d_i. Daraus folgt, dass die Summe der Abstände für die Pixel [Goe17, Seite: 4]:

$$\Delta = \sum_{i=0}^{n} d_i^2$$

zu minimieren ist. Die Lösung erfolgt durch den Ansatz nach [Goe17, Seite: 4]:

$$z = f_e(x, y) = ax + by + d$$

Durch die Verschiebung der Punktwolke in den Koordinatenursprung gilt $d = 0$ und es resultiert nach Quadrieren der zu minimierenden Distanz d_i die Gleichung [Goe17, Seite: 5]:

$$\Delta = \sum_{i=0}^{n} d_i^2 = \sum_{i=0}^{n} (z_i - (ax_i + by_i))^2 = \|z - (ax - by)\|^2 \tag{3}$$

Für a und b gilt nach [Goe17, Seite: 6]:

$$a = \frac{G_x}{G}, \qquad b = \frac{G_y}{G}$$

mit:

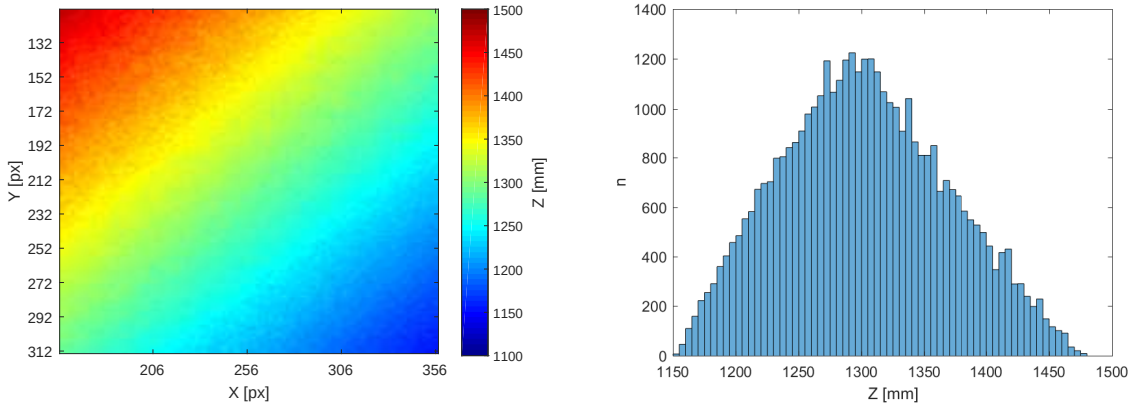
$$G_x = \begin{vmatrix} \mathbf{xz} & \mathbf{xy} \\ \mathbf{yz} & \mathbf{yy} \end{vmatrix}, \qquad G_y = \begin{vmatrix} \mathbf{xx} & \mathbf{xz} \\ \mathbf{yx} & \mathbf{yz} \end{vmatrix}, \qquad G = \begin{vmatrix} \mathbf{xx} & \mathbf{xy} \\ \mathbf{yx} & \mathbf{yy} \end{vmatrix}$$

dabei beschreiben $\mathbf{x}, \mathbf{y}, \mathbf{z}$ Komponentenvektoren, d.h für $\mathbf{x}$ gilt $\mathbf{x} = (x_1 ....x_n)$. Eingesetzt in die Ebenengleichung mit $d = 0$ folgt daraus [Goe17, Gleichung **]:

$$G_x x + G_y y - Gz = 0$$

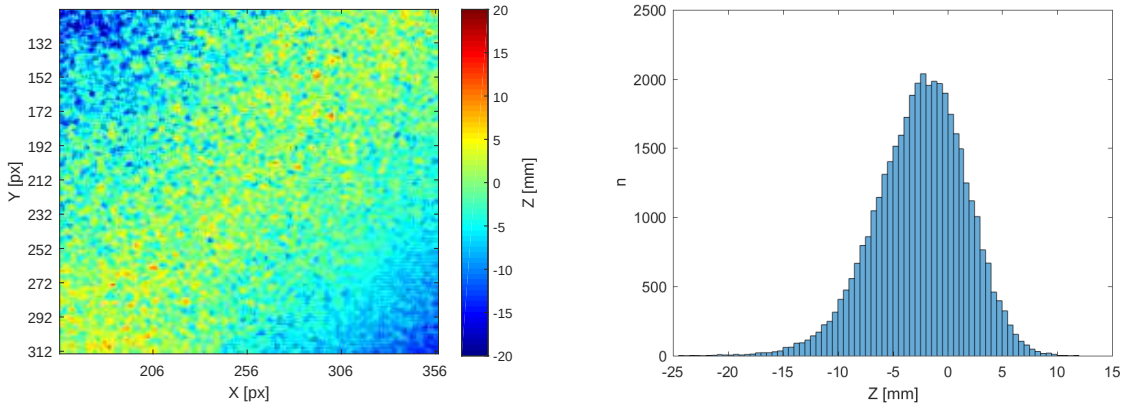
Diese Gleichung ist nur gültig, sofern $G \neq 0$ ist und wird verletzt, wenn $\mathbf{x} = 0$ oder $\mathbf{y} = 0$ ist und sofern $\mathbf{x} \neq 0$, $\mathbf{y} \neq 0$ mit der Bedingung $\mathbf{y} = \alpha \cdot \mathbf{x}$ und $\alpha \neq 0$ erfüllt ist. In diesen Fällen steht die zu approximierende Ebene orthogonal zu der x- oder y-Ebene. Aufgrund der technischen Gegebenheiten ist dieser Fall in dieser Arbeit nicht zu erwarten.

4.3 Gegenüberstellung und Validierung



(a) Farbliche Darstellung der Tiefenwerte. *(b) Histogramm der Tiefenwerte.*

Abbildung 7: Tiefenbildaufnahme vor der Korrektur.



(a) Farbliche Darstellung der Tiefenwerte. *(b) Histogramm der Tiefenwerte.*

Abbildung 8: Tiefenbildaufnahme nach der Korrektur durch den Ansatz der geringsten Standardabweichung.

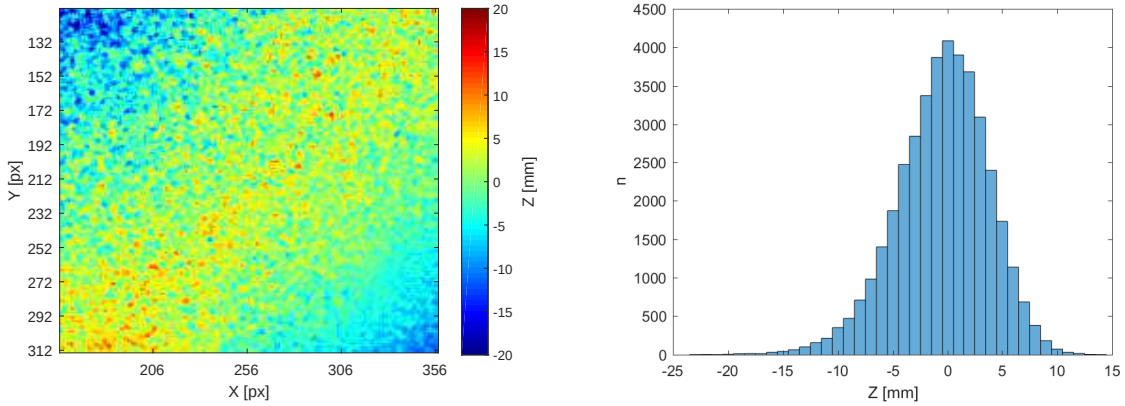
Für eine Gegenüberstellung[2] der beiden Ansätze wird die Kamera sehr stark geneigt und auf einen planaren Fußboden gerichtet. Anschließend wird eine Aufnahme erstellt und an beide Algorithmen übergeben. Die Laufzeit wird nicht beachtet, da die Approximation der Ebene nur einmalig während des Startvorganges ausgeführt wird. Die Erfassung der Tiefendaten für die Korrektur erfolgt in der Bildmitte einer Aufnahme durch ein Quadrat mit den Maßen 100x100 Pixel. Abbildung 8 zeigt das Resultat nach dem Ansatz der geringsten Standardabweichung. Ohne Korrektur (Abbildung 7) ist eine deutliche Steigung von mehreren hundert Millimetern in x- und y-Richtung im Bildmittelpunkt zu erkennen. Diese wird durch die Korrektur über den Ansatz mit der Standardabweichung auf zwanzig Millimeter (Abbildung 8) reduziert. Der Randbereich weist weiterhin Abweichungen ($-20\,\text{mm}$ und $+10\,\text{mm}$) gegenüber der planaren Ebene auf. Diese Abweichungen sind mit der radialen Verzeichnung des Objektives zu erklären und treten im Randbereich stärker auf. Sie werden durch Abbildung 17 näher

[2] Beide Algorithmen sind auf dem Datenträger in der Klasse „configurator“ zu finden.

erläutert und können durch eine lineare Ebene nicht korrigiert werden. Die inhomogenen Tiefenwerte im mittleren Bereich von −5 mm und +5 mm sind auf den „wigglin Effekt“ (Kapitel 6.1.1) zurückzuführen. Die starken Distanzunterschiede zwischen Abbildung 8 und Abbildung 7 entstehen durch die Subtraktion des mittleren Tiefenwertes $\bar{z}$ von jedem einzelnen Tiefenwert z_i, nachdem die Korrektur erfolgreich durchgeführt wurde:

$$z_{i_{neu}} = z_i - \bar{z} \tag{4}$$

Dadurch erfolgt die Verschiebung des z-Schwerpunktes der Punktwolke direkt in den Ursprung des Weltkoordinatensystems.



(a) Farbliche Darstellung der Tiefenwerte. *(b) Histogramm der Tiefenwerte.*

Abbildung 9: Tiefenbildaufnahme nach der Korrektur durch den Ansatz der kleinsten Fehlerquadrate.

Die Korrektur nach dem Ansatz über die approximierte Ebene ist in der Abbildung 9 zu sehen. Auch ihr ist eine deutliche Schräglagenkorrektur zu entnehmen, deren Tiefenwerte in einem Bereich von −20 mm und +14 mm liegen. Die Standardabweichung, in Bezug auf den Bildausschnitt von 200x200 Pixel, beträgt für das Ausgangsbild $\sigma_a = 65.89$, für den Ansatz nach der Standardabweichung $\sigma_{std} = 4,14$ und nach dem Verfahren der Ebenen-Approximation $\sigma_{app} = 4,17$. Der Mittelwert der Tiefenwerte beträgt, nach der Korrektur und vor Anwendung der Gleichung 4, $\overline{z}_{std} = 1298$ und $\overline{z}_{app} = 1300$ bei einem im Versuchsaufbau gemessenem Abstand von ca. $z_{real} = 1225$ mm. Die Histogramme zeigen, dass der Ansatz über die Standardabweichung ca. 1750 px auf den Wert Null korrigieren konnte, während es bei dem Ansatz über die kleinsten Fehlerquadrate 4000 px sind. Zwar ist die Standardabweichung in beiden Ansätzen nahe zu identisch, jedoch weist das Histogramm in Abbildung 8b im relevanten Bereich von Null eine Verschiebung von ca. $\overline{z}_{app} - \overline{z}_{std} = 2$ mm auf.

Zusammenfassung

Aus der Verschiebung um 2 mm und aus den weiter gestreuten Werten des Histogramms (Abbildung 8b gegenüber Abbildung 9b) lässt sich schlussfolgern, dass die durch Minimierung der Standardabweichung erfasste Ebene einen leichten Versatz aufweist. Der Ansatz über die Ebenen-Approximation ist, in Betracht der anzuwendenden Bildverarbeitungsalgorithmen, zielführender um die Ebene zu korrigieren. Gleichzeitig erfordert die Annäherung an die geringste Standardabweichung mehr Zeit (durch mehrere Iterationen). Zu beachten ist, dass die Korrektur der Schräglage die Koordinatentransformation nicht ersetzt, womit sich die hohe Differenz zwischen dem Mittelwert der Tiefendaten ($\overline{z}_{app}, \overline{z}_{std}$) und dem Abstand zwischen Sensor und Boden z_{real} erklären lässt. Sie wird durch die Rotation als Bestandteil der Koordinatentransformation korrigiert.

5 Koordinatentransformation

5.1 Rotation und Translation

Jeder Punkt in einem Raum lässt sich, ausgehend von dem Koordinatenursprung, durch seine Position in x-, y- und z-Richtung beschreiben. Gleiches gilt somit für einen Punkt, der Bestandteil eines zu fotografierenden Objektes ist. Dieser Punkt in der realen Welt wird von dem sogenannten Weltkoordinatensystem ausgehend durch den Vektor

$$\overrightarrow{w} = \begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix}$$

beschrieben. Weiterhin besitzt die Kamera ein eigenes Koordinatensystem, dem sogenannten Kamerakoordinatensystem, dessen Position in dem Weltkoordinatensystem zu finden ist und durch den Vektor

$$\overrightarrow{k} = \begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix}$$

definiert ist. Um den Zusammenhang eines Punktes zwischen beiden Koordinatensystem zu beschreiben, werden zwei mathematische Verfahren, die Rotation und Translation, auf eines der Koordinatensysteme angewendet.
Eine Rotation in einem dreidimensionalen Raum besitzt je nach Rotationsrichtung einen konstanten Wert. Wird um die z-Achse rotiert, bleiben die z-Werte unverändert, während sich x- und y-Werte verändern. Somit folgt für die Rotation um die z-Achse [Ott11, Gleichung 2.52]:

$$\vec{w_{rot,z}} = \mathbf{R_z} \cdot \vec{v} = \begin{pmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix}$$

Für die Rotation um die y-Achse folgt [Ott11, Gleichung 2.52]:

$$\vec{w_{rot,y}} = \mathbf{R_y} \cdot \vec{v} = \begin{pmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{pmatrix} \cdot \begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix} \tag{5}$$

Und für die Rotation um die x-Achse folgt [Ott11, Gleichung 2.52]:

$$\vec{w_{rot,x}} = \mathbf{R_x} \cdot \vec{v} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{pmatrix} \cdot \begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix} \tag{6}$$

Soll ein Punkt um alle Achsen rotiert werden, erfolgt eine Multiplikation der Rotationsmatrizen für jede Richtung. Diese kann zusammengefasst werden zu:

$$\mathbf{R = R_x \cdot R_y \cdot R_z}$$

Nach der Rotation erfolgt zur Überführung von Koordinatensystemen eine translatorische Ausrichtung. Hierbei wird der Ursprung des einen Koordinatensystems in den Ursprung des Zweiten gelegt, weshalb eine Subtraktion mit dem Translationsvektor $\vec{t}$ erfolgt:

$$\vec{w_t} = \vec{k} - \vec{t}$$

Der Vektor $\vec{t}$ enthält dabei die Differenz zwischen beiden Koordinatensystemen für alle drei Richtungen. Somit lässt sich eine translatorische- und rotatorische Transformation zu folgender Gleichung zusammenfassen:

$$\vec{w_T} = \boldsymbol{R}(\vec{k} - \vec{t})$$

Mit dieser Gleichung ist es möglich den Standpunkt eines Objektes im Weltkoordinatensystem in das Koordinatensystem der Kamera zu transformieren.

5.2 Homogene Koordinaten im zweidimensionalem Raum

Die Darstellung von Projektionen werden in der Computergrafik häufig durch homogene Koordinaten beschrieben. Sie bieten den Vorteil Rotation, Translation und perspektivische Verzerrung eines Punktes einheitlich durch die Multiplikation mit einer einzigen Matrix zu realisieren [Jäh12, Seite 84]. Diese Methoden sind für die spätere Koordinatentransformation von Vorteil.

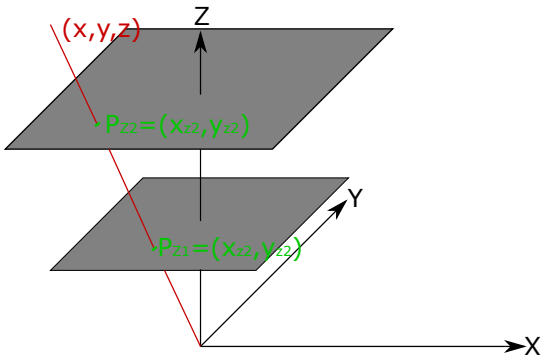


Abbildung 10: Schematische Darstellung der zwei Punkte P_{Z1} und P_{Z2} durch die homogenen Koordinaten (rot), in Abhängigkeit zu der jeweiligen Ebene in Z (In Anlehnung an [Kla10, Abbildung 2.17]).

Die Darstellung homogener Koordinaten basiert auf der Darstellung eines Punktes mit einer zusätzlichen Dimension. Während jeder Punkt in einer zweidimensionalen Ebene durch seine x- und y-Koordinaten definiert ist, beschreiben die homogenen Koordinaten den Punkt in einer bestimmten Ebene z. Bei der Betrachtung der Abbildung 10 sind die beiden Punkte P_{Z1} und P_{Z2} zu erkennen. Beide Punkte können, in Abhängigkeit zu der dritten Dimension z, durch die homogenen Parameter (rote Gerade) beschrieben werden. Mathematisch wird ein Punkt im kartesischen Koordinatensystem, ausgehend von den homogenen Koordinaten $P(x, y, z)$ und der Ebene in z, durch $P(\frac{x}{z}, \frac{y}{z})$ definiert, vorausgesetzt $z \neq 0$ ist erfüllt. Soll der Ursprung der homogenen Koordinaten beschrieben werden, erfolgt dies durch $z = 1$ und den entsprechenden Koordinaten in x- und y-Richtung. Somit lässt sich zusammenfassen, dass für die Abbildung von homogenen auf kartesischen Koordinaten im zweidimensionalen Raum gilt:

$$(x, y, z) \rightarrow (\frac{x}{z}, \frac{y}{z}) \text{ mit } z \neq 0$$

Für die Abbildung von kartesischen auf homogene Koordinaten gilt:

$$(x, y) \rightarrow (x, y, z) \text{ mit } z = 1$$

Die homogene Darstellung findet Anwendung für die Beschreibung der intrinsischen Kameraparameter, die durch das Lochkameramodell erläutert werden können.

5.3 Lochkameramodell

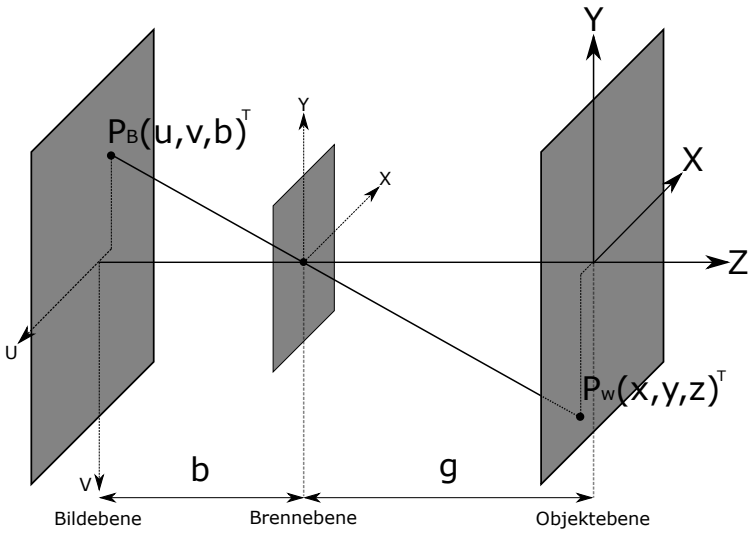


Abbildung 11: Modell einer Lochkamera, mit der Gegenstandsweite g, sowie der Bildweite b (In Anlehnung an [Jäh12, Abbildung 3.3]) .

Die prinzipiellen Verhältnisse der Abbildungsgeometrie einer Kamera lassen sich auf Grundlage der Lochkamera erläutern. Das Modell beschreibt dabei ein infinitesimal kleines Loch durch das, ausgehendend aus der Objektebene, ein Lichtstrahl einfällt und anschließend auf die Bildebene auftrifft. Der Strahl wird durch ein Objekt in dem Punkt $P_w = (x, y, z)^T$ reflektiert und trifft in der Brennebene die optische Achse, die durch die z-Achse repräsentiert wird. Der in der Bildebene erzeugte Punkt $P_B = (u, v, b)^T$ ist dabei abhängig von der Gegenstandsweite g sowie der Bildweite b. Die Zusammenhänge zwischen Bild- und Objektebene werden mathematisch durch den Strahlensatz beschrieben und lauten (in Anlehnung an [DK16, Seite: 30]):

$$P_B = \begin{pmatrix} u \\ v \end{pmatrix} = -\frac{b}{g} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \tag{7}$$

Eine Kamera besitzt in der Realität kein infinitesimal kleines Loch, sondern eine Linse mit der die Position der Brenn- bzw. Bildebene verschoben werden kann. Das Verhältnis von Gegenstands- zu Bildweite $\frac{b}{g}$ (sogenannte Größenverhältnis) wird dadurch beeinflusst und erlaubt ein anderes Abbildungsverhältnis. Dieser Zusammenhang wird durch Abbildung 12 verdeutlicht.

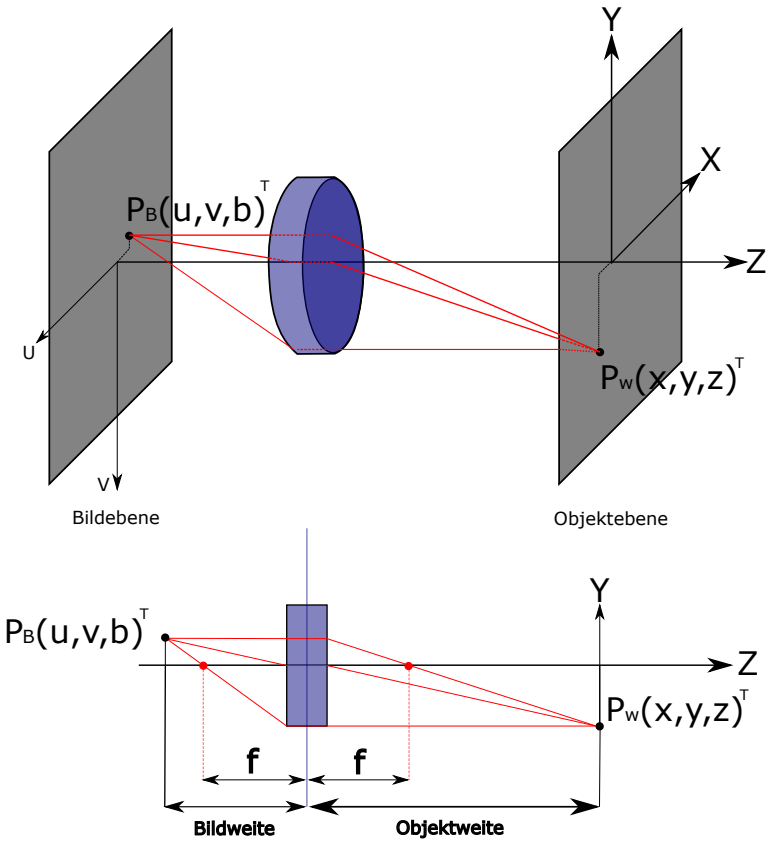


Abbildung 12: Modell einer Abbildung durch eine Linse (In Anlehnung an [Jäh12, Abbildung 3.3]).

Zu sehen ist, dass Ein- und Ausfallswinkel des Konstruktionsstrahls unterschiedlich sind. Sie sind von der Wölbung der Linse und den Brechungsindices der Umgebung und der Linse abhängig [DK16, Seite: 73]. Gegenüber dem Lochkameramodell besitzt die Linse den Vorteil deutlich mehr Lichtstrahlen einzufangen, woraus eine lichtstärkere Abbildung und eine geringere Belichtungszeit resultiert. Um nun ein scharfes Bild aus der realen Welt auf einen Sensor zu projizieren, wird der Sensor in die Bildebene der Linse platziert. Eine Platzierung vor und hinter die Bildebene hätte eine Unschärfe zur Ursache, die einen gewissen Toleranzbereich aufweist. Dieser wird durch die Größe der Pixel bestimmt, da sie den Grenzbereich eines abzubildenden Punktes definieren. Die geometrischen Maße zwischen Sensor und optischem Punkt werden durch die sogenannten intrinsischen Kameraparameter beschrieben.

5.4 Intrinsische Kameraparameter

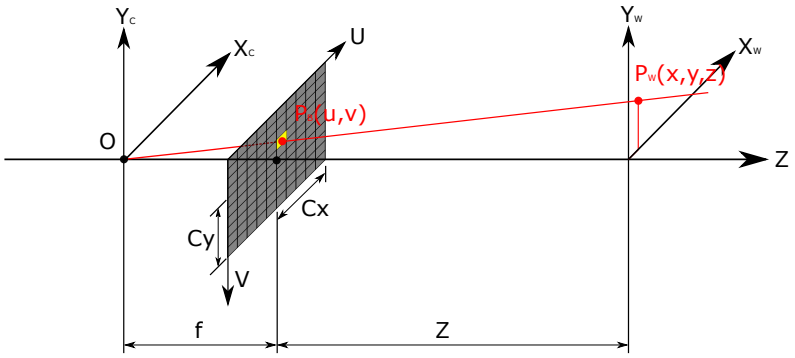


Abbildung 13: Schematische Darstellung der intrinsischen Kameraparameter.

Die innere Geometrie der Kamera wird durch intrinsischen Kameraparameter beschrieben, welche sich je nach Kamera unterscheiden, jedoch unabhängig von dessen Position in einem übergeordnetem Koordinatensystem sind. Der Abbildung 13 kann das Koordinatensystem der Kamera O in dem übergeordnetem Koordinantensystem der Welt entnommen werden. Zu sehen ist der Weltpunkt P_W, der auf den Kamerasensor durch den Punkt P_B abgebildet wird. Die Bildweite f definiert dabei den Abstand zwischen dem Sensor und dem Punkt O, der sich auf Höhe des Sensormittelpunktes befindet. Zu beachten ist, dass dieser Abstand nicht der Brennweite eines Objektives entspricht. Ausgenommen davon ist der Fall, dass das Objektiv auf eine unendliche Objektdistanz fokussiert, wodurch die parallel einfallenden Strahlen auf den Sensor treffen. Die Abstände C_x und C_y definieren den Abstand zwischen dem Ursprung des Sensors und dem Kameraursprung O. Gemeinsam bilden diese Parameter die intrinsischen Werte der Kamera. Diese sind notwendig um die Positionsinformation eines Pixels in Abhängigkeit zu dem Abstand z des Weltpunktes P_W in das übergeordnete Koordinatensystem abzubilden. Nach dem Strahlensatz aus Gleichung 7 ergibt sich in Bezug auf die Abbildung 13 folgender Zusammenhang:

$$P_B = \begin{pmatrix} u \\ v \end{pmatrix} = \frac{f}{z} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

Da sich Bild- und Kameraursprung unterscheiden, erfolgt eine Translation zwischen Bildmittelpunkt und Bildursprung:

$$P_B = \begin{pmatrix} u \\ v \end{pmatrix} = \frac{f}{z} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} C_x \\ C_y \end{pmatrix}$$

Danach gilt folgende Abbildungsgleichung [Sch13, Seite: 6]:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} \frac{f}{z} \cdot x + C_x \\ \frac{f}{z} \cdot y + C_y \end{pmatrix}$$

Nach Kapitel 5.2 kann die Abbildungsgleichung durch Hinzufügen einer weiteren Dimension in homogenen Koordinaten ausgedrückt werden:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} \frac{f}{z} \cdot x + C_x \\ \frac{f}{z} \cdot y + C_y \\ 1 \end{pmatrix}$$

Multipliziert mit z folgt aufgrund der Invarianz gegenüber einer Skalarmultiplikation für die Abbildung keine Änderung in der Ausgangsdarstellung [Sch13, Seite: 6]:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} f \cdot x + C_x \cdot z \\ f \cdot y + C_y \cdot z \\ z \end{pmatrix}$$

In der Matrixschreibweise kann diese mit der Einheitsmatrix $\mathbf{I}$ und der Kameramatrix $\mathbf{A}$ zusammengefasst werden zu [Sch13, Gl: 2.2]:

$$P_B = \begin{pmatrix} u \\ v \\ z \end{pmatrix} = \begin{pmatrix} f & 0 & C_x & 0 \\ 0 & f & C_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \mathbf{A} \cdot [\mathbf{I}|\mathbf{0}] \cdot P_W$$

Die Kameramatrix enthält dabei die intrinsischen Werte f, C_x und C_y, wobei die Bildweite f aufgrund des Herstellungsprozesses der Kamera in x- und y-Richtung unterschiedliche Werte aufweist und entsprechend differenziert werden muss:

$$\mathbf{A} = \begin{pmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{pmatrix} \tag{8}$$

Für eine vollständige Transformation zwischen einem Weltkoordinatensystem und dem Kamerakoordinatensystem werden die extrinsischen Parameter benötigt, die von der Struktur des Arbeitsumfeldes abhängig sind.

5.5 Extrinsische Kameraparameter

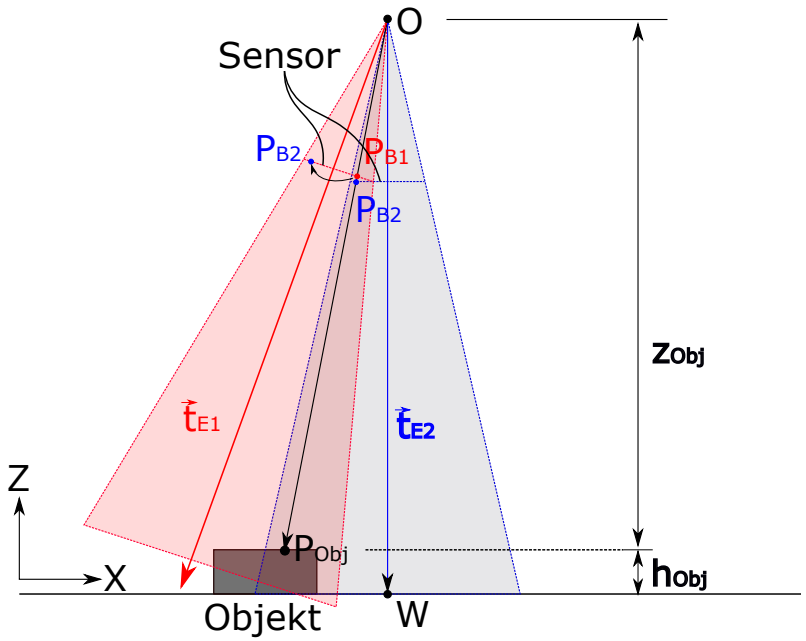


Abbildung 14: Schematische Darstellung der extrinsischen Parameter.

Die extrinsischen Kameraparameter beschreiben die Position des Kameraursprungs O in dem übergeordnetem Koordinatensystem W. Für die Bestimmung der extrinsischen Parameter ist das Arbeitsfeld in Abbildung 14 zu betrachten. Der Abbildung ist zu entnehmen, dass die Kamera nicht exakt senkrecht zu der Tischebene positioniert werden kann. Es entsteht somit eine Schräglage in x- und y-Richtung, die durch Kapitel 4 bestimmt wird. Für eine korrekte Abbildung zwischen Welt- und Kamerakoordinaten muss diese Schräglage beachtet werden, da aus ihr die Rotationswinkel α und β bestimmt werden. Der in Kapitel 4 vorgestellte Algorithmus liefert die Steigung a und b in $\frac{\text{mm}}{\text{px}}$ für die x- und y-Richtung als Ebenengleichung in der Koordinatenform. Aus der Koordinatenform kann die Steigung somit durch die Parameter a und b (vgl. Formel 2) verwendet werden und die Rotationswinkel können wie folgt bestimmt werden:

$$\alpha = \tan^{-1}\left(\frac{a}{1\,\text{px}}\right),\ \beta = \tan^{-1}\left(\frac{b}{1\,\text{px}}\right)$$

Der Punkt P_{Obj} in Abbildung 14 wird auf dem Sensor in dem Punkt P_{B1} dargestellt. Wäre die Kamera exakt senkrecht zu der Ebene platziert, würde der Objektpunkt durch P_{B2} abgebildet werden und somit dem Referenzpunkt entsprichen. Zur Bestimmung des Punktes P_{B1} erfolgt die Rotation dieses Punktes. Für die Rotation wird im ersten Schritt der Punkt P_{B1} um die Winkel α und β rotiert, indem diese durch die Rotationsmatrix aus Gl.5 und 6 mit dem Punkt P_{K1} multipliziert werden:

$$P_{B2} = \mathbf{R_X} \cdot \mathbf{R_Y} \cdot P_{B1} \tag{9}$$

Im Anschluss erfolgt die translatorische Bewegung auf der z-Achse. Der in Abbildung 14 dargestellte translatorische Vektor $\vec{t}_{E1}$ weist keine translatorische Bewegung in x- oder y-Richtung auf, da der Weltursprung auf der z-Achse des Kamerakoordinatensystems liegt. Dieser lautet:

$$\vec{t}_{E1} = \begin{pmatrix} 0 \\ 0 \\ Z_{t_{E1}} \end{pmatrix}$$

Der Wert z_{E1} kann der Schräglagenkorrektur nach Kapitel 4 entnommen werden, da er dem Mittelwert der Tiefendaten entspricht. Da dieser Wert auf der schrägen Kameraposition basiert ist er gegenüber dem senkrecht stehenden Vektor $\vec{t}_{E2}$ im Betrag größer. Es erfolgt daher eine Korrektur durch die Rotation um die Winkel α und β:

$$\vec{t}_{E2} = \mathbf{R_X} \cdot \mathbf{R_Y} \cdot \vec{t}_{E1}$$

Anschließend kann durch die Höhe eines Objektes der z-Wert für den Objektpunkt bestimmt werden:

$$Z_{Obj} = Z_{t_{E2}} - h_{Obj} \tag{10}$$

5.6 Transformation zwischen Pixel- und Weltkoordinaten

Nachfolgend bezieht sich ein Punkt P^O auf das Kamera- und ein Punkt P^W auf das Weltkoordinatensystem. Für die Transformation zwischen Pixel- und Weltkoordinaten werden im ersten Schritt die Pixelkoordinaten durch Multiplizieren mit der inversen Kameramatrix $\mathbf{A}$ aus Gl.8 in Kamerakoordinaten umgerechnet. Für einen homogenen Punkt im Bild: $P_B^O = (u, v, w)^T$ errechnet sich somit der abzubildende Punkt im Kamerakoordinatensystem $P_K^O = (x_k, y_k, z_k)^T$ nach:

$$P_K = \mathbf{A^{-1}} \cdot P_B$$

Dieser wird nun nach Gl. 9 erweitert, um eine Korrektur der Schräglage durchzuführen:

$$P_K^O = \mathbf{R_X} \cdot \mathbf{R_Y} \cdot \mathbf{A^{-1}} \cdot P_B^O \tag{11}$$

Um aus diesem Punkt den Objektpunkt $P_{Obj}^W = (x_{Obj}, y_{Obj}, z_{Obj})^T$ zu erhalten, werden die homogenen Koordinaten auf die Ebene bezogen, in der sich die Oberfläche des Objektes befindet. Die Gerade der homogenen Koordinaten in der Parameterform lautet im Allgemeinen:

$$\vec{r} = \lambda \cdot \vec{u}$$

Übertragen auf den Strahlensatz entspricht dabei $\vec{r}$ dem Objektpunkt P_{Obj}^{W} und $\vec{u}$ dem korrigierten Kamerapunkt P_{K}^{O}:

$$P_{Obj}^{W} = \lambda \cdot P_{k}^{O} \tag{12}$$

In dem Gleichungssystem sind die Parameter z_k (vgl. Gl. 11) und z_{Obj} (vgl. Gl. 10) bekannt, weshalb λ wie folgt bestimmt werden kann:

$$\lambda = \frac{z_{Obj}}{z_k}$$

Eingesetzt in Gl. 12 kann somit der Objektpunkt P_{Obj}^{W} bestimmt werden. Die Einwirkung der Verzeichnung wurde dabei nicht korrigiert.

5.7 Verzeichnung

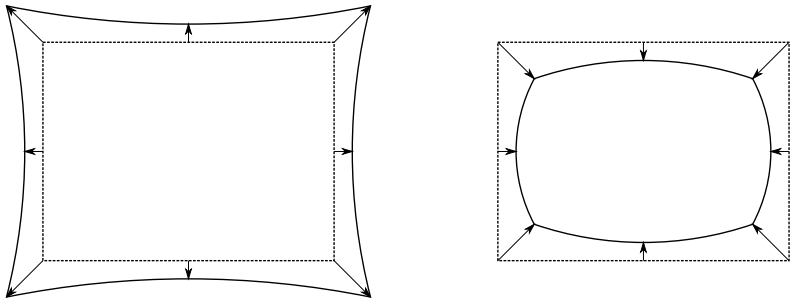


Abbildung 15: Tonnenförmige- (rechts) und kissenförmige (links) Verzeichnung eines Rechtecks (gestrichelt). In Anlehnung an [Jäh12, Abbildung 3.12].

Eine Bildaufnahme unterliegt in den meisten Fällen einer Verzeichnung, da die verwendeten Objektive von der idealen Zentralprojektion abweichen [Jäh12, Seite: 96]. Sie hat zur Ursache, dass ein Rechteck auf der Bildebene kissen- oder tonnenförmig dargestellt wird, wie im Schema der Abbildung 15 dargestellt. Ein Kreis, dessen Mittelpunkt auf der optischen Achse liegt wird zwar kreisförmig auf der Bildebene projiziert, weist jedoch einen veränderten Radius auf. Es handelt sich somit um eine radiale Verzerrung, die wie folgt mathematisch beschrieben werden kann [Jäh12, Gleichung 3.36]:

$$\mathbf{x}' = \frac{\mathbf{x}}{1 + k_3 \left|\mathbf{x}\right|^2}$$

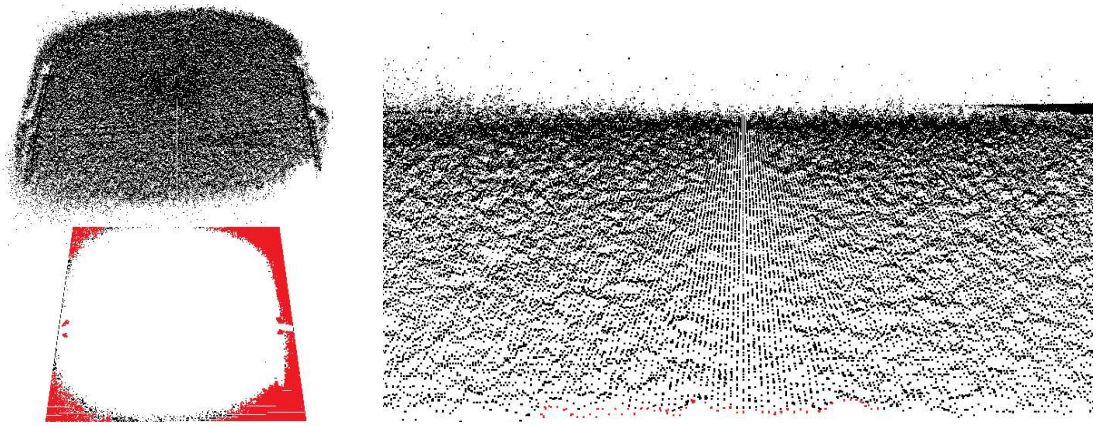
Ist k_3 positiv, handelt es sich um eine tonnenförmige Verzeichnung, andernfalls um eine kissenförmige. Ist $k_3 = 0$, so ist keine Verzeichnung vorhanden. Für eine korrekte Erfassung der Transportboxen muss eine Korrektur der Verzeichnung erfolgen. Die Kinect verfügt über zwei Sensoren und zwei unterschiedliche Linsen, weshalb die Verzeichnung unabhängig voneinander auf beide Bildquellen einwirkt und korrigiert werden muss. Für das Tiefenbild existiert die Korrektur in Form des Coordinate-Mappers. Die Korrektur des Farbbildes erfolgt durch Implementieren bestehender Funktionen aus dem OpenCv-Framework.

6 Datenerfassung und Aufbereitung

6.1 Untersuchung des Sensors

Um den Sensor als Informationsquelle zu nutzen, muss dieser im Vorfeld auf seine Eignung geprüft werden. Hierzu erfolgt im ersten Schritt die Untersuchung des Tiefenbildes und im Anschluss die des Farbsensors.

6.1.1 Tiefenbild

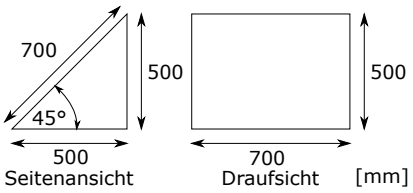


(a) Bildschirmaufnahme ohne Korrektur und Schwellwertbildung. Rot makiert ist das durch den Sensor auf 0 gesetzte Rauschen.

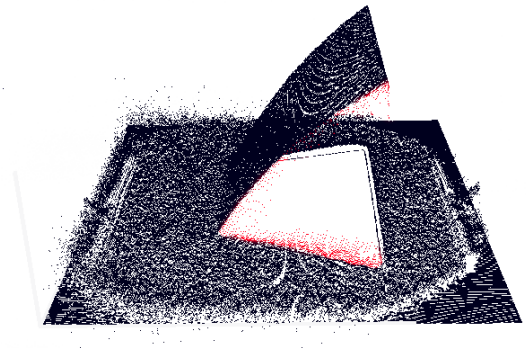
(b) Nahaufnahme der korrigierten Ebene mit dem rot makierten „Wiggling Error“.

Abbildung 16: Aufnahmen im Tiefenbild.

Wird das Bild des Sensors in der Abbildung 16 betrachtet, sind zwei Eigenschaften zu erkennen. Die Pixel weisen im Randbereich ein deutlich höheres Rauschen auf, was auf die die Eigenschaft zurückzuführen ist, dass das Infrarotlicht kegelförmig ausgestrahlt wird und nicht alle Bereiche homogen ausgeleuchtet werden [CGMS15, Seite: 587]. Zusätzlich werden die Pixel aufgrund der Linsenverzerrung im Randbereich fehlerhaft abgebildet. In Abbildung 16b ist außerdem zu erkennen, dass eine wellenförmige Struktur abgebildet wird und keine exakt planare Ebene. Dieses Verhalten wird im Englischen als „Wiggling Error“ also als „Wackel-Fehler“ bezeichnet und wird durch die interne Elektronik verursacht. Dabei ist das meist sinusoidal emittierte Infrarotsignal nicht exakt sinusförmig, weswegen das auf den Sensor eintreffende Signal einen harmonischen Offset aufweist, der je nach Pixel und Distanz variiert [CGMS15, Seite: 587]. Weitere Untersuchungen der Politecnico di Milano in Italien haben gezeigt, dass die Tiefendaten ebenfalls von dem Einfallswinkel des reflektierten Lichts abhängig sind [CGMS15]. Eine starke Neigung eines Objektes im Bild verursacht eine Reflektion und Minderung des IR-Lichts (Infrarot) , wodurch eine Minderung des Signal-Rausch-Verhältnisses entsteht [CGMS15, Seite: 588].



(a) Technisches Konstruktionsschema des Testobjektes, mit Millimeter angaben.

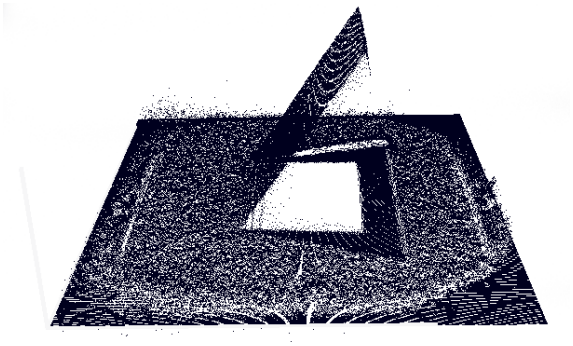


(b) Bildschirmaufnahme des Testobjektes in der grafische Visualisierung durch die Ogre-Enginge. Rot matkiert sind die sog. „Mixed Pixel“. Zu beachten ist der Moire Effekt, der durch die Visualisierung und nicht durch die Kinect entsteht.

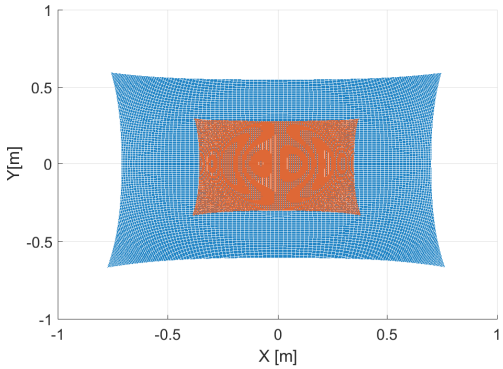
Abbildung 17: Gegenüberstellung der technischen Zeichnung des Testobjektes und der Aufnahme im Tiefenbild.

Für die weitere Analyse der Tiefenwerte steht ein Testobjekt zur Verfügung, das den Maßen der Abbildung 17a entspricht und im ersten Schritt mittig des Bildfeldes platziert wird. Zu beachten ist, dass zu große Tiefenwerte bereits auf Null gesetzt wurden, weswegen die Randbereiche der Aufnahme eben sind. Eine große Differenz zwischen dem realen Messobjekt und der Visualisierung ist in der Form zu finden. Während das Testobjekt eine geradlinige 45°-Steigung aufweist, ist in der visualisierten Darstellung eine deutliche Krümmung zu erkennen. Die in Abbildung 17b gezeigte Aufnahme weist an den Kanten des Objektes abweichende Tiefenbildinformationen (rot markierte Pixel) auf. Diese sogenannten „Mixed Pixel“ verfälschen an den Kanten die Tiefenwerte und sind auf die Mittelwertbildung benachbarter Pixel zurückzuführen. Die Mittelwertbildung dient der Rauschreduzierung des Tiefenbildes [CGMS15, Seite: 591]. Versuche haben gezeigt, dass sowohl die Rohdaten als auch die durch den Coordinate-Mapper korrigierten Werte diese Pixel aufweisen, weshalb eine Elimination nur durch zusätzliche Rechenmethoden erfolgen kann.

6.1.2 Coordinate-Mapper



(a) Bildschirmaufnahme des Testobjektes in der grafischen Visualisierung durch die OGRE-Engine und durch den Coordinate-Mapper korrigiert. Zu beachten ist der Moire-Effekt, der durch die Visualisierung und nicht durch die Kinect entsteht.



(b) Darstellung der Lookup- Tabelle des Coordinate- Mappers bei einer konstanten Distanz von $z = 1000\,\text{mm}$ (blau) und $z = 500\,\text{mm}$ (rot).

Abbildung 18: Transformation durch den im Kinect SDK v. 2.0 enthaltenen Coordinate- Mapper.

Das von Windows mitgelieferte Kinect-SDK (Software Development Kit) umfasst einen auf das Lochkameramodell basierenden „Coordinate-Mapper“, der die Zuordnung zwischen dem Tiefenbild und der realen Welt übernimmt. Mit diesem ist es möglich die Position eines Pixels in die entsprechenden Weltkoordinaten zu transformieren. Der Programmierer kann dabei wahlweise ein ganzes Frame, einen Punkt, ein Array mit Punkten oder manuell durch eine Lookup-Tabelle transformieren. Eine Transformation mit der Lookup-Tabelle ist in Abbildung 18b zu sehen. Sie zeigt zum Einen ein mit der Distanz zunehmendes Sichtfeld sowie eine Korrektur der Tiefenwerte in x- und y-Richtung. Diese Korrektur weist gegenüber der Mitte in den Randbereichen ein stärkeres Maß auf, ist kissenförmig und steigt mit zunehmender Distanz z. Die kissenförmige Struktur lässt schlussfolgern, dass das Rohbild einer tonnenförmigen Verzeichnung unterliegt, die durch das Objektiv verursacht wird. Diese Annahme wird dadurch gestützt, dass eine Transformation mit der Lookup-Tabelle keinen Einfluss auf den Tiefenwert nimmt, sondern nur auf die x- und y-Position. Die Transformation einer Aufnahme mit dem Testobjekt aus Abbildung 17a ist in Abbildung 18a zu sehen, welcher eine korrigierte Form des Testobjektes zu entnehmen ist, die jedoch einen Schatten wirft. Dieser Schatten entsteht durch den kegelförmig von der Kamera ausgehenden Lichteinfall, der nicht alle Bereiche hinter einem Objekt ausleuchtet und somit keine Tiefenwerte erzeugen kann. Der Coordinate-Mapper liefert die metrischen Daten ausschließlich in Form eines Arrays mit metrischen Punkten für die x-, y- und z-Position, weshalb für die Anwendung von Bildverarbeitungsalgorithmen die Abbildung auf eine Matrix erfolgt. Hierzu werden die metrischen Punkte auf eine Matrix der Größe 512x424 px abgebildet. Dabei treten zwei Fehlerquellen auf:

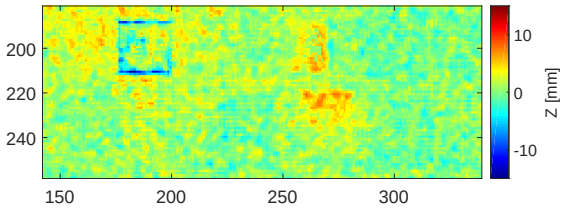
- Moiré-Effekt: Eine Überlagerung von zwei unterschiedlich skalierter Muster hat den Moiré-Effekt zur Folge. Dabei werden einige Pixel der Matrix nicht von einem metrischen Punkt getroffen und es entsteht ein sichtbares, grob erscheinendes Muster.
- Überlagerung: Aufgrund der unterschiedlichen Skalierung zwischen Matrix und metrischen Daten, treffen mehrere metrische Punkte in den Bereich eines Pixels. In diesem Fall findet eine Mittelwertbildung statt.

Eine Gegenüberstellung zwischen originaler Aufnahme und der gemappten sowie auf eine Matrix abgebildeten Aufnahme ist der Abbildung 21 zu entnehmen.

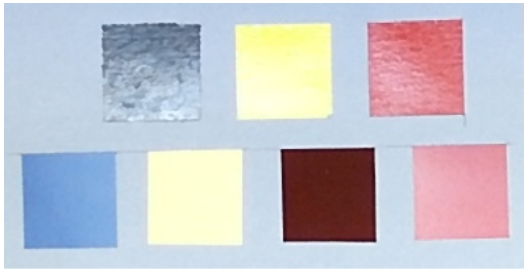
6.1.3 Farbbild

Abbildung 19: Aufnahme einer Box im Farbbild.

Die Aufnahme der Abbildung 19 zeigt die größere Transportbox im Farbbild. Zu erkennen ist, dass im Farbbild keine genauen Merkmale vorhanden sind, die auf die Position der Ecken rückschließen lassen. Zwar kann eine Differenzierung zwischen dem Boden und der Kante erfolgen, diese ist jedoch unklar definiert und abhängig von der Situation. Ein Schattenwurf nimmt z.B Einfluss auf den Unterschied zwischen Boden und Box. Die Differenzierung zwischen Box und Umgebung wird durch die graue Farbe zusätzlich erschwert, da sie im RGB-Raum (Rot Grün Blau) aus gleichen Anteilen rot, grün und blau besteht und somit eine unbunte Farbe repräsentiert. Für eine klare Definition der Kanten werden diese daher um eine farbliche Markierung in den Innenrändern erweitert. Entgegen der Aussage in [CGMS15, Seite 589] muss die Auswahl der Farbe unter Berücksichtigung des Tiefenbildes erfolgen, da diese Einfluss auf die Messung nimmt. Die Messungen in [WS16, Abbildung 6] zeigen, dass die Farbe schwarz eine für die IR-Aufnahme ungeeignete Farbe ist, da sie mit 20 mm - 25 mm die größte Tiefenabweichung gegenüber dem Referenzwert aufweist. Die am geeignetsten Farben sind hingegen orange, gelb, weiß und lila. Weiß entfällt aufgrund der farblichen Nähe zu grau. Durch eine zu helle Einstrahlung wird sehr schnell eine Übersättigung der grauen Farbe in den weißen Bereich erreicht und verhindert somit eine klare Trennung. Für die weiteren Zwecke werden die Farben schwarz matt (Tafelfarbe), orange, rot und blau in matten Farben und rot in Form von Vlies gegenübergestellt. Alle Farben werden durch Klebefolie auf einem grauen Karton platziert und eine Messung im Tiefen- und Farbbild durchgeführt. Ergänzend werden matte Acrylfarben für die Farben rot und gelb untersucht.



(a) Tiefenbildaufnahme, mit der Anordnung der Farben aus Abbildung 20 b.



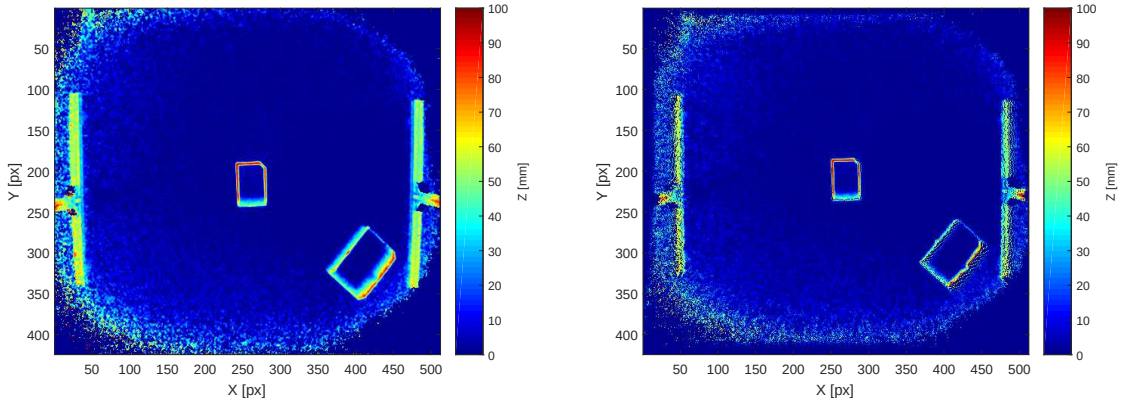
(b) Farbaufnahme, oben Tafellack, Acrylfarbe (orange und rot) und unten klebende Folie (blau, gelb, weinrotes Vlies, rot).

Abbildung 20: Gegenüberstellung verschiedener Farben im Farb- und IR-Bild.

Während die Tafelfarbe im Farbbild weniger reflektiert als die Acrylfarben, verursacht diese im Tiefenbild eine deutliche Verfälschung der Messung, die durch die blauen Tiefenwerte in Abbildung 20 a zu erkennen ist. Diese Eigenschaft ist darauf zurückzuführen, dass schwarze Strahlung absorbiert wird und dadurch die Laufzeitdifferenz der Distanzmessung beeinflusst. Die Farben orange, blau und rot in beiden Varianten, zeigen keine deutlichen Unterschiede im Tiefenbild, dafür deutliche

Reflektionen im Farbbild. Die gelbe Acrylfarbe und Klebefolie geht bereits in den Sättigungsbereich über und erscheint weiß. Einige Stellen in der Tafelfarbe reflektieren nach wie vor stark und sind übersättigt. Die Übersättigung ist auf den Sensor und die Lichtreflektion zurückzuführen, da der Sensor die Belichtungszeit dynamisch anpasst und diese durch den Anwender nicht beeinflussbar ist, sofern dieser die Beleuchtungssituation nicht verändert. Die Filz basierende Oberfläche zeichnet sich durch eine homogene Darstellung im Tiefenbild aus, da ihre raue Oberflächenstruktur eine diffuse Reflektion verursacht und im Farbbild nicht in den Sättigungsbereich übergeht.

6.2 Merkmalsextraktion zur Detektion der Transportboxen im Tiefenbild



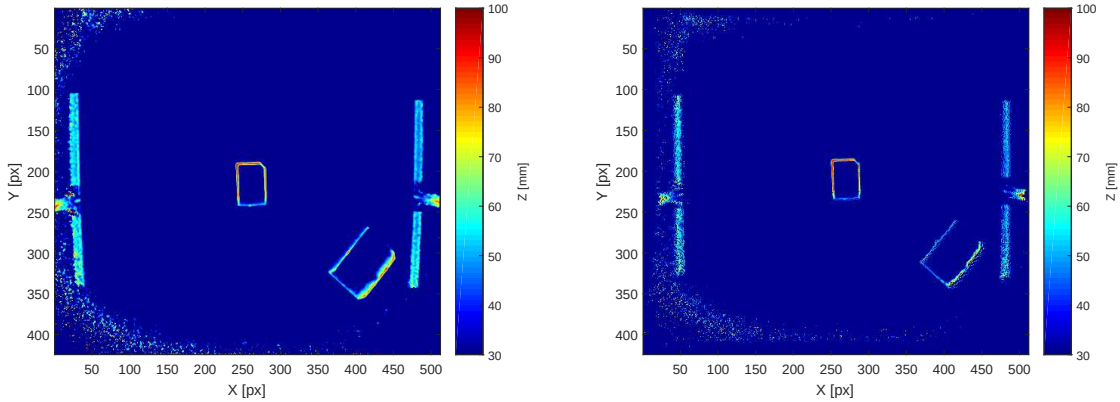
(a) Ohne Verwendung des Coordinate- Mappers.

(b) Unter Verwendung des Koordinate Mappers und durch Abbildung auf eine Matrix.

Abbildung 21: Schräglagen korrigierte Aufnahme des Tiefenbildes in dem sich beide Transportboxen, klein (mittig) und groß (rechts unten) befinden.

Auf Höhe der Objektoberkante (1200 mm) wird ein Abbildungsmaßstab von $3,5\frac{\text{mm}}{\text{px}}$ erfasst, der abhängig vom Lösungskonzept des Objekterkennungsalgorithmus, die maximal zu erwartende Abweichung repräsentiert. Er deckt sich mit dem Abbildungsmaßstab aus [CGMS15, Seite: 590], wonach ein Pixel bei einer Distanz von einem Meter zwischen Objekt und Kamera, $2,7\frac{\text{mm}}{\text{px}}$ entspricht. In diesem Abschnitt sollen unter Verwendung der Transportboxen, Merkmale im Tiefenbild untersucht werden. Für die Untersuchung wird die kleinere Transportbox (vgl. Tabelle 1) in die Bildmitte und die größere Box am Rand des Arbeitsbereiches platziert. Die Aufnahme des Tiefenbildes erfolgt nach Korrektur der Schräglage für das Originalbild und ein weiteres Mal unter Verwendung des Coordinate-Mappers. Anschließend werden die Eckpunkte der Box dazu verwendet die Beträge der Seiten zu bestimmen. Sie sollen einen ersten Aufschluss über die Genauigkeit geben bevor weitere Untersuchungen stattfinden. Beide Abbildungen zeigen, dass mit der mittigen Platzierung der Box eindeutige Merkmale zu erkennen sind. Durch die Ableitung der Tiefeninformation können die innere und äußere Objektoberkante deutlich von der Arbeitsfläche und voneinander unterschieden werden. Die Box im unteren rechten Teil weist hingegen unklare Merkmale auf. Im Originalbild ist eine starke Steigung zwischen der rechten Objektaußenkante sowie der linken Innenkante zu erkennen. Dieses Verhalten ist auf das schräg einfallende Infrarot-Licht und den intrinsischen Kameraparametern zurückzuführen. Aus der Kameraperspektive ist die linke Außenfläche und die rechte Innenfläche der Seiten deutlich zu sehen, während die linke Innenfläche sowie rechte Außenfläche durch den Schatten verdeckt werden. Eine Transformation durch den Coordinate-Mapper (*COOM*) hat eine Korrektur der Flächen zur Folge (vgl. Abbildung 21b), weshalb die linke Außenkante und die rechte Innenkante stärker definiert sind. Die Gegenüberstellung zeigt ebenfalls die Mittelwertbildung benachbarter Pixel neben den scharfen Kanten im Schattenbereich

der Boxen. Für eine einfache Merkmalsextraktion erfolgt im nächsten Schritt die Schwellwertbildung für die Tiefenwerte, deren Ergebnis der Abbildung 22 zu entnehmen ist.



(a) Thresholding ohne Verwendung des Coordinate-Mappers.

(b) Thresholding mit Verwendung des Coordinate-Mappers und durch Abbildung auf eine Matrix.

Abbildung 22: Aufnahme des Tiefenbildes unter Anwendung der Schwellwertgrenze von 35 mm.

Die Abbildungen 22a und 22b zeigen für eine Schwellwertgrenze von 35 mm, dass die erste Kante der unteren rechten Box vollständig verschwunden ist, obwohl diese eine Höhe von 100 mm besitzt. Wird die untere rechte Box um einige Grad um ihren Mittelpunkt gedreht, ist die Kante vollständig zu erkennen, weshalb das frühe Fehlen auf den Einfallswinkel des IR-Lichtes und dem Schmalheitsgrad der Kante zurückzuführen ist. Ein Schwellwert von 15 mm führt zur Entfernung des Rauschens über dem Boden ohne Eliminierung der Kanten. Daher wird dieser Schwellwert im weiteren Vorgehen angewendet. Durch diesen Fehler ist die Anwendung des „Openings“ oder „Erodierens“ ausgeschlossen, da die schmalen Kanten bereits bei einer einmaligen Anwendung eliminiert werden.

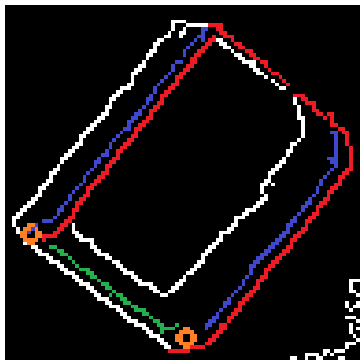
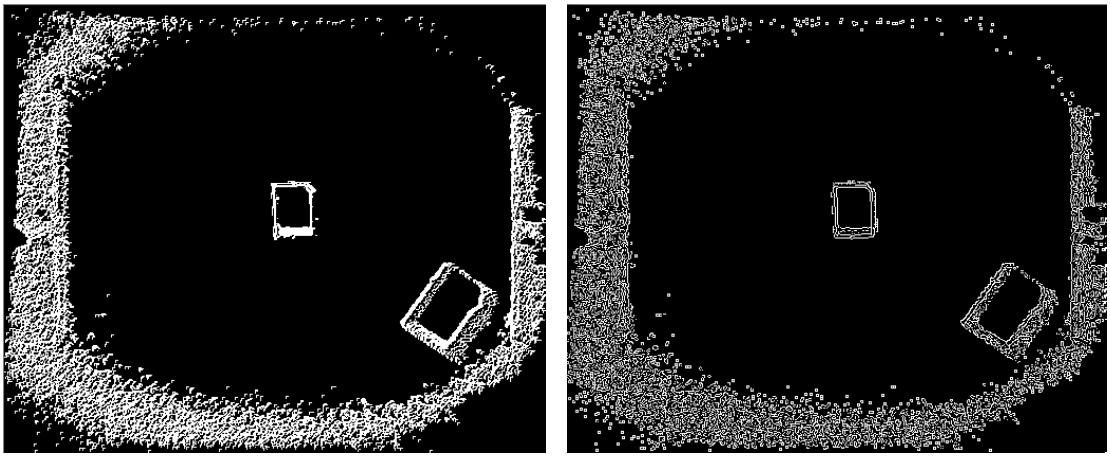


Abbildung 23: Vergrößerte und farblich unterteilte Transportbox, nach Anwendung des Canny-Algorithmus.

Wird die größere Box in Abbildung 22a vergrößert und das Bild durch den Canny-Algorithmus ([Can86]) gefiltert, ist zu erkennen, dass eine klare Definition der Eckpunkte der Box durch die Kameraperspektive deutlich erschwert ist. Durch die Schwellwertgrenze des Canny-Algorithmus sind zu kleine Kanten bereits herausgefiltert. Die stark ausgeprägten Kanten sind rot markiert. Kanten, die nur in wenigen Bildern eindeutige Konturen aufweisen, sind blau und grün marktiert. Objektkanten, die irrelevant sind oder dem Rauschen (unterer rechter Bildteil) zu zuordnen sind, sind weiß dargestellt. Die orangenen Punkte symbolisieren den erforderlichen Schnittpunkt der Kanten und somit die Eckpunkte

der Transportbox. Die Lokalisierung der Eckpunkt auf der Greifseite anhand der Kanten setzt somit eine Zuordnung der unterschiedlichen Kanten (grün, rot innen und blau innen) voraus. Da diese abhängig von der Position der Box bzw. der Kameraperspektive sind, variieren sie und sind nicht eindeutig identifizierbar. Zusätzlich können Innen- und Außenkanten nicht eindeutig getrennt werden, wodurch eine Abweichung in der Positionsbestimmung zu erwarten ist. Aus diesem Grund wird von der Verwendung des originalen Tiefenbildes abgesehen und das durch den Coordinate-Mapper korrigierte Bild betrachtet.



(a) Sobel-Operator. *(b) Canny-Operator.*

Abbildung 24: Coordinate-Mapper korrigierte Aufnahme des Tiefenbildes unter Anwendung der Schwellwertgrenze von 15 mm *und des Canny- sowie Sobel-Operators.*

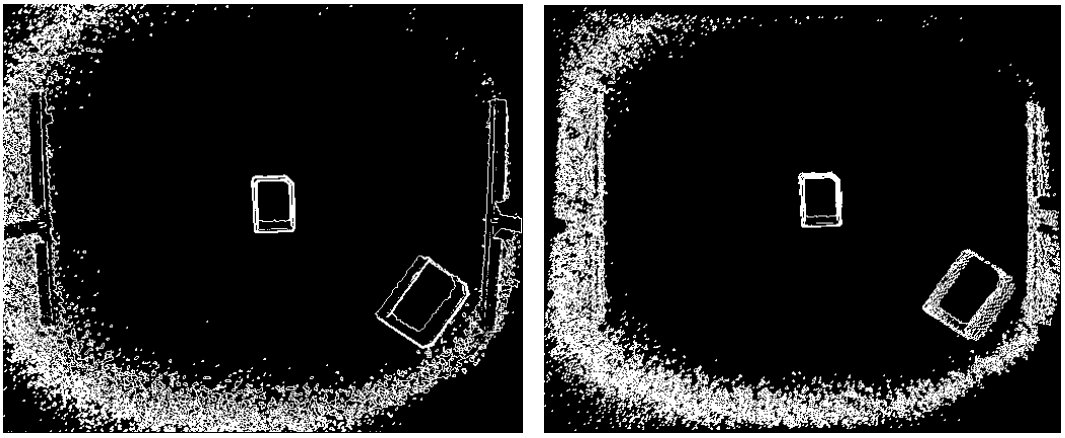
In Abbildung 24 ist die Mittelwertbildung der Pixel deutlich zu erkennen. Da eine Anwendung der Erosion bzw. des Openings ausgeschlossen ist, kann die naheliegendste Methode zur Eliminierung dieser Pixel nicht verwendet werden. Die Anwendung des Sobel- ([DK16, Seite: 176]) und des Canny-Algorithmus auf das korrigierte Bild ist der Abbildung 24 zu entnehmen. Die Gegenüberstellung zeigt, dass der Canny-Algorithmus gegenüber dem Sobel-Operator präziser ist. Eine Abgrenzung der Objektoberkanten ist unter Verwendung des Canny-Algorithmus nur für die kleinere Transportbox möglich, da die Mittelwertbildung der Pixel nahe der großen Box in die Objektoberkante miteinfließen. Das Ergebnis des Sobel-Operators zeigt hier einen Vorteil, da die Oberkante deutlich von der Mittelwertbildung abgegrenzt ist. Beide Verfahren erlauben keine eindeutige Aussage über die Lage aller vier Kanten. Weiterhin sind die Kanten in sich nicht vollständig geschlossen und lückenhaft. Da keiner der Algorithmen durch OpenCv eine minimal und maximale Steigung der Pixel filtern kann, wird an dieser Stelle ein eigener Algorithmus vorgestellt. Durch ihn sollen Kanten mit einer bestimmten Steigung in der Tiefe erkannt werden:

$$dstX(x,y) = \begin{cases} 255 & \text{if } \mathrm{abs}(\mathrm{src}(\mathrm{x}+1,\mathrm{y}) - \mathrm{src}(\mathrm{x},\mathrm{y})) > \min \,\&\, \mathrm{abs}(\mathrm{src}(\mathrm{x}+1,\mathrm{y}) - \mathrm{src}(\mathrm{x},\mathrm{y})) < \max \\ 0 & \text{andernfalls} \end{cases} \tag{13}$$

$$dstY(x,y) = \begin{cases} 255 & \text{if } \mathrm{abs}(\mathrm{src}(\mathrm{x},\mathrm{y}+1) - \mathrm{src}(\mathrm{x},\mathrm{y})) > \min \,\&\, \mathrm{abs}(\mathrm{src}(\mathrm{x}+1,\mathrm{y}) - \mathrm{src}(\mathrm{x},\mathrm{y})) < \max \\ 0 & \text{andernfalls} \end{cases} \tag{14}$$

$$dst(x,y) = dstX + dstY \tag{15}$$

Es erfolgt daher eine Differenzbildung für jedes Pixel zum benachbartem Pixel in x- sowie y-Richtung. Ist die Differenz im Bereich von min bis max, wird der entsprechende Pixel auf den Wert 255 gesetzt, andernfalls auf 0. Das Ergebnis für das Originalbild und das COOM-Bild ist der Abbildung 25 zu entnehmen.



(a) Originalbild. *(b) COOM gemapptes Bild.*

Abbildung 25: Anwendung des Algorithmus aus Gleichung 15 auf das COOM korrigierte und orignale Bild, mit den Algorithmenparametern $min = 10$ *und* $max = 100$.

In Abbildung 25 ist zu erkennen, dass der Algorithmus aus Gleichung 15 in dem Originalbild ausschließlich am Rand befindliche Kanten, deren Steigung zwischen der Arbeitsfläche und der Objektkante stärker ausgeprägt ist, detektiert. Durch den Moiré-Effekt und der Mittelwertbildung in der Abbildung von Tiefendaten auf eine Matrix, verringert sich die Pixeldichte in Abbildung 25b, weshalb dort die Bedingungen 13 und 14 in der schrägen Fläche erfüllt sind. Sie erlauben keine eindeutige Erkennung der unteren rechten Transportbox. Das Ergebnis aus dem Originalbild unter Verwendung des eigenen Algorithmus weist dabei die am stärksten ausgeprägten Außenkanten auf. Alle sechs betrachteten Verfahren (Canny-, Sobel-Operator und Gleichung 15 mit jeweils dem Originalbild und dem COOM gemappten Bild) lassen folgende Aussagen zu:

- Die Schwellwertbegrenzung ist nur bedingt einsetzbar, da dünne und schräge Kante früh entfernt werden. Dies ist auf die Auflösung des Sensors, den Reflektionseigenschaften des Materials und der Mittelwertbildung benachbarter Pixel durch den Kinect-Sensor zurückzuführen.
- Das COOM gemappte und auf eine Matrix abgebildete Frame, lässt keine eindeutige Kantenerkennung unter Verwendung durch den Canny- oder Sobel-Operator zu. Gleiches gilt für das in Gleichung 15 vorgestellte Verfahren.
- Eine Kantendefinition im Originalbild ist durch den Canny-, Sobel-Operator und dem Verfahren nach 15 möglich. Eine eindeutige Identifikation der außen liegenden Kanten ist aufgrund der Kameraperspektive durch keinen Operator möglich, weshalb die genaue Bestimmung der Eckpunkte in dem Originalbild deutlich erschwert ist.
- Die Mittelwertbildung benachbarter Pixel durch den Kinect-Sensor erschwert die Kantenerkennung.

Aus den zuvor genannten Gründen wird sich dazu entschieden, das Tiefenbild ausschließlich als Bildquelle in der Vorsegmentierung zu verwenden. Mit den dort erfassten Punkten wird im Anschluss das Farbbild als weitere Bildquelle verwendet um die Genauigkeit zu erhöhen. Für die Vorsegmentierung im Tiefenbild sollen zwei Verfahren betrachtet werden. Zum Einen, die naheliegendste Methode durch das Bilden

von konvexen Hüllen und dem Filtern nicht relevanter Objekte, sowie einem genauem Verfahren unter Verwendung der Hough-Transformation und dem eigenen Ansatz nach Gleichung 15 auf das Originalbild.

6.2.1 Ansatz über die konvexe Hülle und Hu-Momenten

Der naheliegendste Ansatz zur Vorsegmentierung der Transportboxen liegt in der Erstellung sowie Filterung von konvexen Hüllen und einer anschließenden Approximation des Rechtecks. Die Verwendung der konvexen Hülle gegenüber der Kontur eines Objektes begründet sich durch die reduzierte Anzahl an Datenpunkten und ist anwendbar aufgrund der Rechteck ähnlichen Struktur der Box. Die Untersuchung einer im Mittelpunkt platzierten großen Box ergibt, dass die durchschnittliche Anzahl an Punkten für die Kontur $77,4$ und für die konvexe Hülle $8,2$ beträgt[3]. Zusätzlich kann die Kontur Teile des Rauschens oder irrelevante Kanten (Vgl. Abbildung 23) enthalten, die Einfluss auf das Moment der Kontur nehmen. Für die Bestimmung der geometrischen Momente von Objekten mit abgerundeten Elementen wird dieser Ansatz als ungeeignet angenommen und die Verwendung der vollständigen Kontur empfohlen. Alle notwendigen Funktionen für die Umsetzung sind in OpenCv enthalten. Damit die Erkennung der konvexen Hüllen eine hohe Trefferrate besitzt, ist eine vollständig geschlossene Kontur des Objektes notwendig. Aus diesem Grund findet nach Anwendung des Canny-Algorithmus, der aufgrund seiner hohen Genauigkeit implementiert wird, eine Dilatation Anwendung. Durch die Erkennung der Konturen und einer anschließenden Approximation von geöffneten Konturen zu geschlossenen Objekten, können die konvexen Hüllen im Tiefenbild erkannt und nach ihren Merkmalen extrahiert werden. Dazu werden die Momente der relevanten Objekte angewendet. Ein geometrisches Moment ist definiert durch [Bro02, Gleichung 3.8]:

$$m_{pq} = \int_x \int_y x^p \cdot y^q \cdot g(x,y) dxdy \text{ mit } p,q = 0,1,2,...$$

Das Integral definiert die Aufsummierung in die Dimensionen x und y und wird in der Literatur teilweise durch die Summe definiert. Ausgehend von einer Kontur in einem binärem Bild beschreibt $g(x,y)$ den Binärwert an der Stelle $x-$ und y. Die Ordnung des Momentes beträgt $j = (p+q)$ und besteht aus $j+1$ Momenten. Die Anteile $x^p\, y^q$ zeigen, dass ein Moment ortsabhängig und somit nicht translationsinvariant ist. Diese Eigenschaft wird durch den Schwerpunkt der Kontur gewonnen, der wiederum durch die ersten Momente definiert wird [Bro02, Gleichung 3.9]:

$$x_s = \frac{\int_x \int_y x \cdot g(x,y) dxdy}{\int_x \int_y g(x,y) dxdy} = \frac{m_{10}}{m_{00}},\ y_s = \frac{\int_x \int_y y \cdot g(x,y) dxdy}{\int_x \int_y g(x,y) dxdy} = \frac{m_{01}}{m_{00}}$$

Das Moment m_{00} wird als Moment 0. Ordnung bezeichnet und besteht aus der Summe aller Bildpunkte, währenddessen dies die Fläche in der Kontur selbst im Binärbild repräsentiert. Die Momente m_{01} und m_{10} definieren das Spalten- und Zeilenmoment und enthalten die Gewichtung in der jeweiligen Dimension. Durch die Verschiebung der Konturwerte in den Schwerpunkt resultiert das zentrale und translationsinvariante Moment [Bro02, Gleichung 3.8]:

$$\mu_{pq} = \int_x \int_y (x - x_s)^p \cdot (y - y_s)^q \cdot g(x,y) dxdy \text{ mit } p,q = 0,1,2,...$$

Eine Translation wirkt sich auf den Anteil $x^p y^q$ und den Schwerpunkten x_s, y_s aus und wird somit ausgeglichen. Um skalierungsinvariante Momente zu erhalten, wird eine Normierung durchgeführt. Für eine Normierung des um α skalierten Bildobjektes $g(x,y)$ mit dem Ergebnis $g'(x',y')$ wird folgende Transformation gebildet [Bro02, Gleichung 3.13]:

$$\iint x'^p y'^q g'(x',y') dx'dy' = \iint (\alpha x)^p (\alpha y)^q g(x,y) \alpha dx \alpha dy = \alpha^{p+q+2} \iint x^p y^q g(x,y) dxdy$$

[3] Die Ergebnisse der Messung sind dem Datenträger zu entnehmen.

Eine Normierung mit dem zentralen Flächenmoment und dem Anteil des Skalierungsfaktors erfolgt durch [Bro02, Gleichung 3.14]:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\frac{p+q+2}{2}}}$$

Durch die Normierung der zentralen Momente sind alle η_{pq} einheitenlos und $\eta_{00} = 0$.
Auf Grundlage und durch Kombination der normalisierten Momente konnte Ming-Kuei-Hu, in der 1962 veröffentlichten Arbeit, Momente finden, die eine Rotationsinvarianz aufweisen [Hu62]. Sie werden als Hu-Momente bezeichnet und sind somit invariant gegenüber der Translation, Rotation, Spiegelung (mit Ausnahme des siebten Hu-Momentes) und Skalierung. Mit Bezug auf die Transportboxen wird daher angenommen, dass die gefundenen Parameter für beide Größen gültig sind (skalierungsinvariant). Dabei ist zu berücksichtigen, dass die Verzeichnung des Objektives eine ortsabhängige, fehlerhafte Darstellung der Form verursacht. Eine Entzerrung des vollständigen Bildes ist rechenintensiv, weshalb Parameter gesucht werden, die eine bestimmte Abweichung tolerieren. Da die Boxen rechteckig sind und somit eine geringe Rundheit zu erwarten ist, wird diese ebenfalls als Maß verwendet, um nicht relevante Objekte zu filtern. Die Rundheit R eines Objektes setzt sich aus dem Moment m_{00} wie folgt zusammen (In Anlehnung an [DK16, Gleichung 140]):

$$R = \frac{4\pi A}{U^2} = \frac{4\pi m_{00}}{U^2}$$

Der Umfang entspricht dabei der Länge der konvexen Hülle bzw. der Konturlänge eines Objektes. Die Analyse der beiden Boxen erfolgt im Bildmittelpunkt und am Außenrand in dem Arbeitsbereich. Die Erfassung der Momente im Randbereich einer Aufnahme erfolgt nicht, da dort das Rauschen einen starken Einfluss auf die Form der konvexe Hülle nimmt und die Messung zu stark beeinflusst wird. Die in dieser Messung erfassten Werte dienen daher als Richtwert und werden in der Umsetzung derart angepasst, dass eine Erfassung der Boxen in allen Bereichen erfolgreich ist. Die Messung ergibt die Werte aus Tabelle 4, wobei die Abnahme des numerischen Wertes mit steigendem Moment auf die steigende Ordnung der Hu-Momente zurückzuführen ist.

Merkmal	Mitte 0°	Mitte 90°	Außen 0°	Außen 90°
h_1	0.170	0.171	0.169	0.176
h_2	0.00211	0.00216	0.00163	0.00430
h_3	1.018e-5	3.130e-5	3.015e-5	4.172e-6
h_4	4.859e-7	1.103e-6	1.162e-7	5.649e-8
h_5	-8.76e-13	-6.48e-12	1.15e-13	-2.50e-14
h_6	-2.21e-8	-5.13e-8	-3.50e-9	-1.54e-9
h_7	-6.32e-13	-2.74e-13	1.85e-13	-1.13e-14
Rundheit	0.810539	0.807813	0.832	0.835
Fläche $[px^2]$	2050	2287.09	2453.38	2648.16

(a) Kleine Transportbox.

Merkmal	Mitte 0°	Mitte 90°	Außen 0°	Außen 90°
h_1	0.172	0.172	0.170	0.176
h_2	0.00239	0.00232	0.00177	0.00406
h_3	2.349e-6	1.134e-6	2.528e-5	3.414e-6
h_4	1.534e-7	8.746e-9	1.530e-7	1.223e-7
h_5	-1.28e-14	-1.62e-16	1.22e-13	-7.43e-14
h_6	-5.372e-9	1.72e-11	-5.071e-9	-5.035e-9
h_7	-9.12e-14	8.56e-16	2.75e-13	-2.69e-14
Rundheit	0.815821	0.814406	0.823	0.827
Fläche $[px^2]$	4429.56	4492.28	4754.32	5037.65

(b) Große Transportbox.

Tabelle 4: Messung der Hu-Momente auf Basis der konvexen Hülle für beide Größen der Transportboxen in unterschiedlicher Ausrichtung und Position.

6.2.2 Ansatz über die Hough-Transformation

OpenCv bietet die Implementation der Hough-Transformation in zwei Varianten an, der herkömmlichen sowie der optimierten ([GJK00]). Das optimierte Verfahren verwendet dabei bekannte Parameter der gesuchten Kanten und bricht mit einer Verletzung der Grenzwerte das Verfahren im aktuellen Pixel ab. Die Vorgabe der gesuchten Längen verursacht somit eine Optimierung. Da es sich bei dem Verfahren dennoch um eine sehr rechenintensive Anwendung handelt, ist die Reduzierung irrelevanter Pixel und eine effektive Hardwareausnutzung notwendig. Für die Umsetzung werden drei Schritte unternommen:

1. Vorsegmentierung durch die konvexe Hülle in AOIs.
2. Verteilung der AOIs auf die Kerne der CPU zur parallelen Bearbeitung.
3. Kantenerkennung nach Gl. 15.

Sind die Linien in den segmentierten Bildern erkannt, folgt eine Schnittpunktbildung aller Linien. Anschließend werden Linien, deren Schnittpunkt nicht in der Nähe der Enden beider Linien liegen, ignoriert. Weiterhin muss der Winkel zwischen zwei Linien im Bereich von 85°-95° liegen, da nur diese Bestandteil eines Rechtecks sein können. Die anschließende Gruppierung der sich orthogonal schneidenden Elemente erlaubt so eine Extraktion der im Bild vorhandenen Rechtecke, auch wenn die Linien unterbrochen sind. Die Umsetzung implementiert somit eine Segmentierung der relevanten Objekte. Ausschließlich rechteckige Objekte, deren Kanten den gefilterten Längen entsprechen, werden berücksichtigt. Eine weitere Filterung der Objekte nach ihren Merkmalen ist somit nicht erforderlich.

6.2.3 Gegenüberstellung und Validierung

Für die Gegenüberstellung der beiden Algorithmen werden diese nach den Kriterien, Zuverlässigkeit, Performance, Genauigkeit und Komplexität beurteilt. Als Bildquelle dient das Schräglagen korrigierte und orignale Tiefenbild in einer reduzierten Auflösung (46,27)-(473,389), (x,y in px). Das Rauschen hat daher nur Einfluss auf die unteren und oberen Bildränder. Durch die spätere Reduzierung des Bildes auf den relevanten Arbeitsbereich ist eine deutliche Rauschreduzierung und geringere Rechenzeit zu erwarten (Das Rauschen wird nicht auf ein mögliches Objekt geprüft). Für die Beurteilung der Zuverlässigkeit werden 15 Bildaufnahmen verwendet und die Anzahl der erkannten Boxen und der fehlerhaft erkannten Objekte erfasst. Hierbei werden acht Boxen (vier Kleine und vier Große) zufällig im Bildbereich platziert. Das selbe gilt für 16 irrelevante Objekte, die durch die Algorithmen nicht erkannt werden dürfen. Die Erfassung der Performance erfolgt über 15 Bilder, indem am Ende der Algorithmen die Zeitdifferenz zwischen Start- und Endzeit errechnet und gemittelt wird. Die Beurteilung der Genauigkeit basiert auf dem händisch ermittelten Abstand zwischen den erfassten Positionen der Ecken sowie den realen Positionen. Eine Mittelwertbildung der Differenz aller Ecken definiert das Ergebnis. Die Komplexitätsbeurteilung basiert auf den Erfahrungen die durch die Umsetzung erlangt werden.

Zuverlässigkeit

	Objekt ist eine Kiste	Objekt ist keine Kiste
Objekt als OK klassifiziert	36	1
Objekt als NOK klassifiziert	0	35

(a) Klassifizierung durch den Ansatz der Hu- Momente.

	Objekt ist eine Kiste	Objekt ist keine Kiste
Objekt als OK klassifiziert	31	3
Objekt als NOK klassifiziert	5	34

(b) Klassifizierung durch den Ansatz der Hough- Transformation.

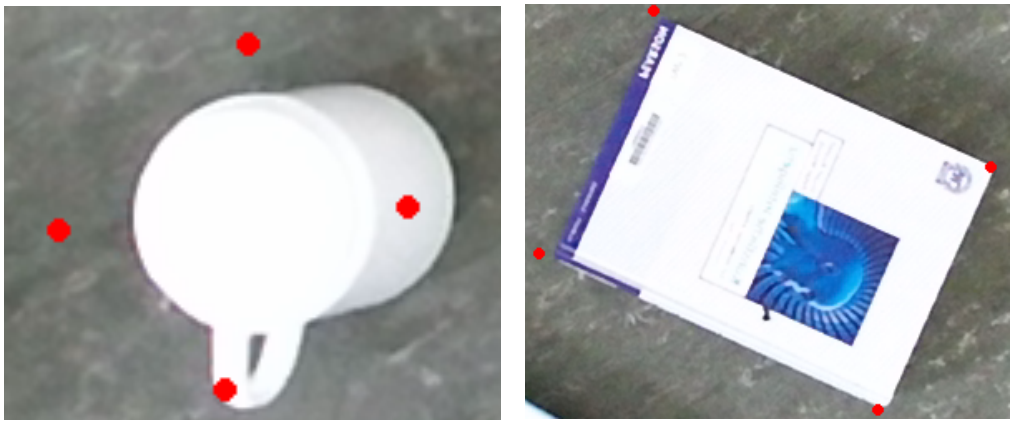
Tabelle 5: Gegenüberstellung der Zuverlässigkeit der Algorithmen zur Erkennung der Boxen.

Werden die erfassten Werte der Algorithmen aus der Tabelle 5verwendet um die „false-negativ“ (FRR, auch als „false reject rate“ bezeichnet) und „false-positiv“ (FAR, auch als „false acceptance rate“ bezeichnet) -Rate zu bestimmen, resultiert:

$$FAR_{Hu} = \frac{1}{36} = 0,027 \; FAR_{Hough} = \frac{3}{37} = 0,081$$

$$FRR_{Hu} = \frac{0}{36} = 0, \; FRR_{Hough} = \frac{5}{36} = 0,139$$

Dabei erkennen beide Ansätze ein in der Bildmitte platziertes Buch, welches eine äquivalente Form wie die Box aufweist, als relevantes Objekt. Der Ansatz über die Hough-Transformation erkennt darüber hinaus eine Tasse im linken Bildbereich als Transportbox (Vgl. Abbildung 26).



(a) Die durch den Hough- Ansatz als Box klassifizierte Tasse.

(b) Das durch beide Ansätze als Box klassifizierte Buch im Bildmittelpunkt.

Abbildung 26: Fehlerhaft klassifizierte Objekte.

Die Erkennung des Buches basiert auf der starken Ähnlichkeit zu einer Box, während für die Tasse nur die Ränder erkannt werden. Dies ist darauf zurückzuführen, dass bereits kleine Linien ungefiltert passieren und die Linien an den Rändern einer Tasse gemeinsam ein Quadrat bilden und somit die erforderlichen Kriterien erfüllen. Die Erkennung basiert dabei auf einem Zufall zusammen mit der perspektivischen Darstellung, da andere runde Objekte in dem Bild nicht als „OK“ klassifiziert werden. Der höhere Wert für FRR_{Hough} resultiert aus dem im Randbereich auftretenden Rauschen. Da die verauschten Pixel in die Segmentierung mit einfließen, entstehen weitere Linien, die wiederum nicht die Bedingung des Schnittwinkels zwischen 85°-95° erfüllen. In Situationen in denen das Rauschen nicht einwirkt, wurden die Boxen ebenfalls durch den Ansatz der Hough-Transformation durchgehend erkannt. Das Rauschen wirkt sich in der Bewertung für den Ansatz über die Hu-Momente nicht aus, da eine durch das Rauschen vergrößerte Box als „OK“ beurteilt wird. Insgesamt besitzt der Ansatz über die Hu-Momente eine geringere FA- und eine geringere FR-Rate und ist gegenüber dem Ansatz mit der Hough-Transformation zuverlässiger.

Performance

Die Messung der Algorithmen-Laufzeiten mit einer großen Box im Mittelpunkt ergibt im Durchschnitt[4]:

$$t_{Hu1} = 1,87\,\text{ms}, t_{Hough1} = 22,56\,\text{ms}$$

Unter Platzierung von acht Boxen im Bild ergibt sich:

$$t_{Hu8} = 2,25\,\text{ms}, t_{Hough8} = 29,65\,\text{ms}$$

Die Messung zeigt, dass der Ansatz über die Hu-Momente gegenüber dem Ansatz mit der Hough-Transformation deutlich schneller ist und unter Verwendung von mehreren Boxen keinen bzw. einen sehr geringen Anstieg in der Laufzeit aufweist. Der Anstieg des Ansatzes über die Hough-Transformation von t_{Hough1} auf t_{Hough8} ist darauf zurückzuführen, dass die Laufzeit des Thread basierten Algorithmus durch den längsten Thread definiert wird. Während die Box der Messung t_{Hough1} gerade im Bild platziert wird, verursacht eine schräg positionierte Box (wie in Messung t_{Hough8}) ein größeres „Area Of Interest“ (AOI), weshalb eine höhere Laufzeit entsteht. Die Laufzeitmessung beginnt dabei an dem Punkt, an dem sich beide Algorithmen voneinander unterscheiden. Die Gruppenbildung der erkannten Linien zu einem Rechteck im Ansatz über die Hough-Transformation ist daher Bestandteil der Laufzeitmessung.

[4] Unter Verwendung der Debug DLLs aller Frameworks

Genauigkeit

Box Nr.	Größe	Min. Abw. [px]	Max. Abw. [px]	⌀ [px]
1	Gr.	11,31	29,15	17,82
2	Kl.	7,28	19,23	12,56
3	Gr.	3,60	23,56	12,44
4	Kl.	10,20	26,17	16,26
5	Kl.	6,32	16,76	10,49
6	Kl.	6,08	19,21	11,72
7	Gr.	9,85	31,32	20,21
8	Gr.	12,81	23,26	17,08
⌀ [px]		8,37	23,54	14,82

(a) Abweichungen des Ansatzes über die Hu-Momente.

Box Nr.	Größe	Min. Abw. [px]	Max. Abw. [px]	⌀ [px]
1	Kl.	7,21	16,28	12,22
2	Kl.	7,61	14,56	10,50
3	Gr.	7,21	18,68	14,48
4	Gr.	10,20	12,80	11,65
5	Gr.	8,48	12,73	10,46
6	Gr.	2,24	6,71	4,63
7	Kl.	8,60	18,87	11,56
8	Kl.	8,25	12,81	10,19
⌀ [px]		7,47	14,69	10,71

(b) Abweichungen des Ansatzes über die Hough- Transformation.

Tabelle 6: Gegenüberstellung der Genauigkeit der Algorithmen zur Erkennung der Boxen im Tiefenbild.

Für die Bestimmung der Genauigkeit wird händisch die Differenz zwischen der Ecke einer Box und der durch die Algorithmen erfasste Ecke für acht Boxen erfasst und gemittelt[5]. Das Farbbild dient dabei als Bildquelle. Für den Ansatz über die Hough-Transformation resultiert eine durchschnittliche Differenz von $10,71\,\text{px}$, während die maximale Abweichung im Durchschnitt $14,69\,\text{px}$ und nahezu der durchschnittlichen Abweichung über den Ansatz der Hu-Momente mit $14,82\,\text{px}$ entspricht. Die maximale Abweichung beträgt hierbei im Schnitt $23,54\,\text{px}$ und die minimale $8,37\,\text{px}$. Es lässt sich daher schlussfolgern, dass der Ansatz über die Hough-Transformation genauer ist, da die maximale Abweichung und die durchschnittliche Abweichung gegenüber dem Ansatz über die Hu-Momente um 60% und 38% geringer ist. Die durchschnittliche minimale Abweichung beider Algorithmen unterscheidet sich nur sehr gering voneinander. Die Ursache der starken Abweichungen in dem Ansatz über die Hu-Momente ist in der Vorsegmentierung zu finden. Um alle vier Kanten der Box zu erkennen, kann nur ein sehr geringes Tiefen- Thresholding (15 mm) angewendet werden. Gleiches gilt für den Canny-Algorithmus, dessen Ergebnisse durch die Länge einer Kante gefiltert werden und bei einem zu starken Filter relevante Kanten filtert. Diese Faktoren verursachen zusammen mit der perspektivischen Darstellung der Kisten im Randbereich sowie der Dilatation eine Abweichung der tatsächlichen Kanten. Der Algorithmus über die Hough-Transformation filtert Kanten, deren Winkel zueinander nicht in einem Bereich von 85°-95° liegen. Aufgrund der geringen Länge die eine Kante haben kann z.B für eine kleine Box, erfüllen mehrere Kanten die Kriterien und tragen durch die Mittelwertbildung der daraus resultierenden Ecken einen Anteil zu dem Ergebnis bei. Die Filterung der kleineren Kanten oder die Verringerung des Winkelbereichs erzeugen genauere Ergebnisse. Dadurch werden wiederum nicht alle Boxen erkannt, da in anderen Situationen zu wenige Kanten die Kriterien erfüllen. Zu beachten ist, dass die Differenzmessung zwischen dem Soll- und dem Istpunkt mit Unsicherheiten behaftet ist. Es entstehen Ablesefehler, indem die Sollposition nicht exakt ($\pm 5\,\text{px}$) bestimmt werden kann und durch Rundungen der Eckpunkte beim Cast von dem Datentyp Point2f zu dem Typ Point2i (float zu integer) ($\pm 0.5\,\text{px}$) . Weiterhin ist die Auflösung des Tiefenbildes gegenüber dem Farbbild geringer, weshalb die Auflösung durch das Tiefenbild definiert ist. Unsicherheiten treten ebenfalls durch den Coordinate-Mapper auf, der die Transformation der Tiefenpunkte auf das Farbbild übernimmt.

[5] Die originale Aufnahme inklusive der Zuordnungen zu den Boxen ist dem Datenträger dieser Arbeit zu entnehmen.

Komplexität

Die Umsetzung zeigt, dass die Hough-Transformation sehr einfach durch die bestehende Funktion im OpenCv-Framework realisiert werden kann. Im Anschluss erfolgt die Zuordnung der Linien zu Rechtecken, indem Linien Gruppen zugeordnet werden, sofern ihr Schnittwinkel im Bereich von 90° liegt. Schneiden sich Linien zweier Gruppen, werden diese zu einer zusammengefasst bis vier Linien ein Rechteck bilden. Da mehrere Rechtecke gefunden werden, erfolgt anschließend eine Aussortierung und eine Mittelwertbildung der Ecken.
Der Ansatz über die Hu-Momente basiert nahezu vollständig auf in OpenCv implementierten Funktionen und ist daher deutlich schneller umzusetzen.

Zusammenfassung

Ansatz	Performance	Genauigkeit	Stabilität	Komplexität
Hough- Transformation	-	o	o	o
Hu-Momente	+	-	+	+

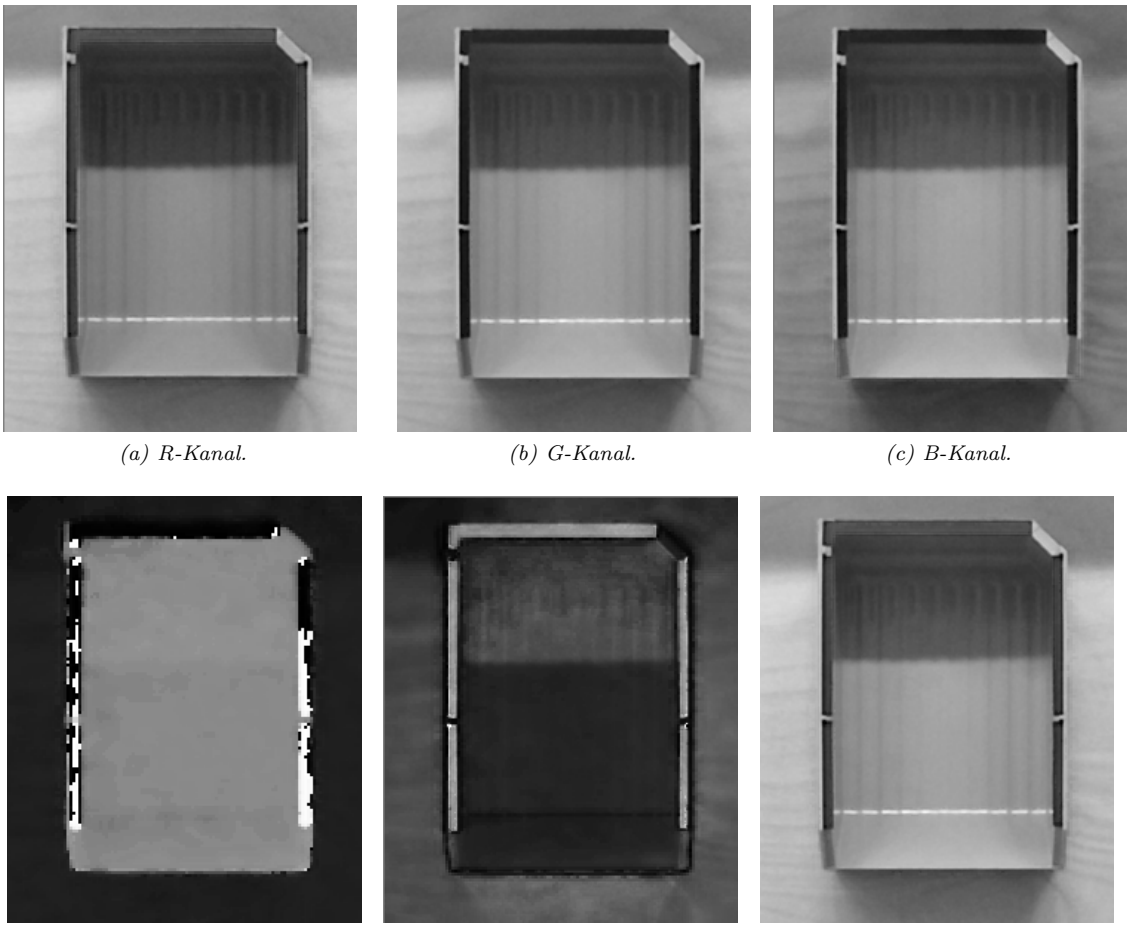
Tabelle 7: Gegenüberstellung der umgesetzten Algorithmen im Tiefenbild.

Die Gegenüberstellung in der Tabelle 7 zeigt, dass der Ansatz über die Hu-Momente zwar ungenauer ist, in der Umsetzung jedoch einfacher und im Leistungsverbrauch Ressourcen-sparender ist. Zusätzlich ist die Klassifizierung zuverlässiger, da diese nur einmal mit einem sehr ähnlichen Objekt falsch lag obwohl eine Entzerrung des Bildes nicht erfolgte. Die Hough-Transformation ist zwar genauer, weist aber weiterhin Abweichungen bei der Bestimmung der Eckpunkte auf. Die Umsetzung zeigt in beiden Fällen, dass eine höhere Genauigkeit zwar realisierbar ist, gleichzeitig aber die Trefferquote der relevanten Objekte reduziert wird, da weniger Objekte die strengeren Kriterien erfüllen. Beide Ansätze kommen zu keinem Ergebnis, das ausreichend genau ist um eine Box zu fassen. Aus diesem Grund wird das Farbbild als weitere Informationsquelle verwendet.

6.3 Merkmalsextraktion zur Detektion der Transportboxen im Farbbild

Die Farbe grau setzt sich im RGB-Farbraum durch gleiche Anteile an rot, grün und blau zusammen. Die Differenzierung zwischen einer grauen Box und einem roten Streifen ist daher erschwert. Aus diesem Grund erfolgt die Umwandlung der Farbinformationen aus dem RGB- in den HSV-Farbraum.

6.3.1 HSV-Farbraum



(a) R-Kanal. *(b) G-Kanal.* *(c) B-Kanal.*

(d) H-Kanal. *(e) S-Kanal.* *(f) V-Kanal.*

Abbildung 27: Gegenüberstellung der Kanäle des RGB- und des HSV-Farbraumes einer Box mit roter Markierung an den Rändern.

Während sich der Farbwert im RGB-Farbraum durch den Anteil der Farben rot, grün und blau zusammensetzt, erfolgt dies im HSV-Raum durch den Farbton (z. engl. Hue), der Sättigung (z. engl. Saturation) und der Helligkeit (z. engl. Value). Eine weitere Bezeichnung für den HSV-Raum ist der HSB-Raum (B für Brightness, z. Dt. Helligkeit). Die Darstellung in Abbildung 27 verdeutlicht den Zusammenhang zwischen den Parametern H,S und V. Die Differenzierung der Farbparameter im HSV-Raum ist für die Unterscheidung zwischen einer grauen Box und einem farbigen Streifen simpler, da sie sich vor allem in der Sättigung und dem Farbton deutlich unterscheiden. Die Aufnahme zeigt zusätzlich, wie sich ein Schattenwurf in den Kanälen auswirkt. Während sich der Schatten in den RGB-Kanälen durch die verringerten Farbwerte dem niedrigen Wert der roten Markierung annähert und

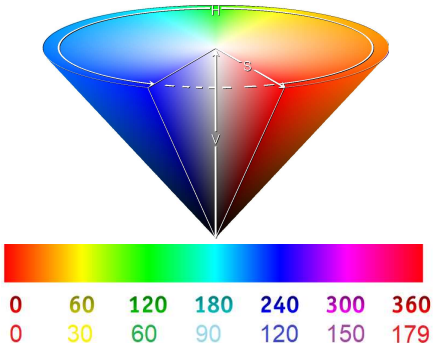


Abbildung 28: Visualisierung des HSV-Raumes. Die untere Farbskala ist in Grad aufgetragen (Normale Skala (oben) / Skala in OpenCV (unten)) [wik].

eine Detektion erschwert, ist eine klare Abgrenzung im S- und H-Kanal zu erkennen. In dem H-Kanal wird nach Abbildung 27d der rote Streifen durch weiße (179) oder schwarze (0) Werte dargestellt. Die Ursache ist in den Farbwerten für die Farbe rot zu finden, da diese im Bereich von ca. 0°-10° und im Bereich von 170°-179° (vgl. Abbildung 28) liegt. Für die Anwendung der Detektionsmethoden erfolgt im ersten Schritt eine Konvertierung aus dem RGB- in den HSV-Farbraum. Dabei beschreibt C_{max} den maximal zu erreichenden Wert in den RGB-Kanälen (in der Regel beträgt $C_{max} = 255$) und die Sättigung berechnet sich nach [BB06, Gl. 12.10]

$$S_{HSV} = \begin{cases} \frac{C_{rng}}{C_{maxx}} & \text{für } C_{high} > 0 \\ 0 & \text{sonst} \end{cases}$$

daher aus dem maximalen Wert der drei Kanäle: $C_{high} = \max(R, G, B)$ sowie dessen Differenz zu dem minimalen Wert der drei Kanäle $C_{min} = \min(R, G, B)$ durch $C_{rng} = C_{high} - C_{low}$ [BB06, Gl. 12.12]. Die Helligkeit ist definiert durch [BB06, Gl. 12.11]:

$$V_{HSV} = \frac{C_{high}}{C_{max}}$$

Der Farbwert H_{HSV} ergibt sich aus drei Farbkomponenten rot $R´$, grün $G´$ und blau $B´$, die normiert wie folgt lauten [BB06, Gl. 12.13]:

$$R' = \frac{C_{high} - R}{C_{rng}}$$

$$G' = \frac{C_{high} - G}{C_{rng}}$$

$$B' = \frac{C_{high} - B}{C_{rng}}$$

Abhängig davon, aus welchem Kanal (RGB) der maximale Wert C_{maxx} entstammt, resultiert [BB06, Gl. 12.14]:

$$H' = \begin{cases} B' - G' & \text{wenn } R = C_{high} \\ R' - B' + 2 & \text{wenn } G = C_{high} \\ G' - R' + 4 & \text{wenn B} = \text{C}_{\text{high}} \end{cases}$$

Dieser liegt im Bereich von -1 bis 5 und repräsentiert durch die Normierung in den Bereich 0 bis 1 den Farbwert H_{HSV} [BB06, Gl. 12.15]:

$$H_{HSV} \leftarrow \frac{1}{6} \cdot \begin{cases} (H' + 6) & \text{für } H' < 0 \\ H' & \text{sonst} \end{cases}$$

Durch die Multiplikation mit 360° erfolgt die Darstellung durch ein Winkelintervall von 0° bis 360°. Die Umrechnung zeigt, dass eine unbunte Farbe wie grau ($R = G = B$) , $C_{high} = C_{low}$ und $C_{rng} = 0$ verursacht. In diesem Fall ist H_{HSV} nicht definiert und es gilt $S_{HSV} = 0$, wodurch die deutliche Abgrenzung der roten Markierung von dem Grau der Box in Abbildung 27e zu erklären ist. Da die Farbe Rot in dem Grenzintervall für H_{HSV} liegt , wird für die praktische Umsetzung das Winkelintervall auf 0° bis 179° (definiert durch OpenCv) verringert, um im Anschluss die Kosinus-Funktion anzuwenden:

$$H_{cos} = 180° \cdot cos(H_{HSV}) \tag{16}$$

Somit beträgt der Wert H_{cos} im Bereich 0°-10° und 170°-179° ca. 180° und bei stärkeren Abweichungen ca. Null. Eine Multiplikation mit S_{HSV} verstärkt die roten Merkmale und schwächt die graue Farbe der Box:

$$V = \alpha \cdot H_{cos} \cdot S_{HSV} \tag{17}$$

Hierbei dient α der Normierung, um im Intervall von 0 bis 255 zu bleiben, und beträgt $\frac{1}{180}$. Dies ist allerdings abhängig von der gewählten Auflösung im HSV-Raum.

Validierung

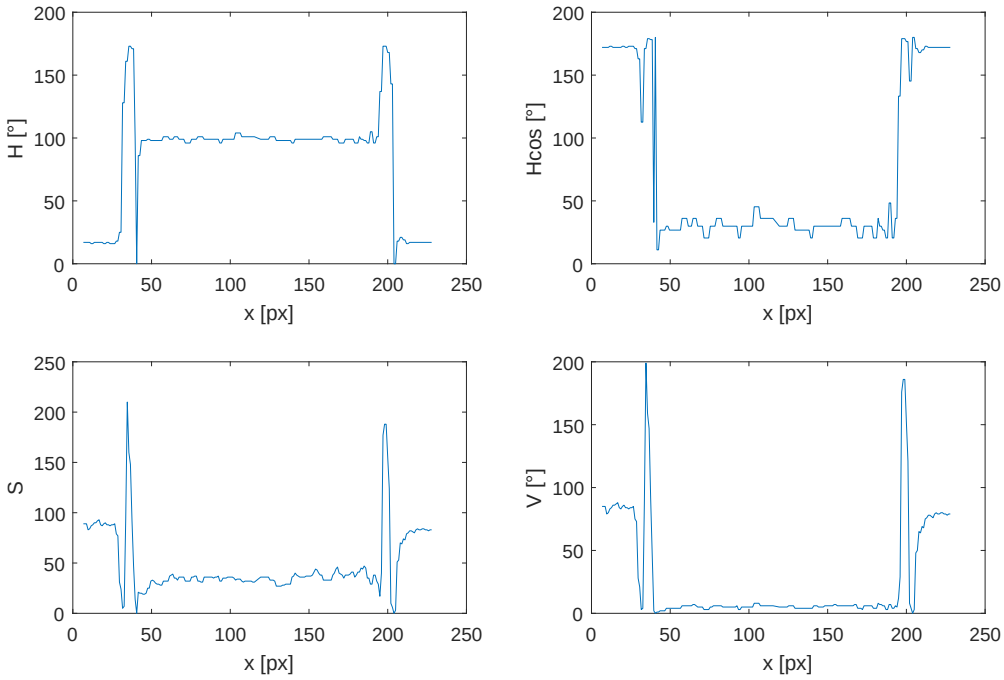


Abbildung 29: Gegenüberstellung der HSV-Anteile H und S sowie dem Faktor V .

Für die Validierung wird über eine Box das Profil der Werte H, H_{cos}, S und V unter eingeschalteter Beleuchtung erfasst und gegenübergestellt. Abbildung 29 zeigt, dass für das H-Profil eine deutliche Trennung zwischen der Box (zwischen x=45 und x=180) und der roten Markierung möglich ist. Gleichzeitig tritt der linke rote Streifen (das linke Maximum im Profil) in den Farbbereich Richtung orange, weshalb das Maximum durch ein Minimum unterbrochen wird (Rot liegt im Grenzintervall des H_{HSV} Kanals). Die in Gleichung 16 angewendete Korrektur ist in dem oberen rechten Plot der Abbildung 29 zu erkennen. Das vorherige Minimum des roten Streifens ist hier dem Maximum

angeglichen. Gleichzeitig ist das Grau der Box minimiert und der Wert für den Tisch auf dem die Messung stattfand maximiert. Zweiteres ist durch den braunen/rötliche Farbton des Tisches zu erklären, der durch die Kosinus-Funktion verstärkt wird. Diese Verstärkung tritt im S-Profil nicht auf (vgl. linker unterer Plot in Abbildung 29), weswegen die Multiplikation mit dem H_{cos}-Profil den Anteil des Tisches reduziert. Dem Profilverlauf für V ist zu entnehmen, dass die Merkmale der roten Markierung einen starken Anstieg verursachen. Für die Erfassung der Merkmale bieten sich daher zwei Verfahren an. Zum Einen können die Maxima durch Bilden des Differentials über das Profil erkannt werden, zum Anderen durch die Kreuzkorrelation mit einem Referenzsignal, dessen Form dem Profilverlauf im Bereich des Merkmals entspricht. Nach einer Vorsegmentierung im Farbbild werden beide Ansätze umgesetzt und gegenübergestellt.

6.3.2 Vorsegmentierung

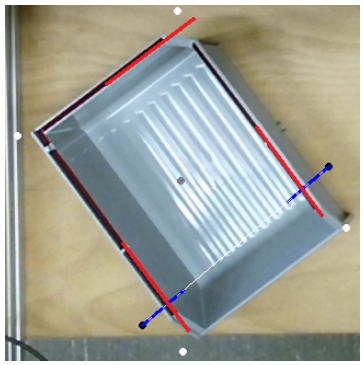


Abbildung 30: Darstellung der Auswertung einer Box im Farbbild mithilfe des Ansatzes über die Korrelation.

Die Erkennung der Kanten im Farbbild soll skalierbar und recheneffizient sein. Aus diesem Grund werden nur relevante Teile im Farbbild untersucht, indem über die Box mehrere Profile erfasst werden. Da durch die Vorsegmentierung im Tiefenbild bekannt ist, wo sich die Ränder der Box ungefähr befinden, werden Farbprofile nur in den zu erwartenden Bereichen erfasst. Dazu wird im ersten Schritt das AOI aus dem Tiefenbild vergrößert. Die weiteren Schritte können der Abbildung 30 entnommen werden, in der die Erfassung eines Profils verdeutlicht ist. Auf Basis der Tiefenbildpunkte (weiße Punkte) werden Linien über die Box gebildet (weiße Linie) und die ersten sowie letzten Farbwerte (blaue Punkte) auf dieser Linie erfasst. Der Inhalt einer Box nimmt daher nur an den Randbereichen auf die spätere Auswertung Einfluss. Die roten Linien repräsentieren das Ergebnis über die Auswertung des Farbbildes mithilfe der Korrelation. Der Anwender kann über die Anzahl der Messlinien sowie der Anzahl an Messpunkten die Genauigkeit und Performance beeinflussen, weshalb das System skalierbar ist. Die so erfassten Werte werden der Differential- und Kreuzkorrelationsfunktion übergeben.

6.3.3 Differential

Die Ableitung der Werte für $V(m)$ nach der Position $m(x, y)$ lautet:

$$\Delta_V = \frac{dV(m)}{dm}$$

An der Stelle mit einer hoher Steigung werden Maxima erzeugt. Diese können verwendet werden um den Mittelpunkt zwischen zwei Maxima zu berechnen, der die Mitte der roten Markierung beschreibt.

6.3.4 Kreuzkorrelationsfunktion

Die Kreuzkorrelationsfunktion gibt Aufschluss darüber, in welchem Maß zwei übereinander liegende Funktionen miteinander übereinstimmen. Die diskrete Kreuzkorrelation kann daher verwendet werden, um das bekannte Profil einer Kante im gemessenen Profil für V wiederzufinden. Zusätzlich handelt es sich um ein Verfahren zur Bestimmung des optischen Flusses, ohne Reaktion auf zeitliche Änderungen der Lichtverhältnisse [Jäh12, Seite: 475]. Die diskrete Kreuzkorrelationsfunktion ist dabei wie folgt definiert:

$$R_{xy}[n] = \sum_{m=-\infty}^{\infty} x[m]\, y[m+n]$$

Übertragen auf die gewünschte Anwendung, folgt:

$$R_{VB}(n) = \sum_{m=1}^{V_x} V(m) \cdot B(m+n)$$

Dabei enthält $V(m)$ den $\alpha \cdot H_{cos} \cdot S_{HSV}$ Wert an der Stelle m (x- ,y-) in der Vorlage, $B(m+n)$ den Wert an der Stelle $m+n$ im neu aufgenommenem Profil und V_x die Anzahl der Werte die als Vorlage dienen. Durch einen Offset in der Position n wird die Vorlage $V(m)$ durch das Profil der Aufnahme $B(m)$ geschoben, indem n erhöht wird. Dabei wird jedes mal $R_{VB}(n)$ bestimmt. Anschließend definiert der höchste Wert aller erfassten $R_{VB}(n)$, die Position an der die Vorlage mit dem gemessenem Profil am meisten korreliert und die Kante erwartet wird.

6.3.5 Gegenüberstellung und Validierung

Für die Gegenüberstellung der beiden Algorithmen werden diese nach den Eigenschaften Zuverlässigkeit, Performance, Genauigkeit und Komplexität beurteilt. Zur Erfassung der Laufzeit werden acht Boxen (vier kleine und vier große) sowie eine Box im Bildbereich platziert und durch den Algorithmus über die Hu-Momente im Tiefenbild vorsegmentiert. Die so erstellen AOI´s werden an beide Algorithmen übergeben und die Laufzeit ab dem Programmpunkt erfasst, an dem sich beide Ansätze voneinander unterscheiden. Die Vorverarbeitung in beiden Algorithmen ist identisch. Beide Prozesse sind dabei vollständig parallelisiert und werden auf Threads aufgeteilt (1 AOI aus dem Tiefenbild je Thread). Die Mittelwertbildung der Laufzeit aus 16 Messungen repräsentiert das Ergebnis. Die Beurteilung der Genauigkeit erfolgt händisch, indem in zwei unterschiedlichen Situationen mit jeweils unterschiedlicher Ausrichtung und Positionierung von acht Boxen, zwei Aufnahmen erfasst werden. Die Abweichung zwischen Soll- und Istposition wird erfasst sowie deren Mittelwert gebildet und gegenübergestellt. Die Komplexität der Umsetzung resultiert auf den eigenen Einschätzungen durch die Umsetzung.

Performance

Die Messung der Algorithmen-Laufzeiten mit einer großen Box im Mittelpunkt ergibt im Durchschnitt[6]:

$$t_{Diff1} = 9\,\text{ms}, t_{Kreuz1} = 9,5\,\text{ms}$$

Unter Platzierung von acht Boxen im Bild ergibt sich:

$$t_{Diff8} = 62,18\,\text{ms}, t_{Kreuz8} = 40,12\,\text{ms}$$

Die Messung zeigt, dass der Ansatz über die Kreuzkorrelation gegenüber dem Ansatz mit der Differentialbildung unter Verwendung eines Objektes nahezu gleich performant ist. Werden acht Objekte detektiert, ist das Verfahren über die Kreuzkorrelation gegenüber dem Ansatz des Differentials ca. 40% schneller.

[6] Unter Verwendung der Debug DLLs aller Frameworks

Genauigkeit und Robustheit

Box Nr.	Größe	Min. Abw. [px]	Max. Abw. [px]	⊘[px]
1	Gr.	4,12	92,2	26,93
2	Kl.	2,83	4,12	3,67
3	Kl.	0	2,83	1,27
4	Gr.	0	2,26	0,56
5	Gr.	7,07	22,20	16,66
6	Kl.	2,24	4,48	3,17
7	Kl.	0	2,24	1,12
8	Gr.	2,24	3,60	2,58
⊘ [px]		2,32	16,74	6,74

(a) Abweichungen des Ansatzes über das Differential.

Box Nr.	Größe	Min. Abw. [px]	Max. Abw. [px]	⊘[px]
1	Gr.	2,24	2,24	2,24
2	Kl.	2,24	3,16	2,47
3	Kl.	2,24	2,83	2,53
4	Gr.	0	2,83	1,27
5	Gr.	0	2,24	1,12
6	Kl.	2,24	3,16	2,61
7	Kl.	0	2,83	1,97
8	Gr.	0	2,83	1,82
⊘ [px]		1,12	2,74	2,00

(b) Abweichungen des Ansatzes über die Kreuzkorrelation.

Tabelle 8: Gegenüberstellung der Genauigkeit der Algorithmen zur Erkennung der Boxen im Farbbild für die erste Testsituation.

Box Nr.	Größe	Min. Abw. [px]	Max. Abw. [px]	⊘[px]
1	Gr.	23,02	25,18	24,18
2	Kl.	0	3,16	1,90
3	Kl.	2,24	3,60	2,96
4	Gr.	2,24	2,24	2,83
5	Gr.	26,08	81,02	53,80
6	Kl.	2,24	4,24	2,88
7	Kl.	2,24	3,16	2,61
8	Gr.	2,24	4,47	3,14
⊘ [px]		7,53	15,96	11,72

(a) Abweichungen des Ansatzes über das Differential.

Box Nr.	Größe	Min. Abw. [px]	Max. Abw. [px]	⊘[px]
1	Gr.	0	2,24	1,12
2	Kl.	0	2,24	1,68
3	Kl.	0	4,24	1,77
4	Gr.	0	2,24	1,12
5	Gr.	2,24	3,16	2,47
6	Kl.	0	2,83	0,71
7	Kl.	2,24	3,16	2,76
8	Gr.	2,24	3,16	2,47
⊘ [px]		0,84	2,90	1,76

(b) Abweichungen des Ansatzes über die Kreuzkorrelation.

Tabelle 9: Gegenüberstellung der Genauigkeit der Algorithmen zur Erkennung der Boxen im Farbbild für die zweite Testsituation.

Die Tabellen 8 und 9 zeigen die Ergebnisse der Genauigkeit in zwei verschiedene Situationen. Jede Situation beschreibt differenzierte Positionen und Ausrichtungen der unterschiedlich großen Boxen[7]. Für die Erfassung der Eckpunkte werden 30 Linien mit je 100 Messpunkten (50 am Anfang und am Ende einer Linie) betrachtet. Der Mindestwert beträgt $\Delta_V \geq 0$ für das Differential und $R_{VB}(n) \geq 28000$ für die Korrelation. Beide Messungen werden unter eingeschalteter Beleuchtung durch geführt.
Die Ergebnisse (Tabelle 8 und 9) zeigen das beide Ansätze gegenüber der Erfassung im Tiefenbild im Durchschnitt eine deutliche höhere Genauigkeit aufweisen. Zu erkennen ist, dass der Ansatz über das Differential nicht in jeder Messung den Mittelpunkt der roten Markierung findet und in der Linienapproximation eine starke Abweichung verursacht. Hierin begründen sich die Ausreißer der

[7] Die originalen Bildaufnahmen und Messungen der einzelnen Boxen sind dem Datenträger zu entnehmen.

Messungen für die Box 1 in Tabelle 8.a und die Ausreißer für Box 5 in den Tabelle 8.a und 9.a. Die maximale Abweichung des Ansatzes über die Kreuzkorrelation beträgt nur $3,16\,\text{px}$, weshalb dieser Ansatz gegenüber dem Ansatz mit dem Differential robuster ist. Zusätzlich ist die durchschnittliche Abweichung der Kreuzkorrelation in beiden Testsituationen um mindestens $4\,\text{px}$ besser und genauer (vgl. Durchschnitt Tabelle 8). Auffällig ist, dass in dem Ansatz über das Differential die hohen Abweichungen jeweils bei der Detektion einer großen Box auftreten. Ein Zusammenhang konnte nachträglich nicht gefunden werden, jedoch befinden sich in beiden Messungen die Boxen im linken Bereich der Aufnahme und besitzen einen sehr schlechten Wert aus der Vorsegmentierung im Tiefenbild. Eine Ursache, die reproduziert werden kann ist, dass die Erfassung des Farbprofils auf der roten Markierung endete. Für das Differential werden die zwei höchsten Maxima verwendet, von dem in diesem Fall Eines fehlt. Die Kreuzkorrelation kann dennoch im Mittelpunkt der Markierung die höchste Korrelation aufweisen und somit einen Ausreißer unterbinden. Daraus folgt, dass der Ansatz über die Korrelation auch bei unvollständigen Maxima robuster ist. Diese Situation kann durch Erhöhen der Anzahl an Messpunkten vermieden werden. Gleichzeitig wird dadurch die Laufzeit des Algorithmus erhöht. Da es sich bei der Lichtquelle um eine Gasentladungslampe handelt, kommt es in dem Farbbild zu einem Flimmern. Da keine Synchronisation erfolgt, sind einige Bilder aufgrund des $50\,\text{Hz}$ Netzes dunkler (Nulldurchgang der Phase) als Andere (Maximum der Phase). Ein Einfluss auf die Erfassung konnte nicht ermittelt werden. Beide Ansätze weisen keine Sprünge in den dunkleren Messungen auf. Zu beachten ist, dass die Erfassung der Punkte mit Fehlern behaftet ist. Zum Einen durch das händische Ablesen ($\pm 1\,\text{px}$) zum Anderen durch die Umwandlung der Schnittpunkte von dem Datentyp „float“ in den Datentyp „int“ ($\pm 0.5\,\text{px}$).

Zusammenfassung

Ansatz	Performance	Genauigkeit	Stabilität	Komplexität
Differential	-	o	o	+
Kreuzkorrelation	o	+	+	+

Tabelle 10: Gegenüberstellung der umgesetzten Algorithmen im Farbbild.

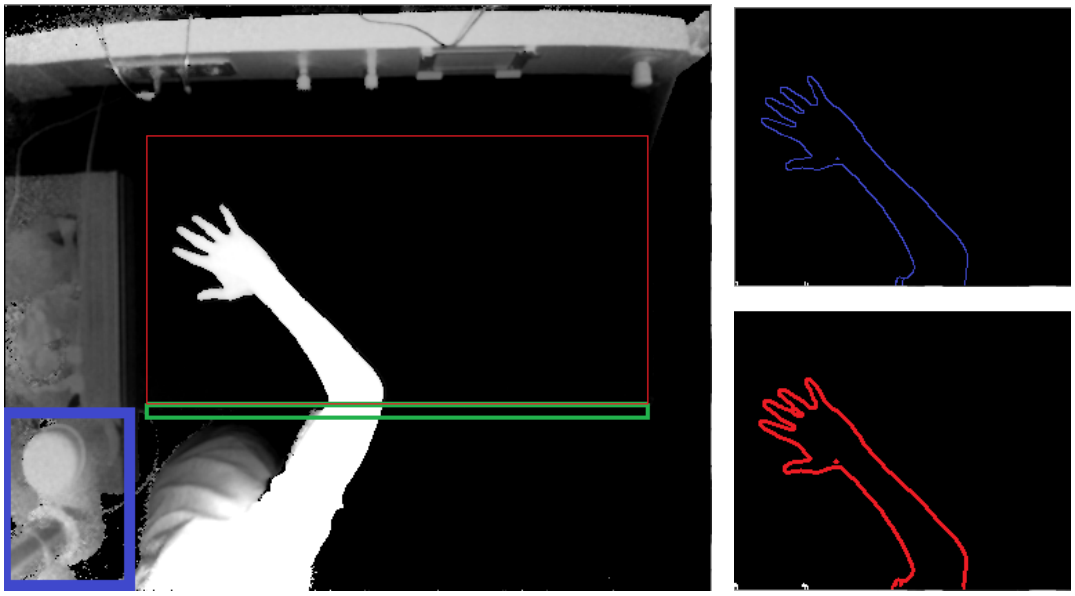
Die Gegenüberstellung zeigt, dass der Ansatz über die Kreuzkorrelationsfunktion genauer, robuster und performanter gegenüber dem Ansatz über das Differential ist. Der Ansatz erlaubt Genauigkeiten bis zu $2,24\,\text{px}$ und maximale Abweichungen von $3,16\,\text{px}$. Die Umsetzung beider Algorithmen ist mit einigen Vorkenntnissen in der Programmierung einfach umzusetzten. Die Algorithmen unterscheiden sich nur in der Zeilenanzahl, jedoch nicht in der Komplexität[8]. Durch eine Mittelwertbildung der Eckpunkte über mehrere Bilder ist außerdem eine höhere Genauigkeit zu erwarten.

[8] Beide Algorithmen sind in dem Programm auf dem Datenträger in der Klasse „armestimator“ zu finden.

6.4 Merkmalsextraktion zur Detektion der menschlichen Gliedmaßen

Die Erkennung der Transportboxen ist eine für diese Aufgabenstellung individuelle Problematik, während für die Erkennung einer Hand in einem Farbbild bereits Lösungen existieren. Die Erkennung der Hand im Tiefenbild einer Kinect v. 1 wird in [FP11] erläutert. Der hier verwendete Ansatz setzt voraus, dass die zu detektierende Hand näher am Sensor ist als andere Objekte. Durch OpenCv werden anschließend Konturen erkannt und deren konvexe Hülle gebildet. Die stärksten Abweichungen zwischen der konvexen Hülle und der Hand sind in den Fingerinnenräumen zu finden. Sie können durch die Funktion *convexityDefects()* in OpenCv erfasst werden. Gemeinsam mit dem Schwerpunkt der konvexen Hülle, welche den Mittelpunkt der Handfläche repräsentiert, kann eine Hand rekonstruiert werden. Die Verwendung eines Kalman-Filters stabilisiert die Positionsschätzung. Dieser Ansatz setzt voraus, dass die Hand geöffnet ist und sich in der Nähe des Sensor befindet sowie, dass die Finger ausreichend gespreizt sind. Dabei wird ein idealer Arbeitsbereich mit einer Distanz von 70 cm zwischen Hand und Sensor angegeben [FP11, Seite: 319]. Durch die höhere Auflösung der Kinect v. 2 wird angenommen, dass diese Distanz mit dem aktuellen Sensor größer ist. Da der Mensch mit den Transportboxen arbeitet, entfällt dieser Ansatz durch die Bedingung, dass die Finger nicht geschlossen sein dürfen sowie der Tatsache, dass andere Objekte wie der Roboter näher an der Kamera sind. Das in [BNC+16] vorgestellte Verfahren dient der Erkennung von Mensch und Roboter durch vier TOF-Kameras. Diese werden dabei schräg zu der Arbeitsfläche platziert, sodass mehrere Strukturelemente der Person erkannt werden. Das dabei verwendete Verfahren basiert auf der Nutzung der OpenNi-Bibliothek, die in dem Kinect-SDK Anwendung findet. Dieser Ansatz basiert auf dem Ziel die Person vollständig zu erkennen und umfasst einen umfangreicheren Systemaufbau, weswegen dieser Ansatz entfällt. Das in [Kle16] entwickelte Minimal-Programm enthält die Möglichkeit, den Menschen in dem Tiefenbild zu erkennen, wenn dieser frontal vor der Kamera steht und basiert ebenfalls auf der OpenNi-Bibliothek. Die Ausführung des Programms in der Arbeitsumgebung zeigt, dass eine reine Erkennung der Arme und der Hand nicht möglich ist und eine Person vollständig von dem Sensor erfasst werden muss. Aus diesem Grund werden mögliche Alternativen gesucht.

6.4.1 Vorsegmentierung



(a) Arbeitsbereich (rot), Sicherheitsbereich (grün) und Roboter (blau) mit einem Arm im Arbeitsbereich.

(b) Ergebnis des Canny-Algorithmus (blau) und Ergebnis der anschließenden Dilatation (rot).

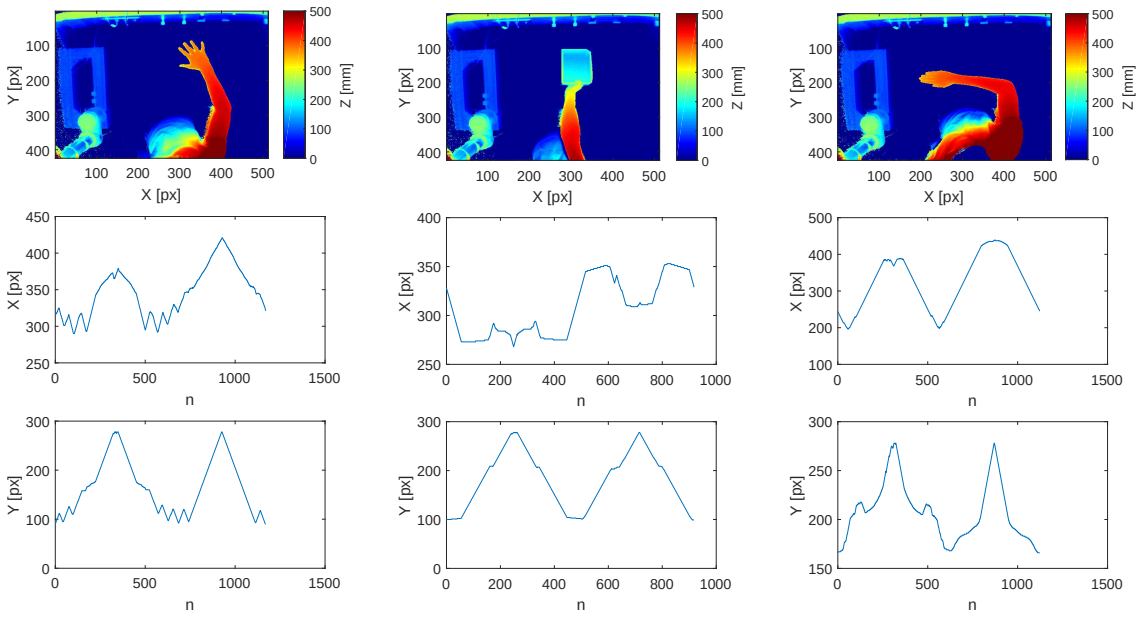
Abbildung 31: Darstellung der Vorsegmentierungsschritte in Reihenfolge der Durchführung a,b.

Der grundlegende Ansatz in der Vorsegmentierung definiert Objekte, die von dem unteren Rand in die Arbeitsumgebung laufen, als Bestandteil einer Person. Für die Extraktion der relevanten Objekte werden nachstehende Schritte ausgeführt:

1. Reduktion des Bildausschnittes auf den relevanten Arbeitsbereich.
2. Verringerung des Schwellwerts zur Entfernung des Rauschens.
3. Betrachtung von Objekten, deren konvexe Hülle eine Schnittmenge mit dem Arbeitsbereich und dem Sicherheitsbereich haben.
4. Interpretation von Objekten, deren konvexe Hülle ausreichend breit und hoch sind als Bestandteil einer Person, um das Rauschen im Randbereich zu filtern.

Über die konvexe Hülle sind die Randpunkte der Gliedmaßen bekannt und können für die Approximation des Armes verwendet werden. Für eine Approximation der Arme durch ein Objekt (Ellipse oder ein Zylinder), werden mindestens die optischen Start- und Endpunkte (Hand und Schulter) sowie die Breite der Hülle benötigt. Eine Erfassung der Punkte ist aufgrund der Vielzahl an möglichen Ausrichtungen des Armes nicht möglich, da eine Hülle beide Gliedmaßen enthalten kann. Ein weiteres Problem entsteht sobald sich ein Mensch in den Arbeitsbereich hineinlehnt und eine Hülle erzeugt, die aus den Armen und dem Oberkörper besteht. Es ist daher ein Ansatz zu finden, mit dem die relevanten Objekte in unterschiedlichen Positionen erkannt werden.

6.4.2 Ansatz über die Ortsinformation der Objektkontur



(a) Testsituation 1: offene Hand.

(b) Testsituation 2: vertikale Bewegung einer Box.

(c) Testsituation 3: Stark angewinkelter Arm.

Abbildung 32: Ortsprofile der Kontur eines menschlichen Armes für drei unterschiedliche Situationen.

Über die Kontur des relevanten Objektes sind die Randpunkte der Arme bekannt. Eine Approximation des Armes kann erfolgen, wenn folgende Punkte bekannt sind:

- Die Spitze der Hand.
- Rechter und linker Punkt des Armes nahe des Sicherheitsbereiches.
- Die Position des Ellbogens.

Werden diese Randpunkte verwendet um Ellipsen zwischen Startpunkt-Ellbogen und Ellbogen-Endpunkt zu bilden, wird angenommen, dass ein Arm hinreichend nachgebildet werden kann. Dafür erfolgt eine Trennung der Ortsprofile an den benötigten Punkten sowie eine anschließende Zuordnung in Punktmengen für zwei Armteile. Für die Gewinnung der Punkte werden die in Abbildung 32 gezeigten x- und y- Profile für drei Situationen gegenübergestellt. Die erste Situation zeigt den vertikalen Arm mit geöffneter Hand. Die Hand ist sowohl in dem x- als auch dem y-Profil durch den gezackten Bereich zu erkennen und enthält die Spitze der Hand (Minimum im y-Profil). Der rechte und linke Punkt im Sicherheitsbereich kann im y- und x-Profil durch die Maxima entnommen werden. Die aufgrund der angewinkelten Armposition benötigte Position des Ellbogens ist durch das Maximum im x-Profil zu entnehmen. Durch diese kann das Gegenstück in dem y-Profil erfasst werden und eine Einteilung erfolgen.
In der zweiten Situation ist der Arm vertikal ausgerichtet und die Person hält eine Transportbox fest. Die Spitze der Hand wird durch die Spitze der Box ersetzt und kann ebenfalls dem Minimum des y-Profils entnommen werden. Der rechte und linke Punkt des Armes nahe dem Sicherheitsbereich entsprechen in dieser Situation nur den Maxima im y-Profil und sind im x-Profil durch die Breite der Box nicht eindeutig identifizierbar. Ein Merkmal für den Ellbogen kann in keiner der beiden Profile gefunden werden.

Die dritte Situation beschreibt einen stark angewinkelten Arm. Entgegen der vorherigen Situationen ist die Spitze der Hand nicht in dem Minimum des y-Profils, sondern in dem Minimum des x-Profils zu entnehmen. Der Ellbogen äußert sich nicht durch ein eindeutiges Merkmal, während der rechte und linke Punkt im Sicherheitsbereich durch die Maxima des y-Profils definiert ist.
Die Gegenüberstellung zeigt, dass die notwendigen Merkmale zur Approximation eines Armes durch zwei Teile nicht in jeder Situation eindeutig sind. Zusätzlich handelt es sich nicht um Situation-invariante Merkmale wodurch eine Zuordnung der Minima und Maxima in den Profilen deutlich erschwert ist. Die Betrachtung der Profile in den drei Situationen zeigt außerdem eine variierende Breite und Höhe der Merkmalsstruktur in Form von Dreiecken aufgrund unterschiedlicher Distanzen zwischen Sensor und Arm. Der Ansatz über die Ortsinformation führt daher nicht zum gewünschten Erfolg und wird an dieser Stelle verworfen. Eine Einteilung der Punkte kann hingegen durch ein Clustern realisiert werden.

6.4.3 Ansatz über Clustern mit dem K-Means-Algorithmus

Die Zuordnung der Randpunkte der Hülle zu gemeinsamen Ortsbereichen (Clustern) bietet die Möglichkeit, Segmente eines Armes zu approximieren. Ein einfacher und bekannter Algorithmus für das Clustern ist der K-Means-Algorithmus, der auf der Minimierung kleinster Fehlerquadrate basiert und wie folgt lautet [HNCM05, Seite:288]:

$$\min f\left(\mathcal{M}, Z\right) = \sum_{i=1}^{M} \sum_{j=1}^{N} z_{ij} \cdot \left\| x_j - m_i \right\|^2$$

Dabei erfolgt das Clustern von $x_j = (x_{j1}, x_{j2}, ...x_{jd})$ Punkten, deren Index von $j = 1, ...N$ läuft in eine Anzahl von M Clustern. Für z gilt $z_{ij} \in \{0, 1\}$ und beträgt 1, wenn der Punkt x_j dem Cluster i zugeordnet ist, andernfalls 0. Der Schwerpunkt der Cluster m berechnet sich durch den Mittelwert nach [HNCM05, Seite:288]:

$$m_i = \frac{\sum\limits_{j=1}^{N} z_{ij} x_j}{\sum\limits_{j=1}^{N} z_{ij}}$$

$\left\| x_j - m_i \right\|^2$ beschreibt dabei das Quadrat der euklidischen Distanz zwischen einem Clusterpunkt und dem Schwerpunkt des Clusters, dessen Betrag minimiert wird. Die Problemstellung kann vereinfacht definiert werden durch den Ausdruck [HNCM05, Seite:289]

$$P_M^* = P_M \overset{min}{\in} \mathcal{P}_{\mathcal{M}} \sum_{i=1}^{M} \sigma^2(C_i)$$

mit der Summe der quadrierten euklidischen Distanz [HNCM05, Seite:289] :

$$\sigma^2(C_i) = \sum_{j | x_j \in C_i} \left\| x_j - m_i \right\|^2$$

Dabei ist $\mathcal{P}_{\mathcal{M}}$ die Potenzmenge bestehend aus allen möglichen Teilmengen, M die Anzahl dieser Teilmengen, P_M^* die Menge mit der geringsten Summe der quadrierten euklidischen Distanz und somit das Optimum das gesucht wird sowie C_i der i-te Cluster. Der Algorithmus findet somit Cluster in einer Punktwolke, zu deren Schwerpunkt die Distanz der einzelnen Punkte so klein wie möglich ist. In einem C++ Programm wird er iterativ aufgerufen und mit der Anzahl der Aufrufe genauer. In jedem Schritt wird dabei der Schwerpunkt der Cluster verschoben und die euklidische Distanz zu den Punkten in diesem Cluster verringert. Es werden folgende Parameter benötigt, die sinnvoll gewählt werden müssen:

1. Die Anzahl der Cluster, in die eine Punktwolke eingeteilt werden soll.

2. Die Startpunkte (Schwerpunkte) der Cluster.

3. Die Anzahl der Iterationen, um ein hinreichend genaues Ergebnis zu erhalten.

Ein weiterer Algorithmus für die Einteilung von Datenpunkten in Cluster ist der Expectation-Maximization-Algorithmus, der dem K-Means Algorithmus gegenübergestellt werden soll.

6.4.4 Ansatz über Clustern mit dem Expectation-Maximization-Algorithmus

Der Expectation-Maximization-Algorithmus (EM-Algorithmus) kann verwendet werden um für eine Reihe von Messungen, die falsche oder fehlende Werte enthält, die Wahrscheinlichkeitsfunktionen zu bestimmen und gleichzeitig die falschen oder fehlenden Werte zu verbessern/schätzen. Ein weites Anwendungsgebiet ist das Clustern, das die Unterteilung von Daten in Bereiche (Cluster) durchführt und für die Aufgabenstellung in diesem Kapitel untersucht werden soll. Um die Funktion zu erklären wird angenommen, dass die Punkte der Arme die geclustert werden sollen, normalverteilt sind.
Eine Normalverteilung in der p-ten Dimension setzt sich aus der Kovarianzmatrix

$$\sum = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \cdots & \sigma_{pp} \end{pmatrix}$$

und dem Erwartungswertvektor

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{pmatrix}$$

zusammen. Die Kovarianzmatrix beschreibt dabei das Maß, mit dem zwei Zufallsvariablen zusammenhängen. D.h σ_{32} wie sehr die Zufallsvariablen drei und zwei zusammenhängend sind. Besteht kein Zusammenhang, so gilt $\sigma = 0$. Der Erwartungsvektor enthält für jede Zufallsvariable den durchschnittlich auftretenden Wert. Die Dichtefunktion für eine multidimensionale Normalverteilung ist definiert durch:

$$f_X(x) = \frac{1}{\sqrt{(2\pi)^p \det(\sum)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}} \sum^{-1} (\mathbf{x}-\boldsymbol{\mu})\right) \tag{18}$$

Die Dichtefunktion gibt an wie wahrscheinlich es ist, unter Verwendung von $\sum$ und μ, dass ein Zufallsexperiment mit dem Ergebnis der Messung $\mathbf{x}$ endet. Anhand der reinen Messdaten lässt sich die Dichtefunktion nicht bestimmen, da hierfür mehrere Lösungen existieren. Aus diesem Grund wird die Dichtefunktion gesucht, die mit der höchsten Wahrscheinlichkeit die Messwerte erzeugt hat. Angenommen es werden N Messungen erfasst und die Ergebnisse aller möglichen Dichtefunktionen miteinander multipliziert, dann resultiert mit Gleichung 18:

$$\mathcal{L}(\theta \mid \chi) = \left(\frac{1}{\sqrt{(2\pi)^p \det(\sum)}}\right)^N \prod_{i=1}^{N} \exp\left(-\frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu})^{\mathrm{T}} \sum^{-1} (\mathbf{x}_i-\boldsymbol{\mu})\right) \tag{19}$$

Sie enthält die sogenannte „Likelihood“–Funktion und multipliziert für alle N Messungen mit dem Messwert $\mathbf{x}_i$ die Wahrscheinlichkeit mit der eine Zufallsmessung in dem Ergebnis $\mathbf{x}_i$ endet. Durch Variieren der Parameter $\sum$ und μ kann die Dichtefunktion gefunden werden, deren Produkt am höchsten ist und somit die Dichtefunktion beschreibt, die der Messung mit den Werten $\mathbf{x}$ am wahrscheinlichsten zu Grunde liegt. Die Dichtefunktion wird durch das Maximum der Likelihood–Funktion (vgl. Formel 19) definiert und kann durch die Ableitung, deren Ergebnis am Maximum Null ist, der Gleichung 19 ermittelt werden.

Im Weiteren definiert die Wahrscheinlichkeitstheorie über die Produktregel in den beiden Formen (In Anlehnung an [RN12, Seite: 585])

$$P(a \wedge b) = P(a|b)P(b) = P(b|a)P(a) \tag{20}$$

wie wahrscheinlich es ist, dass a und b gemeinsam auftreten. Wenn a für einen Datenpunkt des Armes steht und b ein Cluster ist, definiert $P(a \wedge b)$ die Wahrscheinlichkeit dass das Cluster b existiert ($P(b)$) und die Wahrscheinlichkeit dass der Punkt a diesem angehört ($P(a|b)$). Gleichung 20 kann umgestellt werden zu [RN12, Gleichung: 13.12]:

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)}$$

und wird als Bayessche-Regel bezeichnet. Für die Lösung wird die A-priori-Wahrscheinlichkeit $P(a)$ benötigt, also das Wissen darüber, wie wahrscheinlich es ist, dass der Datenpunkt a auftritt. Alternativ kann für jeden möglichen Cluster b die A-posteriori- Wahrscheinlichkeit (Er tritt auf b , oder nicht $\neg b$) bestimmt werden, daraus folgt (In Anlehnung an [RN12, Seite: 586])

$$\mathbf{P}(B|a) = \alpha \langle P(a|b)P(b), P(a|\neg b)P(\neg b)\rangle$$

Existieren mehrere Attribute in A (x-, y-, z- Position im 3D-Raum), so kann diese Gleichung für einen Cluster B umgeschrieben werden zu [RN12, Gleichung: 13.15] :

$$\mathbf{P}(B|A) = \alpha \mathbf{P}(A|B)\mathbf{P}(B) \tag{21}$$

α beschreibt die Normalisierungskonstante, damit die Summe der Einträge in $\mathbf{P}(B|A)$ eins beträgt. Das Clustern verfolgt das Ziel, N Punkte die gemessen werden in i Cluster C zu unterteilen. Daher notiert nachfolgend $\mathbf{A}$ alle Punkte eines Armes und $\mathbf{a}$ einen Punkt mit den Werten x-, y- und z im Bild. Im Initialisierungsschritt wird davon ausgegangen, dass die Zuordnung der Punkte in die entsprechenden Cluster bekannt sind. Für die erste Annahme bietet sich die Anwendung des K-Means-Algorithmus an, wodurch eine erste Einteilung geschätzt wird. Durch die „Likelihood"-Funktion kann die Dichtefunktion und anschließend das Gewicht jedes Clusters bestimmt werden. Das Gewicht w_i eines Cluster ist definiert durch die Anzahl der Punkte n_i in dem Cluster C_i, dividiert durch die Gesamtanzahl der Messungen N. Im nächsten Schritt, dem sogenannten E-Schritt, wird für jeden Punkt die Wahrscheinlichkeit bestimmt mit der er einem Cluster angehört [RN12, Seite: 946] :

$$p_{ij} = P(C = i|\mathbf{a}_j)$$

Daraus resultiert wie wahrscheinlich es ist, wenn Punkt $\mathbf{a}_j$ gemessen wird, dass dieser dem Cluster C_i angehört. Nach Gleichung 21 resultiert [RN12, Seite: 586] :

$$p_{ij} = \alpha P(\mathbf{a}_j|C = i)P(C = i)$$

Dabei beschreibt $P(C = i)$ den Gewichtungsparameter der i-ten Gaußschen Dichtefunktion und $P(\mathbf{a}_j|C = i)$ die Wahrscheinlichkeit von $\mathbf{a}_j$ der i-ten Gaußschen Dichtefunktion. Im sogenannten M-Schritt werden anschließend die Parameter der Dichtefunktion jedes Clusters wie folgt neu bestimmt [RN12, Seite: 586] :

$$\mu_i \leftarrow \frac{1}{n_i}\sum_j p_{ij}\mathbf{a}_j$$

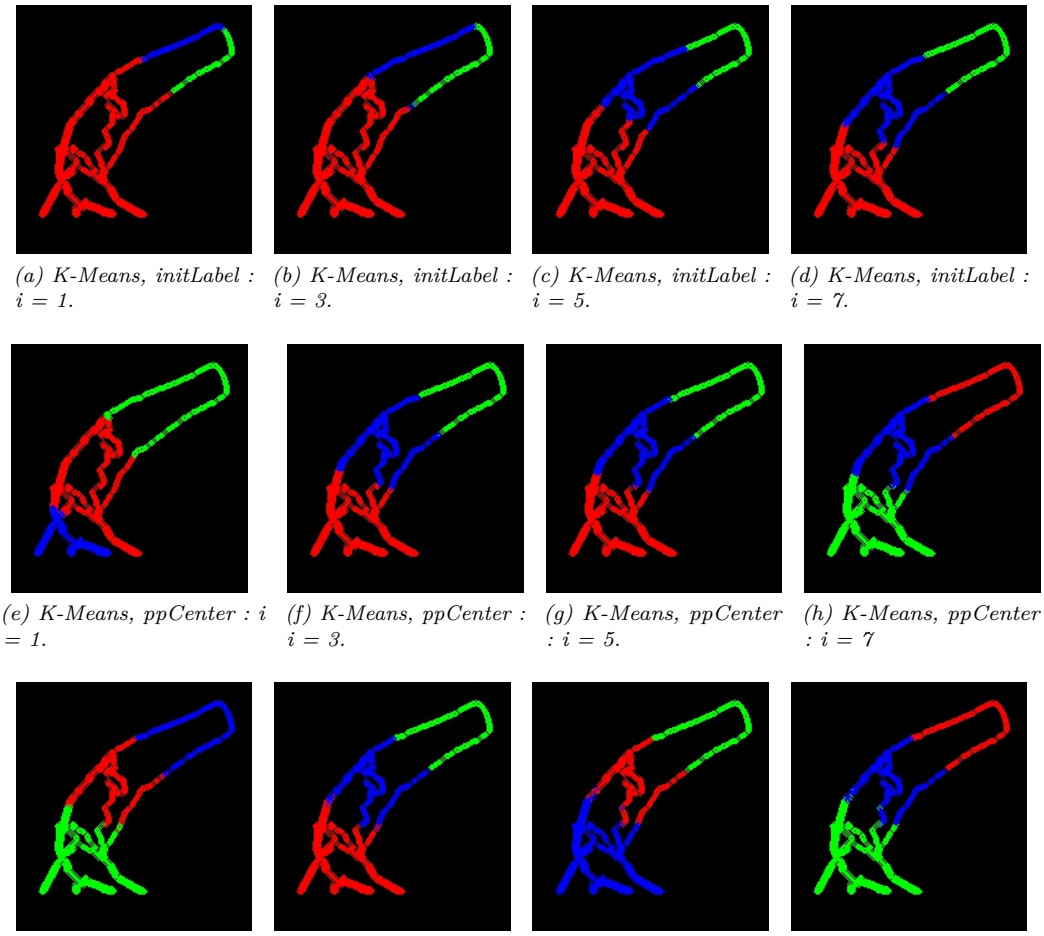
$$\sum_i \leftarrow \frac{1}{n_i}\sum_j p_{ij}(\mathbf{a}_j - \boldsymbol{\mu})(\mathbf{a}_j - \boldsymbol{\mu})^T$$

$$w_i \leftarrow \frac{n_i}{N}$$

Die Wahrscheinlichkeit p_{ij} dient dabei als „Gewichtungsfaktor“ und ist niedrig sofern die Werte x-, y- und z des Punktes $\mathbf{a}_i$ einen geringen Erwartungswert haben und im äußeren Bereich der Dichteverteilung des i-ten Clusters sind. Ist der Cluster weit entfernt beträgt $p_{ij} = 0$ und bleibt in dem M-Schritt für den i-ten Cluster unbeachtet. Werte, die in dem i-ten Cluster einen hohen Erwartungswert besitzen werden daher stärker beachtet, wodurch sich die Verteilung des Clusters in eine bestimmte „Richtung“ bewegt, vorausgesetzt der E- und der M- Schritt werden iterativ aufgerufen. Nach Erreichen der gewünschten Iterrationsanzahl wird jeder Punkt dem Cluster zugeordnet, zu dem er den höchsten Erwartungswert besitzt. Für diesen Ansatz werden somit folgende Parameter benötigt:

1. Die Anzahl der Cluster.
2. Die Initialisierungscluster, die z.B durch den K-Means bestimmt werden können.
3. Die Anzahl der Iterationen oder die Abbruchbedingung (Stärke der Änderung gegenüber dem vorherigen Schritt).

6.4.5 Gegenüberstellung und Validierung



(a) K-Means, initLabel : i = 1. *(b) K-Means, initLabel : i = 3.* *(c) K-Means, initLabel : i = 5.* *(d) K-Means, initLabel : i = 7.*

(e) K-Means, ppCenter : i = 1. *(f) K-Means, ppCenter : i = 3.* *(g) K-Means, ppCenter : i = 5.* *(h) K-Means, ppCenter : i = 7*

(i) EM : i = 1. *(j) EM : i = 3* *(k) EM : i = 5* *(l) EM : i = 7.*

Abbildung 33: Farbliche Gegenüberstellung der Clusterschritte i des K-Means (Mit den Initialisierungensvariablen PP_CENTERS und INITIAL_LABELS) - und des EM- Algorithmus.

Abbildung 33 zeigt die Ergebnisse der Algorithmen bei identischer Informationsgrundlage und unterschiedlichen Iterationen. Für den K-Means werden zwei Initialisierungsarten gegenübergestellt, die Initialisierung INITIAL_LABELS ohne Angabe der initialisierten Labels und die Initialisierung über das Verfahren nach [AV07]. Im ersten Ansatz ist zu erkennen, dass sich mit steigender Anzahl der Iterationen der Algorithmus der optimalen Verteilung nähert und der Arm in drei Bereiche eingeteilt wird (Abbildung 33 Abbildung a-d). Unter Verwendung des Initialisierungsansatzes nach [AV07] wird diese Einteilung bereits nach drei Iterationen erreicht (Abbildung 33 Abbildung e-h). Es ist zu beachten, dass die selben Cluster unterschiedliche Farben besitzen. Dies lässt sich darin begründen, dass der Algorithmus sieben Mal mit entsprechender Iterrationsanzahl ausgeführt wird, da eine Extraktion der Cluster zwischen den einzelnen Schritten nicht möglich ist. Die Initialisierung variiert trotz gleicher Information in diesem Ansatz und es entstehen unterschiedliche Zuordnungen zwischen Cluster und Farbe für jede Durchführung. Dies gilt ebenfalls für den EM-Algorithmus. Die Gegenüberstellung zeigt deutlich, wie relevant die Initialisierung der Cluster ist, weshalb der zweite Ansatz (Abbildung 33 Abbildung e-h) nach drei Iterationen hinreichend gut ist um den Arm in drei Cluster zu teilen. Wird der EM-Algorithmus betrachtet ist zu erkennen, dass die Einteilung bereits nach der ersten Ausführung (E- und M-Schritt) erreicht ist (Abbildung 33 Abbildung i-l). Die Initialisierung der Cluster basiert dabei ebenfalls auf dem K-Means-Algorithmus. Da keine Informationen über die Implementation des K-Means in dem EM-Algorithmus vorliegen, kann nicht ausgeschlossen werden, dass dieses Ergebnis auf den K-Means im Initialisierungsschritt mit mehreren Iterationen zurückzuführen ist. Die Gegenüberstellung zeigt, dass der K-Means mit drei Iterationen und dem Initialisierungsschritt nach [AV07] sowie der EM-Algorithmus mit einem Iterationsschritt für das Clustern hinreichende Ergebnisse erzielt. Beide Algorithmen werden daher durch die im QT-Framework bereits existierenden Algorithmen näher betrachtet. Dabei wird die Laufzeit beider Algorithmen für 72 Aufnahmen erfasst und der Mittelwert gebildet. Da beide Algorithmen gleichzeitig auf Basis der identischen Punkte ausgeführt werden, wird die Laufzeit des gesamten Programms stark reduziert, sodass ausreichend Zeit für verschiedene Handpositionen vorhanden ist. Während der Messung findet eine Variation verschiedener Positionen des Armes in Höhe und Ausrichtung statt, wodurch eine variierende Anzahl an Messpunkten an die Algorithmen übergeben wird. Während der gesamten Messung wird ausschließlich ein Arm verwendet und der Arbeitsplatz ist leer geräumt. Zusätzlich werden 450 Bilder aufgezeichnet, aus denen das Ergebnis des Clustern zu entnehmen ist um sicherzustellen, dass die Gliedmaßen vollständig umhüllt sind. Beide Algorithmen clustern in drei Bereiche, die anschließend durch ein approximiertes Rechteck visualisiert werden. Als Startwert für den K-Means erfolgt eine Approximierung mit einer einmaligen Initialisierung nach [AV07] und für den EM-Algorithmus wird die Initialverteilung durch den K-Means definiert. Auf die Tiefenwerte wird eine Schwellwertgrenze von $z > 30\,\text{mm}$ für die Rauschunterdrückung angewendet. Es werden nur konvexe Hüllen als Bestandteil eines relevanten Objektes verwendet, wenn die Schnittmenge des darauf approximierten Rechtecks im Sicherheitsstreifen mindestens 100 px, im Arbeitsbereich mindestens 250 px und die Gesamtfläche mindestens 500 px beträgt.
Die Laufzeitmessung ergab für den K-Means-Algorithmus $t_{K-Means} = 6,40\,\text{ms}$ und für den EM-Algorithmus $t_{EM} = 26,99\,\text{ms}$, weshalb der K-Means-Ansatz unter den gegebenen Umständen ca. 4,2x schneller ist. Die Betrachtung der 450 Bildaufnahmen ergibt, dass beide Algorithmen in 50 der 450 erfassten Situationen zu einem Clustern kam bei dem der Roboter bei entsprechender Fahrtrichtung und Geschwindigkeit mit dem Arm kollidieren könnte. Die Ursache ist dabei nicht in den Algorithmen zu finden, sondern in der Tatsache, dass in jeder dieser Aufnahme das Arbeitsfeld verlassen wurde. Punkte, die nicht Bestandteil der Arbeitsfläche sind werden ignoriert. Weitere dieser Situationen entstehen, wenn die Person oder deren Gliedmaßen zu nah am Sensor sind, da sie in den Schwellwertbereich gelangen. Die Reduzierung der Schwellwertgrenze auf 0 mm erzeugt ein Rauschen im unteren Bildbereich, wodurch dieses fehlerhaft als Teil von Gliedmaßen erkannt wird. Da nach weiteren 30 mm in Richtung Kamera die Tiefendaten nicht im Erfassungsbereich des Tiefensensors liegen, ist keine deutliche Verbesserung durch Entfernen der Schwellwertgrenze zu erwarten. Somit fassen alle Cluster die Punkte einer Person hinreichend genau zusammen, wodurch die Schnittmenge zwischen Cluster und Roboter als Stopp-Kriterium verwendet werden kann. Die Behandlung der Grenzfälle wird in der Durchführung beschrieben.

6.5 Klassifizierung und Kollisionserkennung

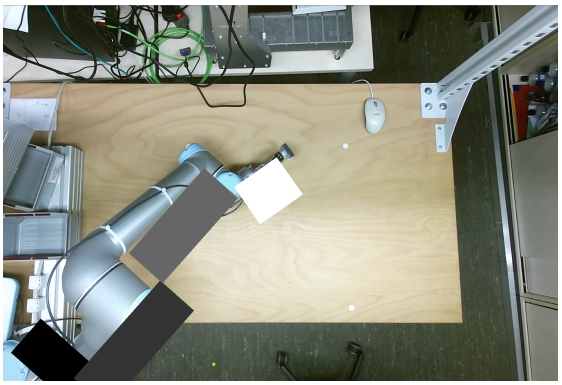
Für die Klassifizierung der Konturen in die Klasse „Roboter“ und „Mensch“ stehen mehrere Ansätze zur Verfügung. Die einzelnen Punkte der Kontur können betrachtet werden und durch die Transformation in das Weltkoordinatensystem mit dem Roboter abgeglichen werden. Dieser Ansatz basiert auf den Tiefenwert-Informationen der Kinect und ist am Rand der Kontur mit Unsicherheiten behaftet. Eine mögliche Lösung ist, die Tiefenwerte eines innenliegenden Punktes zu verwenden um einen besseren Messpunkt für den Tiefenwert zu erhalten. Der Vergleich mit dem Robotermodell benötigt eine Transformationsvorschrift, die zusätzlich die z-Achse beachtet. Diese kann gewonnen werden, wenn die Boxen während des Kalibrierungsprozesses in unterschiedlichen Höhen positioniert werden. Aufgrund der Tatsache, dass der Roboter für die Erfassung der Box, diese im Bild loslassen muss (ansonsten passen die Hu-Momente nicht), wird ein entsprechendes Kalibrierungswerkzeug benötigt, das die Box in der Höhe fixiert. Dieser Ansatz ist in dem Zeitrahmen dieser Thesis nicht umsetzbar und wird daher verworfen.
Stattdessen soll untersucht werden, ob die bereits geclusterten Konturpunkte zu zweidimensionalen Rechtecken approximiert werden können und eine Klassifizierung im Weltkoordinantensystem erlauben. Es wird angenommen, dass dieser Ansatz mit Einschränkungen verbunden ist, gleichzeitig aber eine schnelle Umsetzung erlaubt, da die notwendigen Informationen bereits vorhanden sind. Zusätzlich sind weniger Prozessschritte notwendig, um den Ansatz umzusetzen, wodurch Systemressourcen gespart werden können.
Für die Klassifizierung wird im ersten Schritt das Modell des Roboters benötigt.

6.5.1 Robotermodell

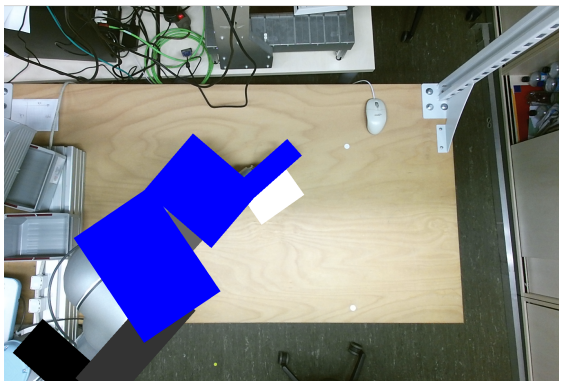
Ein Abgleich der Cluster und der Position des Roboters soll im zweidimensionalen Raum des gemeinsamen Weltkoordinatensystems erfolgen. Das Robotermodell wird mit den händisch erfassten Längen der Robotergelenke sowie dem Wissen über die Gelenkstellung modelliert. Ein Gelenk besteht aus einem Richtungsvektor und einem Breitenvektor, die gemeinsam ein zweidimensionales Rechteck in OpenCV erstellen können. Für die Klassifizierung werden diese Punkte in das Weltkoordinatensystem transformiert und dort mit den erkannten Gliedmaßen abgeglichen. Die Modellierung des zweidimensionalen Modells ist der Abbildung 34 zu entnehmen. Zu erkennen sind alle Robotergelenke von der Basis (schwarz) bis zu dem Tool (weiß). Sobald der Roboter eine Kiste greift, wird das Rechteck des Tools um die Länge einer großen Box vergrößert damit die Box als Bestandteil des Roboters interpretiert wird.

6.5.2 Klassifizierung der Cluster

Die Klassifizierung der Cluster in Bestandteile eines Menschens und eines Roboters erfolgt über die Schnittmenge mit dem Robotermodell. Dazu werden die Richtungs- und Breitenvektoren der Cluster sowie die des Roboters in das Weltkoordinatensystem transformiert und dort ein entsprechendes Rechteck gebildet. Im Anschluss werden alle durch das Clustern gewonnen Rechtecke mit den Rechtecken des Roboters auf eine Schnittmenge abgeglichen. Um Ressourcen zu sparen wird die Frage nach der Schnittmenge binär mit „true“ oder „false“ beantwortet ohne das Maß der Menge genau zu definieren. Dieser Ansatz spart Rechenzeit und ist gleichzeitig ungenauer. Die Ungenauigkeit resultiert aus der Transformation zwischen dem Tiefen- und dem Farbsensor durch den Coordinate-Mapper. Da die Tiefenwerte auf der die Transformation zwischen den beiden Bilder basiert, sich auf den Mittelpunkt des Clusters beziehen oder aus Clustern stammen, die nur Konturpunkte am Rand eines Armes enthalten, weisen die zweidimensionalen Elemente aus dem Clustern ein sehr unruhiges Verhalten in der Position auf. Dieses Verhalten wird durch die vom Initialisierungsschritt des K-Means-Algorithmus abhängigen und variierenden Ergebnisse verstärkt. Eine Überlagerung der Roboter-Gelenke und der geclusterten Arme des Roboters ist der Abbildung 34b zu entnehmen.



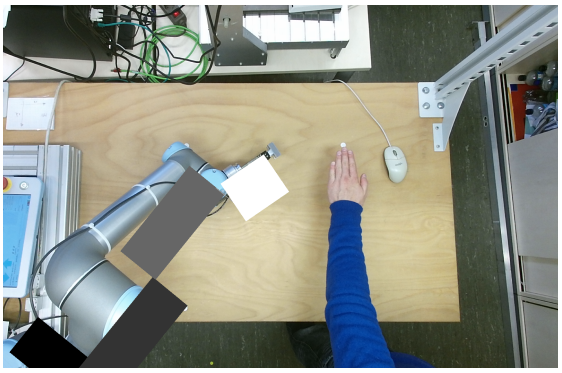
(a) Modell des Roboters im Weltkoordinatensystem (weiß - grau).



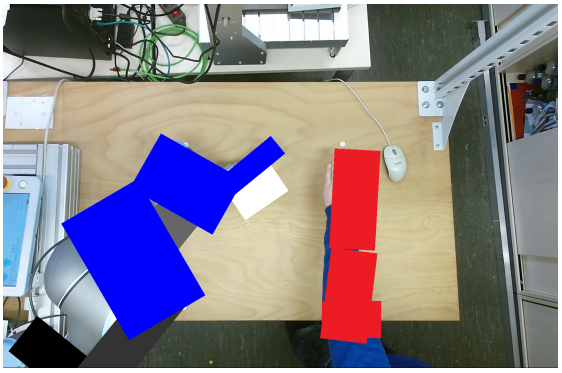
(b) Modell des Roboters (weiß - grau) und die resultierenden Cluster (blau) im Weltkoordinatensystem.

Abbildung 34: Das im Farbbild eingezeichnete Modell des Roboters (Bezogen auf das Weltkoordinatensystem) sowie die durch den Roboter verursachten Cluster (1 px≙1 mm für die eingezeichneten Rechtecke).

Die Abbildung 34 zeigt das Modell des Roboters sowie das Ergebnis des K-Means-Algorithmus auf Basis der durch die Hu-Momente gefilterten Konturen. Zu erkennen ist, dass die Cluster nicht gleichmäßig groß sind und sich unterschiedlich verteilen. Dies liegt daran, dass das Clustern auf den Datenpunkten der Hülle basiert und somit nur die Einteilung der Kontur erfolgt. In dem gezeigten Beispiel wurde für die Stabilität bereits der Punkt am zentralen Moment der Kontur sowie dem Mittelpunkt im Sicherheitsbereich des Arbeitsbereiches der zu clusternden Menge hinzugefügt. Zusätzlich ist zu erkennen, dass alle drei Cluster eine Schnittmenge mit dem Modell aufweisen und dadurch Rückschlüsse auf den Roboter geben. Im nächsten Schritt erfolgt die Betrachtung einer Situation, in der sich ein Mensch im Arbeitsbereich aufhält.



(a) Modell des Roboters und ein menschlicher Arm im Weltkoordinatensystem (weiß-grau).



(b) Modell des Roboters (weiß - grau) und die daraus resultierenden Cluster (blau) zusammen mit den geclusterten Rechtecken der Kontur eines menschlichen Arms (rot) im Weltkoordinantensystem.

Abbildung 35: Das im Farbbild eingezeichnete Modell des Roboters, die durch den Roboter verursachten Cluster und die durch einen Menschen verursachten Cluster (1 px≙1 mm für die eingezeichneten Rechtecke).

Die Abbildung 35 zeigt den Roboter und eine menschliche Hand im Farbbild. Die Rechtecke

basieren dabei ebenfalls auf den Clustern des K-Means-Algorithmus und wurden nachträglich farblich unterschieden. Gegenüber der Abbildung 34b ist für die Cluster des Roboters (blau) zu erkennen, dass die Größe und Ortsinformation der Rechtecke deutlich variiert. Folgende Aussagen lassen sich aus der Aufnahme ableiten:

- Die Bildung der Rechtecke auf Grundlage der geclusterten Kontur verfälscht die Größe und Form des Roboters stark.
- Die Ortsinformation und Ausrichtung der Rechtecke auf Grundlage der Cluster variiert stark.
- Alle Rechtecke des Roboters weisen eine Schnittmenge mit dem Modell des Roboters auf.

Auf Grundlage dieser Aussagen ist zu erwarten, dass eine Klassifizierung der Cluster, sofern ein Clustern in drei Segmente stattfindet, folgendermaßen erfolgen kann:

Mensch Die Cluster und die Kontur werden als Mensch klassifiziert, wenn von drei zusammenhängenden Clustern ein oder kein Cluster eine Schnittmenge mit dem Robotermodell besitzt.

Roboter Die Cluster und die Kontur werden als Roboter klassifiziert, wenn von drei zusammenhängenden Clustern mindestens zwei eine Schnittmenge mit dem Robotermodell aufweisen.

Durch die Klassifizierung der Cluster besteht im Anschluss die Möglichkeit diese auf eine Kollision untereinander zu prüfen.

6.5.3 Kollisionserkennung

Das Merkmal einer Kollision ist in der Schnittmenge der definierten Cluster von Roboter und Mensch zu finden. Die vorherige Betrachtung der Cluster hat gezeigt, dass die daraus gewonnenen Rechtecke in der Position und Ausrichtung stark variieren und die Größe des Roboters nicht realistisch repräsentieren. Eine Kollisionserkennung durch die Schnittmenge der Rechtecke entfällt daher. Eine Alternative bietet sich durch die zuvor gewonnene Kontur des Roboters und des menschlichen Armes, da diese die Form und Ausrichtung sehr gut widerspiegelt. Die Klassifizierung der Cluster wird daher verwendet, um die Zuordnung zwischen dem Menschen und dem Roboter zu übernehmen. Damit eine Kollision frühzeitig erkannt und der Roboter gestoppt wird, werden beide Konturen vergrößert und miteinander auf eine Schnittmenge geprüft.
Dazu wird bereits in dem parallelisierten Prozess des Clusterns die ausgefüllte Kontur in eine einkanalige Matrix gezeichnet, in der alle Pixel der Kontur den Wert 255 aufweisen. Durch Variieren der Linienbreite mit der die Kontur gezeichnet wird, kann die Kontur in der Größe verändert werden. Die so erstellten Bilder für die erkannten Konturen können nach der Klassifizierung dem Roboter und dem Menschen zugeordnet werden. Da mehrere Arme oder Teile eines Menschen in dem Arbeitsbereich arbeiten können, werden gleich klassifizierte Matrizen zusammengeführt, damit im Anschluss zwei Matrizen vorhanden sind. Die Matrix M_1 enthält die Kontur des Roboters und die Matrix M_2 alle Konturen, die als Mensch klassifiziert wurden. Beide Matrizen werden anschließend wie folgt zu der Matrix M_C zusammengefügt:

$$M_C = M_1 \cdot \alpha + M_2 \cdot \beta \tag{22}$$

Gehen beide Matrizen mit dem Faktor $\alpha = \beta = 0,5$ ein, weisen anschließend nur die Pixel in M_C den Wert 255 auf, die sowohl in der Kontur des Menschen als auch in der Kontur des Roboters den Wert 255 aufweisen und damit eine Kollision beschreiben. Das Verfahren wird durch Abbildung 36 grafisch erläutert.

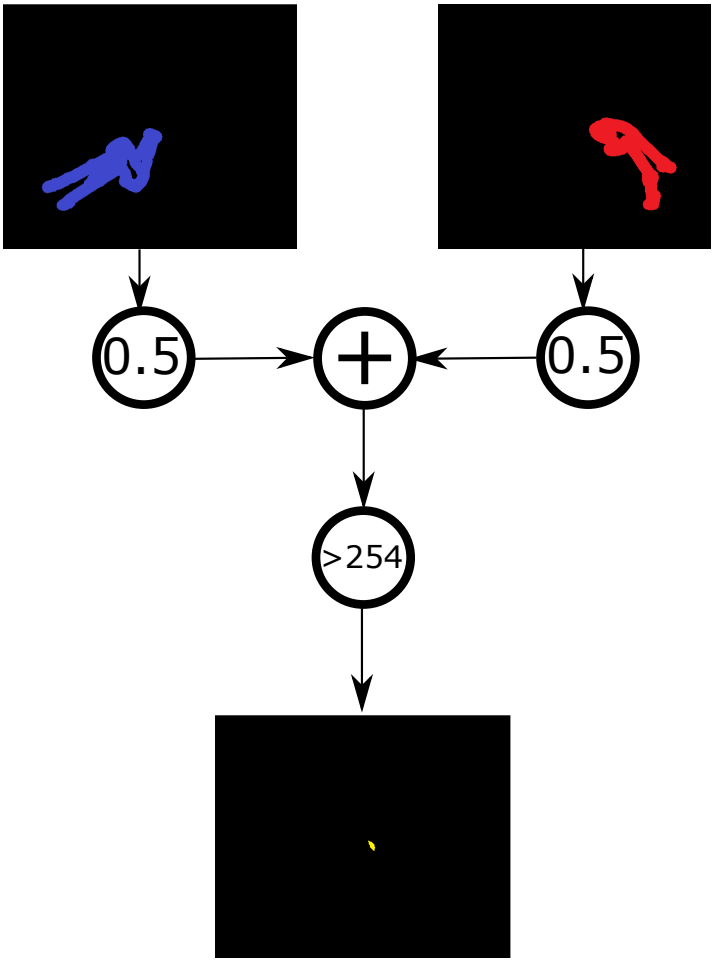


Abbildung 36: Darstellung der klassifizierten und vergrößerten Konturen des Menschen (rechts/rot), des Roboters (links/blau) sowie der Überlagerung beider Konturen (unten/gelb) durch eine Kollision.

Abbildung 36 zeigt das Ergebnis des Ansatzes nach Gleichung 22 während eine Kollision zwischen Roboter und Mensch existiert. Die Kollision konnte in dem gezeigten Beispiel erfolgreich erkannt werden, weshalb der Ansatz durch diverse Situationen validiert wird.

6.5.4 Validierung

Für die Validierung der Klassifizierung werden unterschiedliche Situationen analysiert. Aufgrund der Anzahl an möglichen Situationen kann diese Arbeit nur einen Bruchteil davon behandeln. Aus diesem Grund werden Situationen betrachtet, an denen der Ansatz an seine Grenze gebracht wird. Zu diesen Situationen gehören:

1. Das Einfahren des Roboters in den Arbeitsbereich. In diesen Momenten wird die Kontur des Roboters durch den Sicherheitsbereich unterbrochen und so in zwei Konturen unterteilt. Wird die zweite Kontur als Mensch klassifiziert, wird dies als eine dauerhafte Kollision interpretiert und der Roboter lässt sich nicht mehr bewegen.

2. Der Roboter bewegt sich über den Boxen, weshalb die Kontur der Box Bestandteil der Roboterkontur wird. Zu erwarten ist, dass die Boxen die Verbindung zwischen der Kontur des Roboters und des Menschen herstellen können.

3. Der Roboter ist zu schnell um rechtzeitig zu stoppen.

Bereits im Voraus können die folgenden Situationen ausgeschlossen werden:

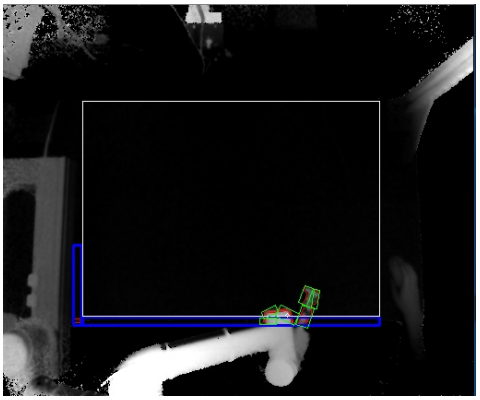
1. Der Roboter oder der Mensch verlassen den Arbeitsbereich oder arbeiten nicht durch den Sicherheitsbereich hindurch.

2. Der Roboter oder der Mensch treten aus dem Erfassungsbereich der Kamera aus, indem der Abstand zu der Kinect zu gering ist, um Tiefenwerte zu erfassen.

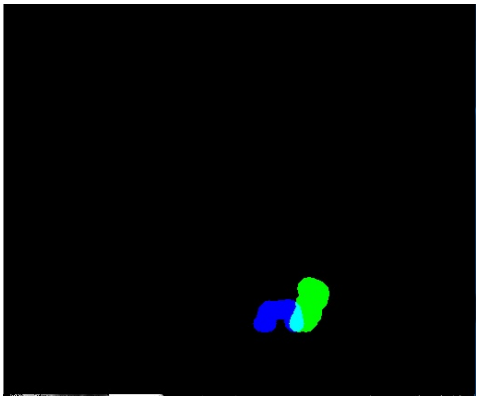
Die zu untersuchenden Situationen werden in diversen Varianten simuliert und die Ergebnise in den nachfolgenden Teilen erläutert.

Einfahren in den Arbeitsbereich

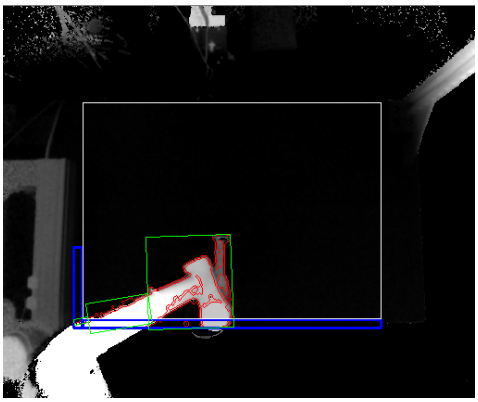
Die Trennung der Konturen führt zu der Annahme, dass diese eine Kollision verursachen können, obwohl diese nicht existiert. Um diesen Vorgang zu validieren, fährt der Roboter in drei unterschiedlichen Posen mit einer Geschwindigkeit von $v = 0,1\,\frac{\mathrm{m}}{\mathrm{s}}$ in den Arbeitsbereich hinein. Posen, Konturen und Cluster sind der Abbildung 37 zu entnehmen.



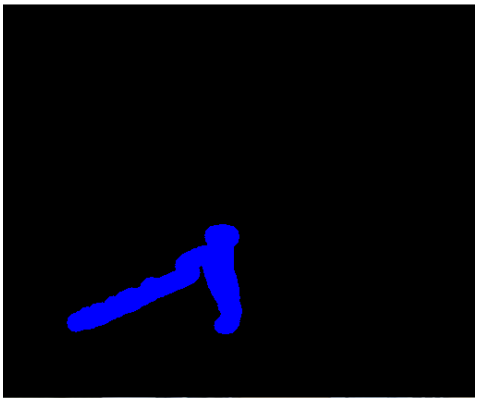
(a) Situtaion 1: Darstellung der Cluster (grün) und Konturen (rot).



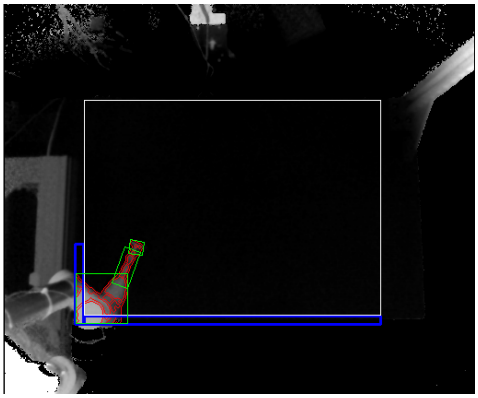
(b) Situation 1: Die Kollisionsmap mit eingezeichneter Kollision.



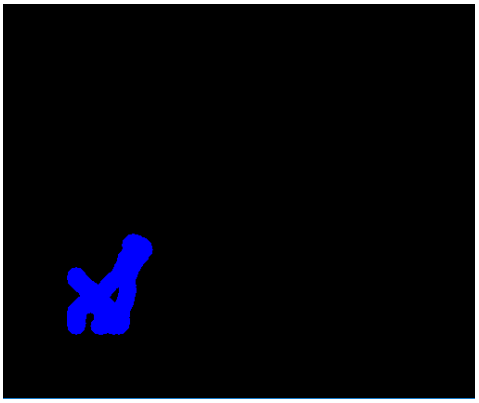
(c) Situation 2: Darstellung der Cluster (grün) und Konturen (rot).



(d) Situation 2: Die Kollisionsmap mit korrekt klassifiziertem Roboter.



(e) Situation 3: Darstellung der Cluster (grün) und Konturen (rot).



(f) Situation 3: Die Kollisionsmap mit korrekt klassifiziertem Roboter.

Abbildung 37: Gegenüberstellung verschiedener Situationen, in denen der Roboter in den Arbeitsbereich fährt.

Die Gegenüberstellung der drei untersuchten Situationen in der Abbildung 37 zeigt, dass die Annahme eintritt und den Roboter durch eine permanente Kollision blockiert wird. Dieses Verhalten ist in der ersten Testsituation aus der Abbildung 37a zu entnehmen, in der die Cluster des Tools dem Menschen zugeordnet werden und der Teil des Gelenkes dem Roboter. Die fehlerhafte Klassifizierung

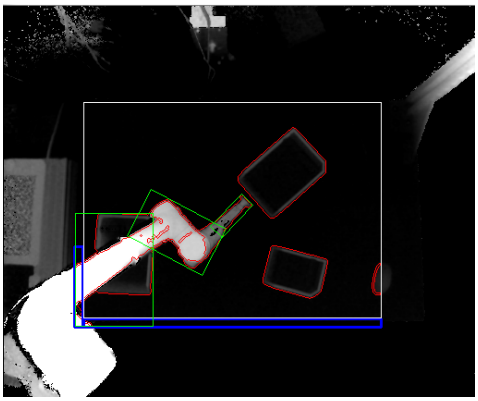
des Tools resultiert aus dem vereinfachtem Robotermodell sowie den ungenauen Rechtecken aus den Clustern, die keine Schnittmenge mit dem Roboter aufweisen. Die Eintrittsversuche aus der zweiten und dritten Situation (vgl. Abbildung 37 c-f) sind dagegen erfolgreich und zeigen eine korrekte Klassifizierung des Roboters. Aus dieser ersten Betrachtung lässt sich ableiten, dass der Roboter wie folgt in den Arbeitsbereich eintreten muss:

- Der Nutzer bewegt den Roboter durch den Free-Drive-Modus in das Arbeitsfeld. Dieses Verfahren ist möglich, da der Free-Drive-Modus von dem erzeugten Befehl, die aktuelle Aufgabe abzubrechen, unberührt bleibt.
- Die Kollisionserkennung wird für den Moment des Eintrittsmomentes deaktiviert.
- Der Roboter tritt an definierten Punkten in das Arbeitsfeld ein.

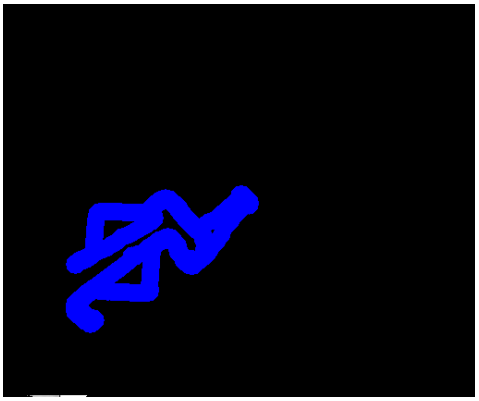
Außerdem darf der Roboter den Arbeitsbereich nur auf eine der drei genannten Möglichkeiten verlassen.

Roboterbewegungen oberhalb des Arbeitsplatzes

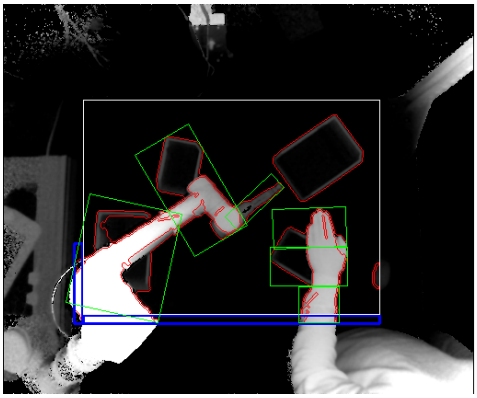
Durch die auf dem Tisch platzierten Boxen soll geprüft werden wie der Algorithmus auf Fremdeinflüsse reagiert, die eine Verfälschung der Kontur verursachen. Dazu wird der Roboter in drei unterschiedlichen Situationen über dem Arbeitsbereich positioniert, der mit mehreren Boxen belegt wird.



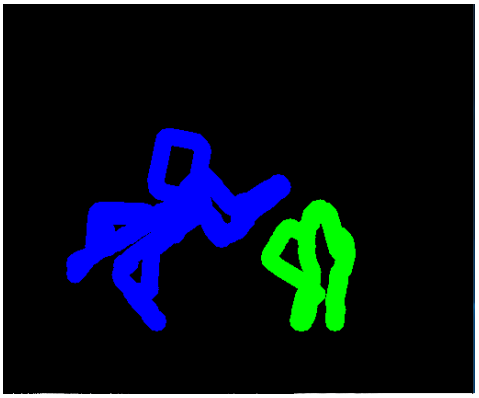
(a) Situation 1: Darstellung der Cluster (grün) und Konturen (rot).



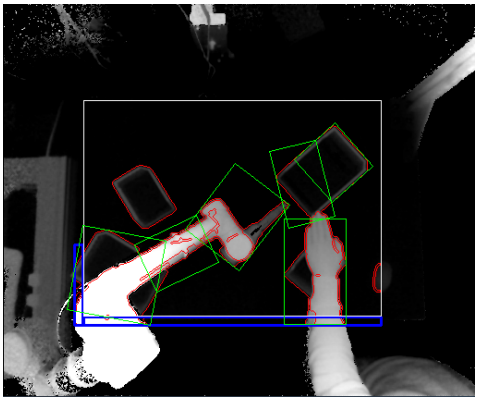
(b) Situation 1: Die Kollisionsmap mit korrekt klassifiziertem Roboter.



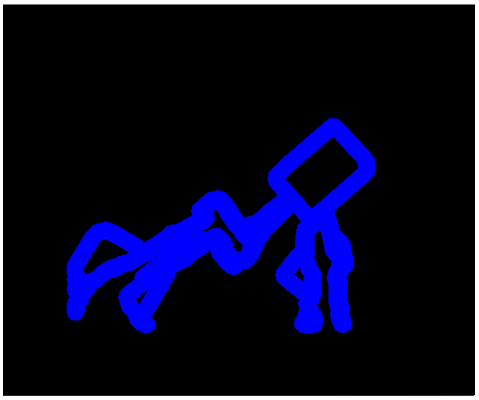
(c) Situation 2: Darstellung der Cluster (grün) und Konturen (rot).



(d) Situation 2: Die Kollisionsmap mit korrekt klassifiziertem Roboter (blau) und Menschen (grün).



(e) Situation 3: Darstellung der Cluster (grün) und Konturen (rot).



(f) Situation 3: Die Kollisionsmap mit fehlerhaft klassifiziertem menschlichen Arm.

Abbildung 38: Gegenüberstellung verschiedener Situationen, in denen der Roboter über den Boxen stationiert ist.

Die in Abbildung 38 gezeigten Situationen repräsentieren nur einen geringen Bruchteil der möglichen Situationen, erlauben aber eine Einschätzung des Systemverhaltens. Die Überlagerung der Konturen zwischen dem Roboter und den Boxen führt in zwei der drei gezeigten Situationen zu einer korrekten

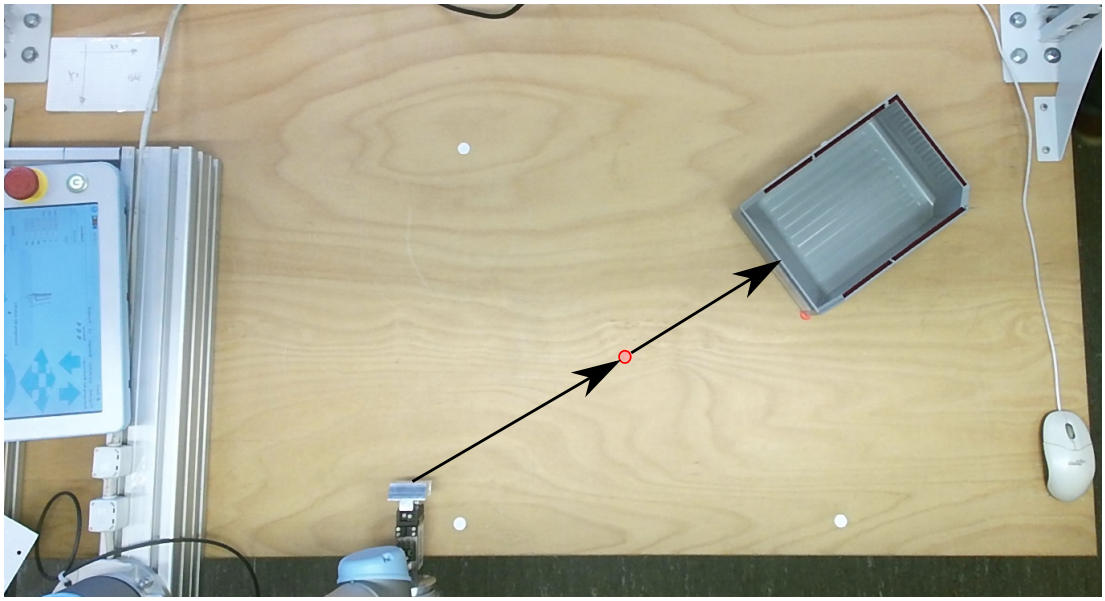


Abbildung 39: Messaufbau für die Erfassung der Kollisionen unter variierender Roboter-Geschwindigkeit und -Beschleunigung sowie unterschiedlichen Konturbreiten.

Klassifizierung des Roboters und des Menschen (Abbildung 38 a-d). Während der Messung wurde in keiner der beiden Situationen eine Kollision erkannt. In der dritten Situation werden alle Cluster-Segmente dem Roboter zugeordnet (Abbildung 38 e-f). In diesem Fall erzeugt die Kontur der Box eine Vergrößerung der Cluster-Segmente, weshalb zwei der drei Segmente eine Schnittmenge mit dem in der Nähe befindlichen Roboter aufweisen. Das System hat vor Zusammenschluss der Konturen eine Kollision erkannt, weshalb diese Situation zu einem Stillstand des Roboters geführt hätte. Anhand des Verhaltens des Systems wird angenommen, dass eine durchgehende korrekte Klassifizierung nicht gewährleistet werden kann, wenn der Roboter oberhalb des Arbeitsfeldes tätig ist, da eine erhöhte Menge an falschen Kollisionserkennungen erwartet wird. Zusätzlich muss die Einschränkung getroffen werden, dass der Mensch nicht unter den Roboter greifen darf, da auch hier eine Überlagerung der Konturen entsteht. Diese Aussagen definieren zwei weitere Kriterien an die Arbeitsumgebung.

Geschwindigkeit des Roboters

Eine Kollision kann nur dann rechtzeitig erkannt werden, wenn das Erfassungssystem schnell genug reagiert. Zusätzlich besitzt der Roboter einen Bremsweg, der abhängig von der Geschwindigkeit ist. Aus diesem Grund werden verschiedene Geschwindigkeiten und Konturvergrößerungen untersucht. Abbildung 39 zeigt den Messaufbau. Für die Validierung wird eine Box in den Randbereich des Arbeitsfeldes (Box oben rechts) platziert und ein Transportvorgang initialisiert. Auf der Strecke zwischen der Roboter-Ausgangsposition und der Box (schwarze Pfeile), wird die Hand des Benutzer positioniert und das Kollisionsverhalten festgehalten. Die Position der Hand wird in der Abbildung 39 durch den roten Punkt beschrieben und ist ebenfalls auf dem Tisch zu finden. Die Kollisionserkennung ist erfolgreich, wenn der Roboter stoppt ohne den Menschen zu berühren. Die Betrachtung erfolgt in dem zuvor definierten Umgebungsbedingungen, weshalb der Roboter manuell in den Arbeitsbereich bewegt wird und ausschließlich auf Höhe der Greifposition für die Kisten arbeitet. Die Anfahrt der Griffposition erfolgt durch eine lineare Bewegung des Tools (durch den „movel“-Befehl vgl. Kapitel 7.2.3) und besitzt eine maximale Geschwindigkeit von $v_{Rmax} = 1\frac{\mathrm{m}}{\mathrm{s}}$ [Rob05]. Die Ergebnisse aus Tabelle 11 zeigen die notwendigen Konturbreiten in Abhängigkeit von der Geschwindigkeit des Roboters. Für die maximale Geschwindigkeit von $1\frac{\mathrm{m}}{\mathrm{s}}$ ist eine große Breite von 55 px für die Roboter- und Menschenkontur

Messung Nr.	Geschwindigkeit v_R $[\frac{\mathrm{m}}{\mathrm{s}}]$ /Beschleunigung a_R $[\frac{\mathrm{m}}{\mathrm{s}^2}]$	Konturbreite [px]
1	1	55
2	0,8	45
3	0,6	45
4	0,4	35
5	0,2	20
6	0,1	10
7	0,05	5

Tabelle 11: Messungen mit den erfolgreich verhinderten Kollisionen unter variierter Geschindigkeit, Beschleunigung ($v_R = a_R$) und Konturbreite.

notwendig. Sie schränkt den Arbeitsbereich deutlich ein. Für geeignet erscheint die vierte Messung, die eine Geschwindigkeit von $0,4\,\frac{\mathrm{m}}{\mathrm{s}}$ erlaubt und den Arbeitsbereich ausreichend frei hält. Zusätzlich kann das System mit voller Geschwindigkeit arbeiten, wenn kein Mensch erkannt wird. Dabei ist zu beachten, dass die Klassifizierung aufgrund des Umfangs nicht vollständig validiert wurde und die Konturbreite unter der maximalen Geschwindigkeit sicherheitshalber auf 55 px zu setzen ist.

Zusammenfassung

Die Validierung zeigt, dass die Klassifizierung und Kollisionserkennung auf Basis der Cluster und Rechtecke mit Einschränkungen erfolgreich ist. Eine Einschränkung ergibt sich aus dem zweidimensionalen Modell des Roboters, welches eine Klassifizierung oberhalb der Griffposition der Boxen nicht erlaubt. Weiterhin ist ein kontrolliertes Ein- und Ausfahren in den Arbeitsbereich notwendig, da die Klassifizierung bei getrennten Konturen Fehlinterpretationen aufweist. Die Trennung der Kontur resultiert aus der Filterung relevanter Punkte, die nicht Bestandteil des Arbeitsbereiches sind. Eine Aufnahme der Punkte außerhalb des Arbeitsbereiches würde größere Konturen verursachen und Teile der Umgebung wie das Rauschen des Sensors oder Konturen nicht-relevanter Umgebungsobjekte beinhalten. Die Geschwindigkeit und Kollisionserkennung zeigt, dass ein Kompromiss zwischen Roboter-Geschwindigkeit und Größe des Arbeitsbereiches geschlossen werden muss, da Beides durch den Bremsweg voneinander abhängig ist. Unter Beachtung der genannten Einschränkungen ist zu erwarten, dass die Kollisionen rechtzeitig und sicher erkannt werden können.

7 Praktische Umsetzung

7.1 Entscheidungsgrundlage für die Verwendung der untersuchten Algorithmen

In dieser Umsetzung werden die für die Aufgabenstellung wichtigen Algorithmen umgesetzt und das Zusammenarbeiten der Klassen und Funktionen erläutert. Für die Schräglagenkorrektur findet der Ansatz nach [Goe17] Anwendung, da der Ansatz über die Standardabweichung in der Validierung einen geringen Versatz aufweist und mehrere Iterationen für die Approximation der Ebene benötigt. Für die Vorsegmentierung im Tiefenbild wird der Ansatz über die Hu-Momente verwendet, da dieser zuverlässig und effizient ist und somit das erste Kriterium der Anforderungsanalyse erfüllt. Die aus dem Tiefenbild gewonnen AOIs werden anschließend auf Threads verteilt, in denen die Erfassung der Werteprofile auf Grundlage der Farbinformationen im HSV-Raum nach Gleichung 17 erfolgt. Anschließend werden die roten Markierungen durch das Korrelationsverfahren erfasst und die Schnittpunkte der Maskierungen bestimmt. Auf dieser Grundlage kann die Griffposition genau bestimmt werden, wodurch das dritte Kriterium der Anforderungsanalyse erfüllt wird. Gleichzeitig wird aufgrund der höheren Prozesslaufzeit das erste Kriterium verletzt. Hierbei handelt es sich um eine Notwendigkeit und einen Kompromiss, da die Ansätze auf Grundlage des Tiefenbildes das vierte Kriterium der Anforderungsanalyse verletzen und nicht ausreichend genau sind.
Um eine Kollision mit dem Roboter zu verhindern, werden in der Vorsegmentierung die konvexen Hüllen mit einer Schnittmenge im Sicherheits- und Arbeitsbereich verwendet, um diese anschließend durch den K-Means-Algorithmus in drei Cluster je Hülle zu unterteilen. Eine Vergrößerung der so gewonnen Konturen (als Cluster) soll den Anforderungen des zweiten Kriteriums der Anforderungsanalyse gerecht werden. Die ausführliche Erläuterung der praktischen Umsetzung erfüllt das fünfte Kriterium, womit angenommen wird, dass alle Kriterien erfüllt werden können.

7.2 Schnittstellen

7.2.1 Schnittstelle zu der Kinect

Die Schnittstelle zu der Kinect v.2 wird durch das Microsoft Kinect-SDK bereitgestellt und gibt eine strikte Prozessstruktur vor. Geringe Abweichungen in der Abfrage von Informationen führen bereits zu einem Fehler in der Abfrage neuer Bilder. Die Schnittstelle und Struktur wird aus der Arbeit von Herrn Kleborn [Kle16] abgeleitet und in einem eigenen Thread umgesetzt. Die Verwendung eines Threads bietet den Vorteil, dass die Datenerfassung und Aufbereitung parallel zu dem Bildverarbeitungsprozess stattfindet und das nächste Frame nach Beenden des vorangegangenen Prozesses zur Verfügung gestellt wird. Die Abfrage des Tiefen-, Farb- und IR-Bildes erfolgt dabei durch den Ablauf aus Abbildung 40, der für jedes Frame identisch ist.

Abbildung 40: Verarbeitungskette der Kinect für das Empfangen und Aufbereiten der Frames (in Anlehnung an [Kle16, Abbildung 13]).

Abbildung 40 zeigt, dass über den geöffneten Sensor die Auswahl der Quelle (Source) erfolgt. Für das Tiefenbild gilt nachstehender Ablauf:

```
//Öffnet den Sensor
hr = sensor->Open();

//Öffnet den Multi Source Frame Reader
hr = sensor->OpenMultiSourceFrameReader(FrameSourceTypes_Depth,&msfr);

//Verknüpft einen Eventhandler
hr = msfr->SubscribeMultiSourceFrameArrived( &newFrameEventHandle );

//Fragt das letzte Frame ab
hr = msfr->AcquireLatestFrame(&MultisourceFrame);

hr = MultisourceFrame->get_DepthFrameReference(&DepthFrameReference)

hr = DepthFrameReference->AcquireFrame(&DepthFrame);

hr =DepthFrame->CopyFrameDataToArray(width * height, Array);

//Aufbereitung des Frames
        //...

//Freigeben der Interfaces

hr = DepthFrame->Release();

hr = DepthFrameReference->Release();

hr = MultisourceFrame->Release();
```

Das Programm-"Listening" zeigt, dass im ersten Schritt der Sensor und das Interface „MultiSourceFrameReader" geöffnet werden. Anschließend wird dieses mit einem Eventhändler verknüpft, der bei Eintreffen eines Frames ausgelöst wird. Hierbei ist zu beachten, dass der Eventhändler, abhängig von den benötigten Frames, für jedes Frame pollt. Es kann daher passieren, dass ein neues Tiefenframe eintrifft, während das Frame für den Farb- und Infrarotsenor noch nicht bereit ist. Eine Abfrage führt in diesem Fall zu einem Fehlercode in der Variable hr vom Typ „HRESULT". Ist die Abfrage des Frames erfolgreich, wird die Referenz des Frames verwendet, um die Daten in ein Array zu speichern. Nach der Verabeitung werden die Interfaces in der umgekehrten Reihenfolge freigegeben wie sie geöffnet wurden. Eine Abweichung in der Reihenfolge führt zu einem Fehler. Zusätzlich kann nur ein Frame zur Zeit abgefragt werden [Kle16, Seite: 21]. Sobald ein neues Multi-Source-Frame eintrifft, beginnt die Kette erneut. Zum Beenden der Kinect werden Reader, Sensor, Interface und Eventhändler geschlossen.

```
//Schließen des Sensors
hr = msfr->UnsubscribeMultiSourceFrameArrived(newFrameEventHandle);
hr = msfr->Release();
sensor->Close ( ) ;
sensor->Release ( ) ;
```

Für die Umsetzung in einem Thread findet die QT-Klasse „QThread", die von der Klasse „QObject" erbt und die virtuelle Funktion

```
void run();
```

bereit stellt, Anwendung. Diese virtuelle Funktion wird mit dem Abfrageprozess des Eventhändlers überschrieben, verwaltet die Threadmutex und enthält eine dauerhafte „while"-Schleife, die durch die Funktionen „start()" und „stop()" der Klasse „framesource" gesteuert wird. Trifft ein neues Frame ein, wird die Mutex in der „run()"-Funktion gesperrt und der Prozess für die Abfrage der Frames durch den Aufruf der Funktion

```
bool doWork(HANDLE hEvent);
```

bearbeitet. Konnten alle Frames empfangen werden, bleibt die Mutex gesperrt und ein Signal wird emittiert, um die Daten abzuholen. Erst wenn die Mutex freigegeben wird, kann das nächste Frame eintreffen. Auf diese Weise werden die Daten der Sensoren dem Programm zur Verfügung gestellt und erlauben die Bildverarbeitung sowie die Visualisierung durch die OGRE-Engine.

7.2.2 Schnittstelle zu der OGRE-Engine

Die Integration der OGRE-Schnittstelle in das QT-Framework erfolgt über die QT-Klasse „QWidget", indem die Render-Klasse von dieser erbt. In der GUI wird die Render-Engine aufgrund der Unterklasse durch ein Layout eingebunden und verfügt über das Signal-Slot-Verfahren. Durch Verwenden eines Layouts können Maus- und Tastaturbefehle innerhalb und außerhalb des Widgets getrennt behandelt werden. Eine Eingabe innerhalb des Layouts löst ein Event in der Unterklasse aus und kann in der Hauptklasse verwendet werden:

```
void RenderEngine::mousePressEvent(QMouseEvent *event){
        //do something...
        event->accept();
}
```

Damit die Visualisierung in dem QWidget erfolgt, benötigt OGRE (mRoot) den Namen des Fensters sowie die Breite, die Höhe, den Initialisierungswert und die ID des Widgets. Diese Informationen werden ebenfalls durch die Unterklasse bereitgestellt und ermöglichen im GUI-Creator ein frei skalierbares Darstellungsfenster. Die Übergabe der Parameter erfolgt durch:

```
mWindow = mRoot->createRenderWindow("PointCloud",
 WindowWidth,WindowHeight,false,&parameterWindow);
```

```
mWindow->setVisible(true);
```

Um ein Frame zu rendern, kann mit einer „puplic“-Funktion auf den Render-Befehl der Engine zurückgegriffen werden:

```
void RenderEngine::RenderOneFrame() {
        mRoot->renderOneFrame();
}
```

Anschließend erfolg eine Aktualisierung in dem QWidget. Für die Darstellung von Objekten wie Boxen (Quader) oder Arme des Roboters (Zylinder) werden folgende Funktionen verwendet:

```
// Erstellt einen Zylinder
void RenderEngine::createCylinder(Point3f firstPoint,
 Point3f secondPoint, int radius, int id, ColourValue colour);

//Erstellt einen Quader
void RenderEngine::createCube(int length, int height,
 int width, int id, ColourValue color, Point3f position,
 vector<cv::Point3f> corners);
```

7.2.3 Schnittstelle zu dem UR5-Roboter

Für die Schnittstelle zu dem Roboter werden Teile aus der Arbeit von Herrn Hoch verwendet [Hoc16]. Die Klasse „ProgManager“ nimmt über Ethernet eine TCP-Verbindung mit dem Dash-, Primary- und Secondary-Client auf und stellt, über das Ausführen eines „Skills“, die Möglichkeit bereit Fahr- oder Werkzeugbefehle auszuführen. Ein Skill (z.D: Fähigkeit) beschreibt nach der Arbeit von Herrn Frank Hoch eine Aufgabenstellung für den Roboter, die in Kombination mit weiteren Modulen selbstständig abgearbeitet werden kann und wurde aus der Definition des Manipulationsprimitivs der Promotion von Herrn Prof. Dr. Jochen Maaß abgeleitet. Ein Skill verfügt über folgende Eigenschaften [Hoc16, Seite: 27]:

- Skilltyp (interpolierende, einfache Trajektorie, öffnende oder schließende Werkzeugmanipulation)
- Bezugssystem (Gelenke oder Robotertool)
- Einheit des Bezugssystems (mm, m, Rad, Deg)
- Zielzustand (Zielposition oder gewünschte Armwinkel)
- Attributen des Skills (Beschleunigung und Geschwindigkeit).

Für Grundaufgaben wird die entsprechende Struktur eines Skills in der Arbeit von Herrn Hoch durch eine XML-Vorlage bereitgestellt. Für eine lineare Fahrbewegung zu einer Zielposition, mit Bezug auf das Werkzeug des Roboters im metrischen System, wird zum Beispiel die Skillvorlage „Skill3.xml–SimpleMovement“ verwendet. Diese wird im ersten Schritt in das Objekt geladen und dessen Attribute entsprechend der Aufgabenstellung angepasst. Durch die Klasse „ProgManager“ kann der Skill ausgeführt und die Aufgabenstellung abgearbeitet werden.
Die Ausführung diverser Skills führt in einigen Situationen zu einem Stopp der Aufgabe, deren Ursache, aufgrund des Umfanges des Frameworks, nicht gefunden werden kann. Aus diesem Grund wird in dieser Arbeit die Schnittstelle zu dem Secondary-Client des Roboters verwendet, um Fahrbefehle auszuführen. Die Abfrage der Tool- oder Joint-Positionen erfolgt weiterhin durch die Schnittstelle von Herrn Hoch. Für die Bewegung des Roboters stehen die zwei Befehle movej(q, a=1.4, v=1.05, t=0, r=0) sowie movel(pose, a=1.2, v=0.25, t=0, r=0) zur Verfügung:, die über die Methode sendCmd(QString) an den Secondary-Client gesendet werden. Dabei definiert „q“ die Gelenkstellungen, die mit der

Beschleunigung a und der Geschwindigkeit v erreicht werden sollen. Alternativ kann eine Zeitdauer t zum Erreichen der Position vorgegeben werden. Eine Pose besteht aus den Zielkoordinaten sowie dem Rotationsvektor des Tools. Das „j“ signalisiert einen Bewegungsablauf mit Bezug auf die Gelenke und das „l“ auf eine lineare Bewegung mit Bezug auf das Tool. Der Befehl, um das Tool „Linear“ mit einer Geschwindigkeit und Beschleunigung von $v = 1\,\frac{\mathrm{m}}{\mathrm{s}}$, $a = 0.5\,\frac{\mathrm{m}}{\mathrm{s}^2}$ an die Position $x = 0.1\,\mathrm{m}$, $y = 0.1\,\mathrm{m}$, $z = 0.1\,\mathrm{m}$ mit dem Rotationsparametern $Rx = 0\,\mathrm{rad}$, $Ry = 0\,\mathrm{rad}$, $Rz = 1,57\,\mathrm{rad}$ zu bewegen, lautet:

```
m_controllInterface->sendCmnd(
ProgManager::ReceiverType::RT_SECONDARY_CLIENT,
QString("%1(get_inverse_kin(p%2,get_actual_joint_positions(),
maxPositionError=0.0001,maxOrientationError=0.0001),
a=%3,v=%4,t=0,r=0)")
.arg("movel")
.arg([0.1, 0.1, 0.1, 0, 0, 1.57])
.arg(0.5)
.arg(1.0));
```

Die Funktion get_inverse_kin() übernimmt die Umrechnung der inversen Kinematik und erlaubt die Übergabe der Zielposition des Tools. Da die Befehle „movel“ und „movej“ einen Rotationsvektor erwarten und dieser die Komplexität des Bewegungsprozesses erhöht, enthält die Klasse „robotcontroll“ die Funktion „rotationToEuler()“, die den Rotationsvektor auf Grundlage der Rotationswinkeln um die Achsen des Tools (Yaw,Roll,Pitch) berechnet. Für die spätere Ansteuerung ist zu beachten, dass unter Verwendung des „movel“ Befehls häufiger Singularitäten in den Gelenksstellungen zu erwarten sind und eine geeignete Ausgangsposition vorausgesetzt wird [Sko].

7.3 Programmstruktur

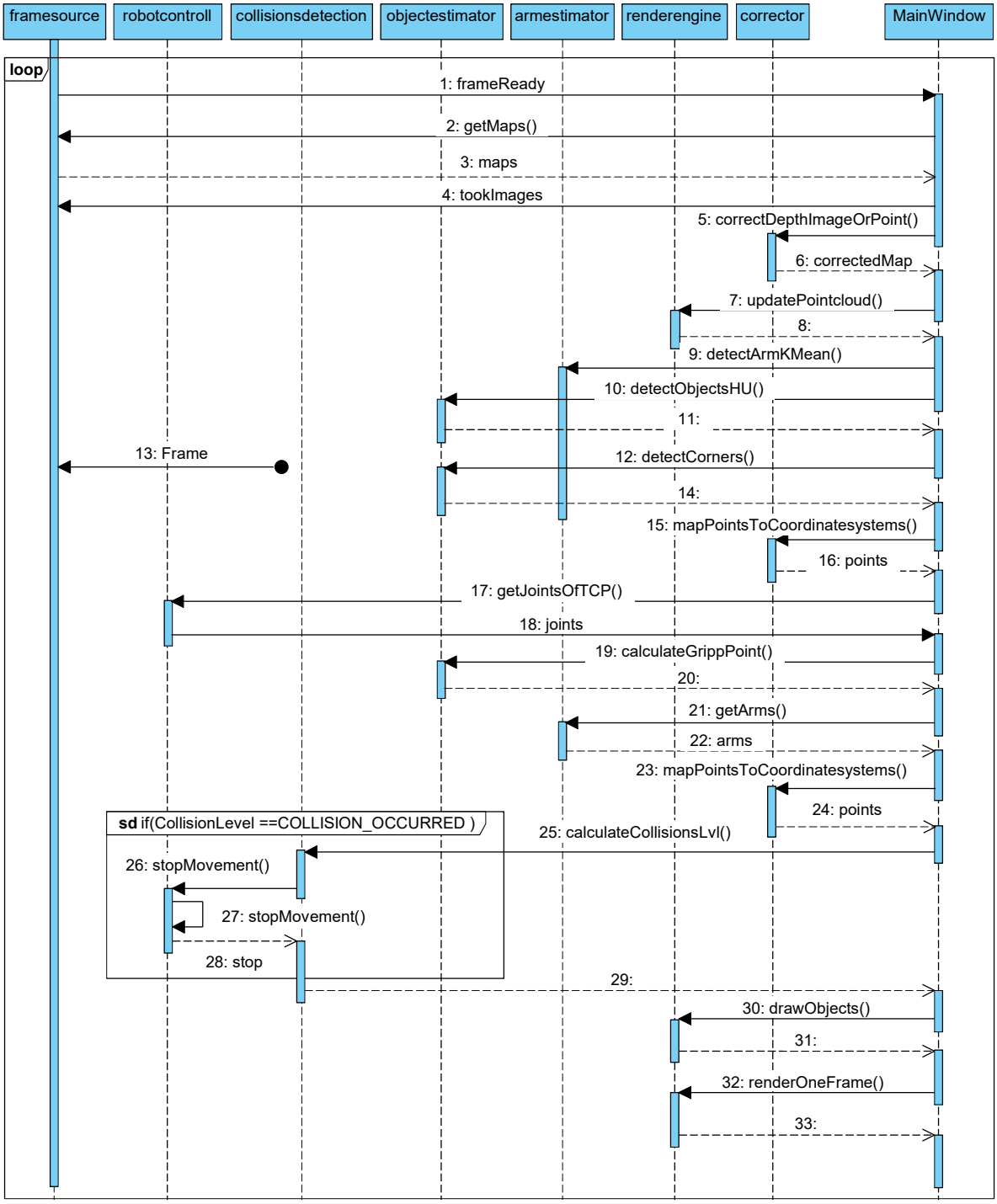


Abbildung 41: Sequenzdiagramm der vollständigen Applikation.

Abbildung 41 zeigt das vereinfachte und zusammengefasste Sequenzdiagramm der umgesetzten Applikation. Zu erkennen ist die modulare Struktur und die zentrale Klasse „Main Window“, die alle anderen

Klassen miteinander verbindet. Aus sicherheitsrelevanten Gründen ist die Klasse „collisionsdetection“ direkt mit der Klasse „robotcontroll“ verbunden, um im Fall einer erkannten Kollision, den Stopp der Roboterbewegung unabhängig von den anderen Prozessen auszuführen. Zusätzlich wird für eine Optimierung der Laufzeit mit der Arbeitsverteilung auf mehreren Threads gearbeitet. So befindet sich der Bilderfassungs- und Bildaufbereitungsprozess in der Klasse „framesource“ in einem eigenständigen Thread. Diese Struktur erlaubt eine Bildverarbeitung der weiteren Klassen, während das nächste Frame (vgl. Signal 13 in Abbildung 41) eintriftt und aufbereitet wird. Darüber hinaus basiert die Arm- und die Objekterkennung auf der Informationsgrundlage des Tiefenbildes. Aus diesem Grund arbeitet die Klasse „armestimator “ in einem eigenen Thread. Dadurch können diese Prozesse zeitgleich bearbeitet werden. Erst am Ende der Prozesskette (vgl. Signal 25 in Abbildung 41) werden die Informationen über die Arme für die Kollisionserkennung benötigt. Nach der Erfassung der Boxen im Tiefenbild erfolgt das Mapping der Punkte in das Farbbildung durch die Kinect. Dieser Prozess ist in dem Sequenzdiagramm aus Platzgründen nicht dargestellt und wird in der MainWindowklasse aufgerufen. Im Anschluss erfolgt die Verabeitung im Farbbild, deren Ergebnis in die notwendigen Koordinatensysteme durch die Klasse „corrector“ transformiert wird. Anschließend kann er Greifpunkt für den Roboter bestimmt werden (vgl. Signal 19 in Abbildung 41). Die zuvor erkannten Arme werden durch das Signal 21 abgefragt und zusammen mit den Robotergelenken in das Weltkoordinantensystem transformiert, damit anschließend die Kollisionserkennung gestartet wird (Signal 25). Anschließend erfolgt die Visualisierung der Objekte durch die Klasse „renderengine“. Die einzelnen Strukturen der Klassen werden durch UML-Diagramme in den nachstehenden Unterkapiteln erläutert.

7.3.1 Die Klasse „framesource“

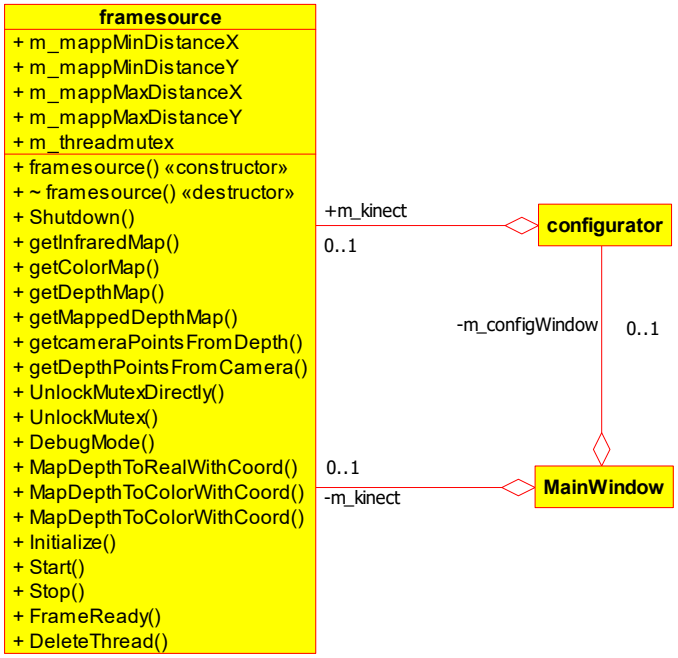


Abbildung 42: UML- Klassendiagramm der Klasse „framesource“.

Die Klasse „framesource“ stellt die Schnittstelle zu der Kinect dar und arbeitet in einem eigenem Thread. Sie wird von der Hauptklasse „Mainwindow“verwendet, um die Daten an die anderen Klassen weiterzugeben sowie von der „configurator“-Klasse, um die Kalibrierung vorzunehmen. Mit der Member-Funktion „initialize“ wird die Verbindung zu der Kinect hergestellt und der Empfangs- sowie Bildaufbereitungsprozess gestartet. Anschließend fragt die Klasse einen Eventhändler auf neue Frames ab, die anschließend für das Farb-, Tiefen- und Infrarotbild aufbereitet werden. Diese können über die „get...Map()“-Funktion abgefragt werden. Der Zugriff auf den Coordinate-Mapper erfolgt durch die „MapDepthTo...“-Funktionen. Um den Thread zu schließen und den Datenstream zu der Kamera korrekt zu beenden, muss vor Schließen des Threads die Funktion „Shutdown()“ aufgerufen werden. Diese beendet die Hauptschleife im Thread und gibt die verwendete Mutex wieder frei. Soll die Verbindung zu der Kinect bestehen bleiben ohne Frames zu verarbeiten, können die Funktionen „stop()“ und „start()“ verwendet werden. Die Slots „UnlockMutexDirectly“ gibt die Mutex wieder frei und führt im Falle einer freien Mutex zu einem Absturz. Sie ist daher nur zu verwenden, um ein erhaltenes Frame zu quittieren und den Prozess für das nächste Frame zu starten. Die Funktion „UnlockMutex“ findet während des Schließens der Software Anwendung, um eine freie Mutex sicherzustellen. Diese prüft vor der Freigabe der Mutex, ob diese gesetzt werden kann und ist gegenüber der Funktion „UnlockMutexDirectly“ langsamer. Sobald alle drei Frames aufbereitet sind, emittiert die Funktion über „FrameReady()“, dass diese abholbereit sind. Solange die Mutex anschließend nicht zurückgesetzt wird, werden keine weiteren Frames verarbeitet. Das Signal „DeleteThread()“ sendet ein Signal, sobald das Objekt selbst korrekt beendet wurde und ein Beenden des Threads möglich ist.

7.3.2 Die Klasse „configurator“

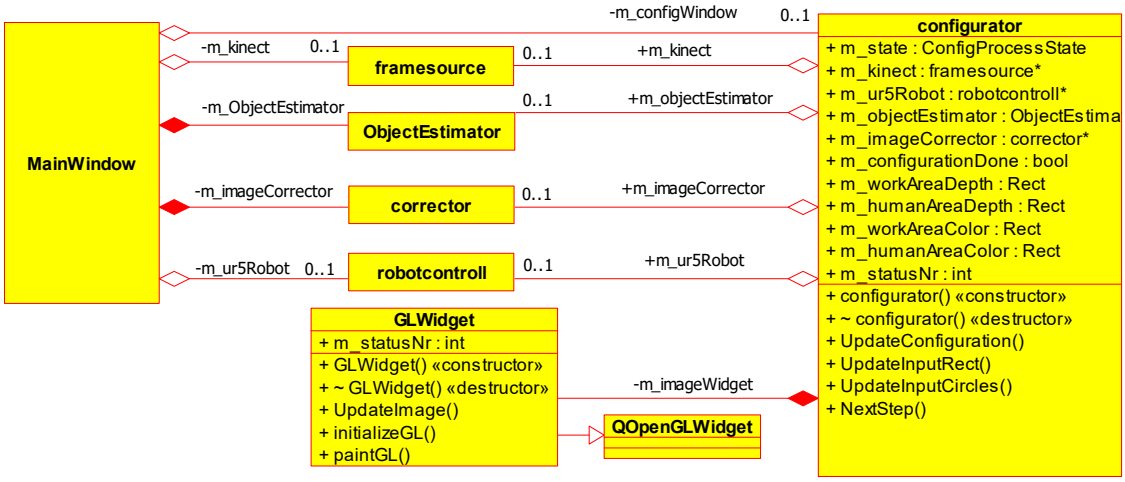


Abbildung 43: UML- Klassendiagramm der Klasse „configurator“.

Die Klasse „configurator“ parametriert die Klassen „objectEstimator“ und „corrector“. Für die Darstellung der Bilder, die dem User für die Kalibrierung angezeigt werden, verwendet die Klasse ein „GLWidget“, das von der Klasse „OpenGLWidget“ erbt. Bilder können über „UpdateImage()“ in der GUI aktualisiert werden. Eingaben des Nutzers werden über die Signale „mousePressEvent“, „mouseReleaseEvent“ und „mouseMoveEvent“ des GLWidgets erfasst und die entsprechenden Aktionen wie das Einzeichnen der Referenzpunkte oder des Arbeitsbereiches an diese Signale (zum Beispiel die Memberfunktion „UpdateInputRect()“ gekoppelt. Ist der Konfigurationsprozess vollständig oder die „config.xml“-Datei erfolgreich eingelesen, werden die Informationen an die zu konfigurierenden Klassen weitergeben und das Objekt emittiert das Signal „ConfigProcessComplete()“. Dieses sorgt dafür, dass in der „MainWindow“-Klasse das Objekt „m_configWindow“ gelöscht wird und das MainWindow-Fenster geöffnet wird. Da die „configurator“-Klasse mit Zeigern der anderen Klassen arbeitet, bleiben diese konfiguriert und können in der MainWindow verwendet werden. Nachstehende Informationen werden durch die Klasse erfasst und an die weiteren Objekte übermittelt:

m_workAreaDepth Der Arbeitsbereich im Tiefenbild, der gleichzeitig das AOI in der Bildverarbeitungskette repräsentiert.

m_transfMatrixCam Die Transformationsmatrix für die Transformation zwischen dem Welt- und dem Roboterkoordinatensystem.

m_transfMatrixRobot Die Transformationsmatrix für die Transformation zwischen dem Kamera- und dem Weltkoordinatensystem.

Sofern keine „config.xml“-Datei existiert, werden diese mit den neu gewonnen Informationen erstellt und bei dem nächsten Programmstart eingelesen.

7.3.3 Die Klasse „corrector“

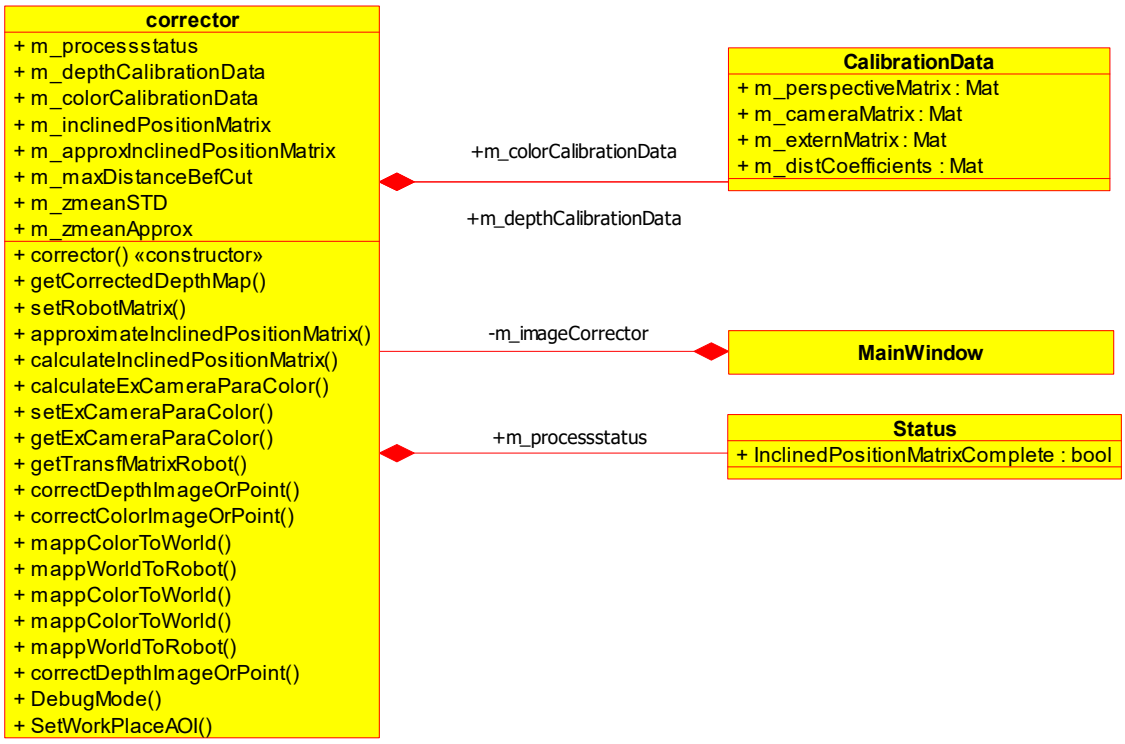


Abbildung 44: UML- Klassendiagramm der Klasse „corrector“.

Die Klasse „corrector“ übernimmt alle Bildaufbereitungsschritte wie die Korrektur der Schräglage, der Verzeichnung oder die Umrechnung zwischen den Koordinatensystemen. Die Meberfunktion „correctDepthImageOrPoint()“ ist überladen und übernimmt die Korrektur eines Punktes oder der gesamten Tiefenbildmatrix. Gleiches gilt für „mapColorToWorld()“ sowie „mapWorldToRobot()“. Die intrinsischen Kameraparameter werden in der Klasse „CalibrationData“ für den Farb- und Tiefensensor gespeichert und für die Transformation zwischen Kamera- und Weltkoordinatensystem in den Memberfunktionen „mappColorToWorld“ verwendet. Zuvor werden diese Parameter während der Initialisierung der Klasse aus der „config.xml“-Datei gelesen. Die Transformationsvorschriften der externen Parameter erhält die Klasse „configurator“ mit der Funktion „setExCameraParaColor()“. Um eine Korrektur der Schräglage durchzuführen, muss vorher wahlweise die Funktion „calculateInclinedPositionMatrix()“ oder „ApproximatePositionMatrix()“ aufgerufen werden. Ersteres basiert auf dem Ansatz der Standardabweichung und wird iterativ ausgeführt bis die variable „InclinedPositionMatrixComplete“ wahr ist.

7.3.4 Die Klasse „armestimator“

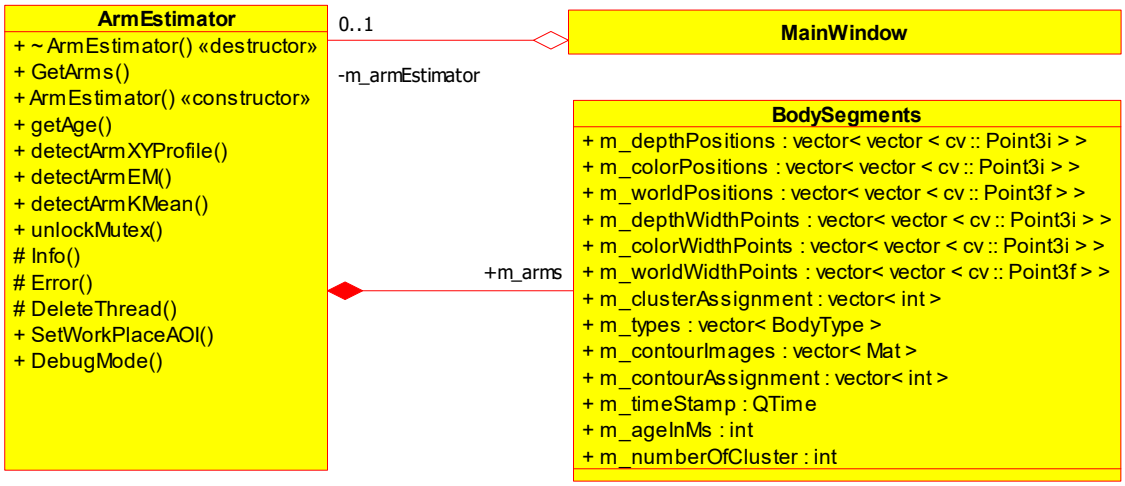


Abbildung 45: UML- Klassendiagramm der Klasse „armestimator“.

Die Klasse „armestimator“ übernimmt durch die Funktionen „detectArm...()“ die Erkennung und Clusterung relevanter Konturen und speichert diese in der Klasse „BodySegment“, sofern zuvor das AOI durch die Klasse „configurator“ übermittelt wurde. Da der Erkennungsprozess parallelisiert auf mehreren Threads bearbeitet wird, muss die Funktion „unlockMutex“ aufgerufen werden bevor das Objekt zerstört wird. Der Thread kann gelöscht werden sobald das Signal „DeleteThread()“ emittiert wird. Zu beachten ist, dass vor dem Zerstören des Objektes keine weitere „detectArm...“-Funktion aufgerufen werden und die Mutex dadurch gesperrt wird. Das durch die Klasse erzeugte BodySegment enthält dabei die Elemente eines Arms in Form von Vektoren mit je zwei richtungsweisenden Punkten (Positionsvektoren) und zwei orthogonal liegenden Punkten mit der halben Breite des jeweiligen Clusters (WidthPoints). Sie liegen, abhängig von dem Fortschritt in der Verarbeitungskette, im Tiefenbild, Farbbild und im Welkoordinantensystem vor. Die Zuordnung zwischen dem Element eines Armes und dem Cluster ist in dem Vektor „m_clusterAssignment“ hinterlegt. Ob es sich bei dem Segment um einen Teil des Menschen oder des Roboters handelt, wird in dem Vektor “m_types“ mit den Typen

BODY_TYPE_HUMAN für einen Menschen

BODY_TYPE_ROBOT für einen Roboter

hinterlegt, welcher erst duch die Klasse „collisionsdetect“ beschrieben wird. Da der Clustering-Prozess bereits auf Threads verteilt ist, wird die Klasse genutzt, um die geclusterten Konturen in eine Teil-„Collisionsmap“ zu zeichnen, die nach der Klassifizierung der Cluster in die gesamtheitliche „Collisionsmap“ eingebunden werden.

7.3.5 Die Klasse „collisiondetect“

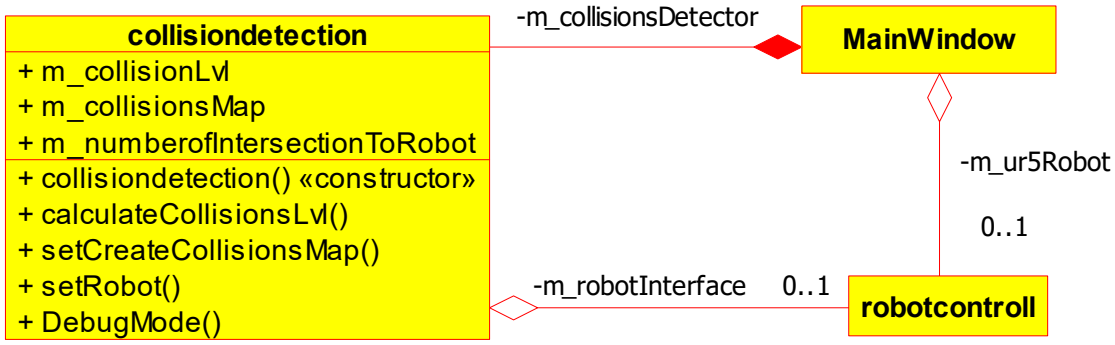


Abbildung 46: UML- Klassendiagramm der Klasse „collisiondetection“.

Die Klasse „collisionsdetect“ klassifiziert die aus dem „armestimator“ erkannten Body-Segmente, indem ein Abgleich mit den Elementen des Roboters stattfindet. Die Funktion „calculateCollisionsLvl“ erhält die BodySegmente und das inverse Modell des Roboters und gibt anschließend das Kollisionslevel wie folgt zurück:

COLLISION_OCCURRED Es konnte eine Kollisions zwischen Roboter und Mensch erkannt werden.

COLLISION_POSSIBLE Eine Kollision kann entstehen, da sich der Mensch und der Roboter in dem Arbeitsbereich befinden.

COLLISION_IMPROBABLE Eine Kollision ist unwahrscheinlich, da Mensch und Roboter nicht gleichzeitig im Arbeitsbereich sind.

Zusätzlich konstruiert die Klasse die Collisionsmap, wenn Roboter und Mensch im Arbeitsbereich sind oder die private Variable „m_createCollisionsMap“ durch die Funktion „setCreateCollisionsMap“ auf „true“ gesetzt ist. Die Collisionsmap kann anschließend für die Visualisierung der Punktwolke verwendet werden, da sie die Teil-Collisionsmaps aus dem Prozess der Arm-Erkennung farblich im Grün- und Blau-Kanal zuordnet. Im Falle einer erkannten Kollision wird direkt über die Klasse „robotcontroll“ der aktuelle Arbeitsablauf des Roboters gestoppt.

7.3.6 Die Klasse „objectestimator“

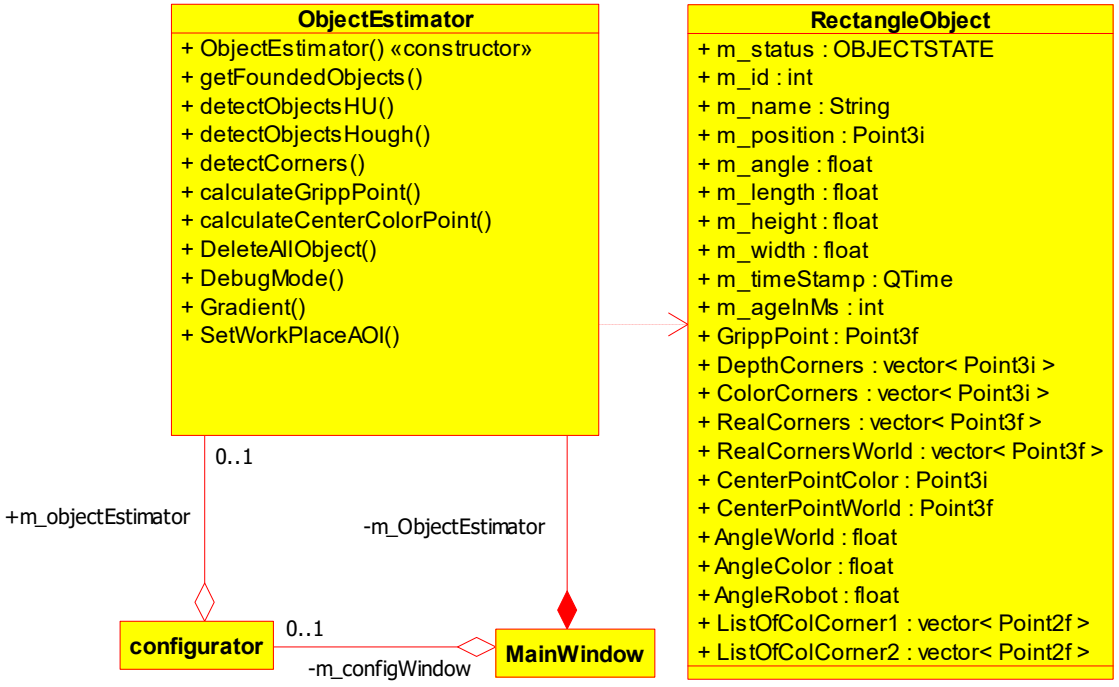


Abbildung 47: UML- Klassendiagramm der Klasse „objectestimator“.

Die Klasse „objectestimator“ erkennt und berechnet die Eigenschaften der Transportboxen im Arbeitsbereich, die in dem Programm durch die Klasse „RectangleObject“ definiert sind. Für eine Erfassung der Transportboxen werden im ersten Schritt mit Übergabe des korrigierten Tiefenbildes die Funktionen „detectObjectsHough()“ oder „detectObjectsHU()“ aufgerufen. Sie erkennen die Transportboxen im Tiefenbild und speichern Merkmale wie Breite, Länge Winkel, Punkte der Ecken oder die erstellte ID, in einem neuen oder bestehenden (es handelt sich um eine bereits erkannte Box) Rectangle-Object. Durch Aufrufen der Memberfunktion „detectCorners()“ werden die im Tiefenbild erkannten Objekte im Farbbild genauer beschrieben, wenn die Box sich bewegt hat oder neu erkannt wurde. Dies geschieht wahlweise durch den Ansatz des Gradienten, indem die Variable „m_useGradient“ durch die Funktion „Gradient()“ auf „true“ gesetzt ist oder durch den Ansatz der Kreuzkorrelation, wenn „m_useGradient“ auf „false“ gesetzt ist. Die im Tiefenbild erstellten oder aktualisierten „RectangleObjects“ werden anschließend über die Ortsinformation der Ecken im Fabbild erweitert. Diese werden auf Basis des Mittelwertes aus den erkannten Punkten in den Vektoren „ListOfColCorner1“ und „ListOfColCorner2“ gebildet, um die Genauigkeit zu erhöhen. Sind die Punkte durch die Klasse „corrector“ in das Weltkoordinatensystem transformiert, kann in diesem der Greifpunkt und der Greifwinkel durch die Funktion „calculateGrippPoint“ bestimmt werden. Eine weitere Transformation dieses Punktes in das Roboterkoordinantensystem ermöglicht anschließend den Transport einer Box.

7.3.7 Die Klasse „renderengine“

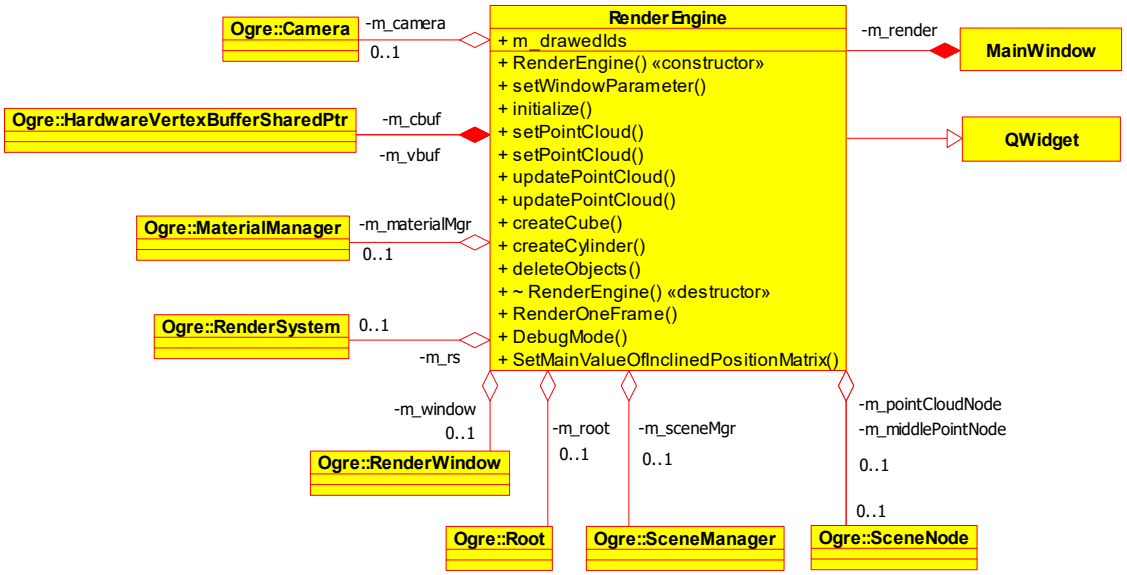


Abbildung 48: UML- Klassendiagramm der Klasse „RenderEngine“.

Über die Klasse „renderengine“ erfolgt die Schnittstelle zu der OGRE-Engine. Die Visualisierung der Punktwolke erfolgt durch die überladene Funktion „updatePointcloud()“ und wird wahlweise mit dem Tiefenbild oder einem Vektor mit den Punkten aus dem Coordinate-Mapper aufgerufen. Zuvor erfolgt eine Initialisierung der Vertex-Buffer für die Punkte („m_vbuf“) und deren Farbe („m_cbuf“) durch die Funktion „setPointCloud()“. Zylinder können durch die Funktion „createCylinder()“ und Quader durch die Funktion „createCube()“ erstellt werden. Die Funktion „createCylinder()“ ist kompatibel zu dem Format der Vektoren in einem Body-Segment und erwartet den Start- und Endpunkt des Richtungsvektors sowie den Radius des Zylinders. In dem Render existiert ein Mittelpunkt („m_middlePointNode“), an den die Punktwolke (m_pointCloudNode) angeknüpft ist. Neue Objekte wie ein Zylinder werden mit dem Punkt der Punktwolke verknüpft und beziehen sich daher immer auf deren Bezugpunkt. Für jedes erstellte Objekt existiert eine eindeutige ID durch die mit der Funktion „deleteObjects“ die gerenderten Objete entfernt werden können. Um das QWidget zu updaten, erfolgt ein Aufruf der Funktion „RenderOneFrame()“. Das Erstellen und Verknüpfen der OGRE eigenen Klassen (Kamera, Material Manager etc.) wird in der „initialize()“-Funktion umgesetzt, die vor allen anderen Funktionen aufgerufen werden muss.

7.3.8 Die Klasse „robotcontroll“

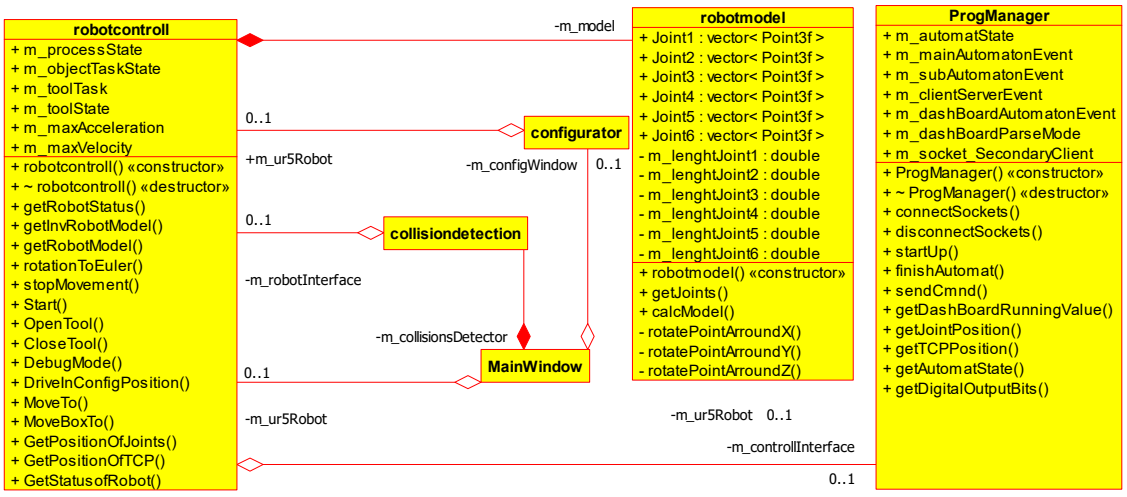


Abbildung 49: UML- Klassendiagramm der Klasse „robotcontroll“ mit Beziehung zu der Klasse „ProgrManager“ von Herrn Frank Hoch [Hoc16].

Die Klasse „robotcontroll“ beschreibt die Schnittstelle zu der Klasse von Herrn Frank Hoch und somit die Schnittstelle zu dem Roboter. Dabei wird die Klasse „ProgManager“ dazu verwendet eine Verbindung zu den Servern des Roboters aufzubauen, um an diese Befehle für Fahrbewegungen und Tool-Zustandsänderungen zu senden. Diese Befehle werden durch die Funktion „sendCmmd()“ an den jeweiligen Server übertragen und ausgeführt. Die Position des Tools wird durch die Funktion „getTCP-Position“ bereitgestellt und die Stellung der Gelenke durch die Funktion „getJointPosition()“. Eine Abfrage der digitalen Ausgänge für das Tool erfolgt über „getDigitalOutputBits()“. Gemeinsam dienen diese Funktionen als Grundlage für die Klasse „robotcontroll“ durch die mit dem Befehl „MoveBoxTo“ eine Box von ihrem Ausgangspunkt in den Zielpunkt bewegt wird. Eine einmalige Bewegung zu einem Zielpunkt erfolgt durch die Funktion „MoveTo“. Das Tool wird duch die Funktionen „OpenTool()“ und „CloseTool()“ angesteuert. Wird eine Kollision erkannt oder soll die aktuelle Aufgabe abgebrochen werden, ist die Funktion „stopMovement()“ zu verwenden. Für einen Abgleich der Roboterpositionen und den erkannten Clustern ist das Modell des Roboters notwendig. Dieses wird durch die Klasse „robotmodel“ bereitgestellt und bezieht sich wahlweise auf das Roboterkoordinatensystem („getRobotModel()“) oder das Weltkoordinantensystem („getInvRobotModel“ mit Übergabe der Transformationsmatrix). Für die Bereitstellung des Models werden die Längen der jeweiligen Gelenke benötigt, die nach der Modellierung durch Start- und Endpunkt in den Vektoren Joint1 - Joint6 bereitgestellt werden und mit der Funktion zum Erstellen der Zylinder aus der Klasse „renderengine“ kompatibel sind. Über die Funktion „calcModel()“ können die aktuellen Winkelstellungen das Modell aktualisieren.

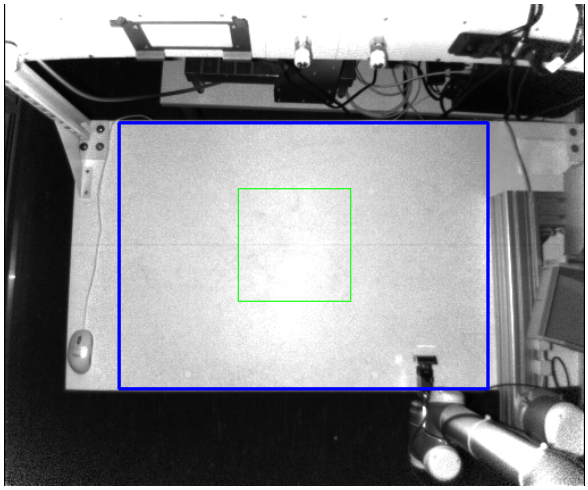
8 Durchführung und Validierung

Das umgesetzte Programm wird durch die Ausführung der folgenden Teilschritte validiert:

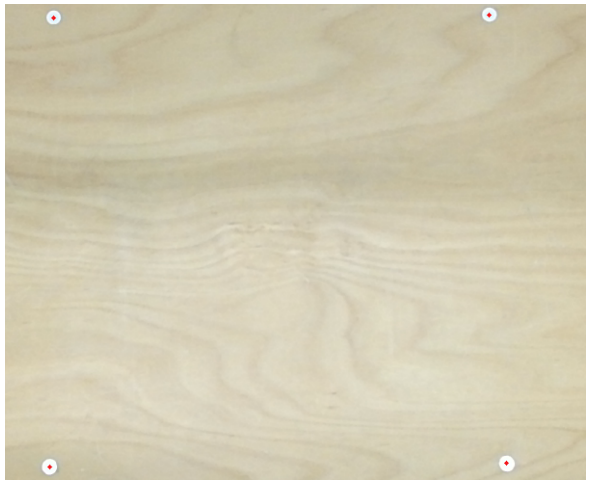
1. Konfiguration des Systems.
2. Greifen und Bewegen der kleinen und großen Boxen durch den Roboter.
3. Provokation von Kollisionen im Arbeitsbereich im Stillstand des Roboters sowie während der Ausführung diverser Bewegungsaufgaben.

Mit Ausnahme der Konfiguration werden die Teilschritte mehrmals ausgeführt und die Ergebnisse erfasst.

8.1 Konfiguration



(a) Ausgewählter Arbeitsbereich (blau) sowie Erfassungsbereich der Tiefendaten für die Schräglagenkorrektur (grün) im Konfigurationsprozess.



(b) Auf dem Tisch vorhandene Refferenzpunkte (weiß) und die im Farbbild erkannten Zentren der Punkte (rot) im Konfigurationsprozess.

Abbildung 50: Konfigurationsprozess

Für die Konfiguration des Systems wird das System gestartet und aufgrund der fehlenden „configuration.xml“ -Datei wird der Konfigurationsprozess automatisch initialisiert. Die intrinsischen Kameraparameter der Kinect sind bereits erfasst und Bestandteil der automatisch eingelesenen „calib.xml“-Datei [9]. Im ersten Schritt gibt der Anwender den Arbeitsbereich (Abbildung 50a) durch Eintragen eines Rechtecks in das IR-Bild vor. Die Verwendung des IR-Bildes begründet sich mit den unterschiedlichen Aufnahmebereichen des Farb- und Tiefensensors und soll verhindern, dass Bereiche ausßerhalb des Tiefensensors ausgewählt werden. Nach Berechnen der Schräglagenkorrektur wählt der Nutzer in dem Farbbild die Refferenzpunkte aus (Abbildung 50b), die zueinander in x- und y-Richtung eine Distanz von 500 mm aufweisen und im Programm hinterlegt sind. Das System bestimmt anschließend durch den Canny-Algorithmus und der Approximation eines Kreises aus der Kontur die Zentren der Markierungen. Diese dienen nach der Umrechnung in das Kamerakoordinatensystem als Referenzpunkt. Die Konfiguration ergibt für die Transformation zwischen Kamera- und Weltkoordinatensystem nachstehende

[9] Das Programm für die Kalibrierung der Kinect speichert die intrinsischen Parameter in der kompatiblen .xml-Datei und liegt dem Datenträger dieser Arbeit bei.

Transformationsmatrix (gerundet):

$$P_W^O = \begin{pmatrix} 1.148 & 0.1027 & -128.3 \\ 0.003 & 1.221 & -75.66 \\ -5 \cdot 10^{-6} & 1 \cdot 10^{-4} & 1 \end{pmatrix}$$

Die Matrix zeigt eine translatorische Verschiebung in x-Richtung um $P_{W13}^O = -128.3$ sowie in y-Richtung um $P_{W23}^O = -75.66$. $P_{W33}^O = 1$ resultiert aus der Tatsache, dass es sich um eine homogene Transformationsvorschrift handelt.
Um die Transformationsvorschrift zwischen Welt- und Roboter-Koordinantensystem zu erhalten, fährt der Roboter vier definierte Punkte ab und platziert dort eine Kiste. Vor Lösen des Greifers an der Zielposition wird die Position des Tools im Roboter-Koordinatensystem erfasst. Anschließend fährt der Roboter ein Stück zurück und der Griffpunkt wird durch das Programm im Weltkoordinantensystem gemessen. Durch Anfahren der restlichen drei Messplätze werden die so ermittelten Referenzpunkte in beiden Koordinatensystemen für die Bestimmung der Transformationsmatrix verwendet. Während der Durchführung der Messung ist zu beobachten, dass der Roboter die Box an der zweiten und dritten Position durch einen falschen Höhenwert nicht korrekt greift. Zusätzlich verrutscht die Box um einige Millementer, da der Greifer zu hoch positioniert ist. Die Vorgabe der Messposition basiert auf einer einzigen Vorgabe an konkreten Gelenkstellungen und der anschließenden Variation der x- und y-Position für die jeweilige Messung. Der z-Wert erfährt somit keine Änderung und lässt darauf rückschließen, dass der Tisch uneben ist oder der Roboter nicht exakt orthogonal zu der Tischebene positioniert ist. Diese Annahme kann durch eine Messung des Abstandes zwischen Werkstück und Tisch für unterschiedliche Positionen gestützt werden, deren maximale Differenz bei $\Delta_{RZ} = 4\,\text{mm}$ liegt. Die Anpassung der z-Position für jede Messposition ermöglicht anschließend ein erfolgreiches Einmessen, bei dem weiterhin leichtes Verrutschen der Kiste ($\pm 1...2\,\text{mm}$) zu beobachten ist. Die Konfiguration ergibt für die Transformation zwischen Welt- und Roboter-Koordinatensystem folgende Transformationsmatrix (gerundet):

$$P_R^W = \begin{pmatrix} -1.00 & 0.006 & 330.6 \\ -0.006 & -1.00 & 950 \end{pmatrix}$$

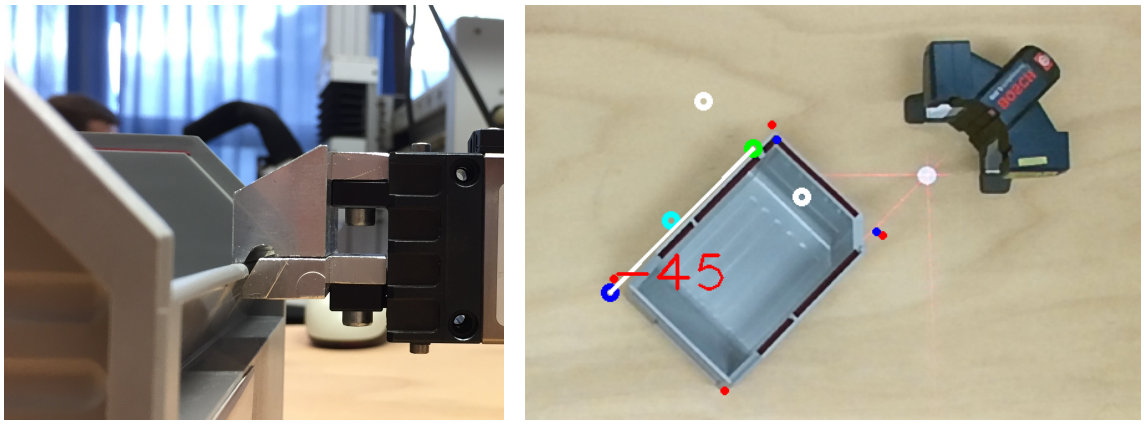
Mit den erfassten Matritzen sind alle notwendigen Vorschriften für das Greifen der Boxen vorhanden.

8.2 Greifen und Bewegen einer Box

Für die Validierung des Transportvorganges werden fünf Messungen je Box (vier Große und vier Kleine) durchgeführt. Der Vorgang wird als erfolgreich klassifiziert, wenn der Roboter die Box sicher greift und transportiert. Beschleunigung und Geschwindigkeit des Roboters betragen das Maximum, um den Messvorgang zu beschleunigen. Für den Transport der Boxen wird aufgrund der möglichen Singularitäten in den Gelenken des Roboters ein definiertes Fahrprofil erstellt, das sich aus den folgenden Schritten zusammensetzt:

1. Anfahrt der Box durch den „movej“-Befehl.
2. Greifen der Box durch das Tool.
3. Drehen der Box durch den „movej“-Befehl, um eine senkrechte Ausrichtung der Box zu erreichen.
4. Lineare Bewegung der Box zu der Zielposition durch den „movel“-Befehl.

Offset



(a) Seitenansicht während des Greifprozesses.

(b) Messaufbau für die Validierung des Bildverarbeitungsprozesses.

Abbildung 51: Greif und Validierungsvorgang.

Bereits der erste Versuch die Box aus dem Konfigurationsprozess zu greifen und zu transportieren schlägt während des Greifprozesses fehl. Abbildung 51a zeigt den Vorgang aus der Seitenperspektive und verdeutlicht, dass der Greifwinkel optisch passend zu der Box ist, jedoch ein Versatz in der y-Position zu entnehmen ist.
Um die Fehlerquelle für diesen Versatz zu finden, werden im ersten Schritt die metrischen Informationen aus dem Bildverarbeitungsprozess im Weltkoordinatensystem betrachtet. Hierzu wird für die große und die kleine Box die Breite anhand der Eckpunkte sowie der Winkel erfasst. Die Platzierung der Boxen erfolgt, mit Bezug zu den Referenzpunkten auf dem Tisch, in einem 45°-Winkel und wird am Rand des Arbeitsbereiches platziert, um die Einwirkung der Verzeichnung des Objektives zu verstärken. Der Aufbau ist der Abbildung 51b zu entnehmen, in der die kleine Box, der Laser zum einmessen, die Punkte im Farbbild (kleine Punkte), die Punkte im Weltkoordinatensystem (große Punkte, es gilt1 mm$\hat{=}$1 px), der Winkel der Box (rote Schrift in Grad) sowie der obere linke Referenzpunkt auf dem Tisch (weißer Punkt, in dem sich die Strahlen des Lasers treffen), zu erkennen sind. Als Referenzwert wird ein Winkel von 45° sowie ein Abstand zwischen den Mittelpunkten der roten Markierungen (vgl. Abbildung 51b die kleinen blauen Punkte und die großen weißen Punkte) von 111 mm (kleine Box) und 161 mm (große

Box) händisch gemessen. Die Erfassung und Mittelwertbildung von 25 Messungen je Boxgröße sind der Tabelle 12 zu etnehmen.

Boxgröße	Winkel in °	Winkeldifferenz in °	Abstand in mm	Abstandsdifferenz in mm
klein	-44,649	0,351	110,850	0,15
groß	-44,649	0,351	160,60	0,4

Tabelle 12: Durchschnittlicher Abstand zwischen den seitlichen Markierungen sowie durchschnittliche Ausrichtungswinkel der Box im Weltkoordinatensystem.

Die Ergebnisse zeigen, dass der Bildverarbeitungsprozess Ergebnisse mit einer Toleranz kleiner als einem Grad sowie kleiner als einem Millimeter liefert. Es wird daher angenommen, dass die Ursache der fehlerhaften Griffposition auf die Transformation zwischen dem Welt- und dem Roboter-Koordinatensystem zurückzuführen ist. Diese Annahme wird durch leichtes Verrutschen der Boxen während des Konfigurationsprozesses gestärkt. Eine weitere Ursache entsteht durch den nicht-orthogonal zu der Tischebene platzierten Roboter, da die Abweichung in x- und y-Positionen mit der Entfernung zum Roboter-Ursprung zunehmen und Abweichungen gegenüber dem Weltkoordinatensystem aufweisen. Bei der Transformation zwischen Welt- und Roboterkoordinatensystem wird die z-Position nicht berücksichtigt. Zusätzlich weist der Roboter in der absoluten Position des Tools eine Abweichung auf, deren Betrag weder durch das Datenblatt, noch durch den direkten Kontakt zu Universal Robots in Erfahrung gebracht werden kann. Ein gemeinsames Einwirken aller drei Ursachen ist die wahrscheinlichste Annahme und soll durch einen manuellen Offset korrigiert werden. Gegenüber dem Versuch den Konfigurationsprozess genauer zu gestalten und eine Kalibrierung des Roboters durchzuführen, bietet der manuelle Offset mehr Komfort für den Anwender und ist mit weniger Aufwand umzusetzen. Darüber hinaus ist eine Transformation unter Beachtung der Tiefe nicht ohne eine Stütze für die Fixierung der Box in der Höhe zu realisieren. Die Genauigkeit der Ortsinformation einer Box wird nicht beeinträchtigt und würde eine spätere Kollisionsvermeidung im Ergebnis nicht mindern. Die Durchführung diverser Transporte ergibt einen maximalen Offset von 10 mm, der abhängig von dem Griffwinkel der Box ist. Die Abweichung beträgt 2 mm für einen Winkel von -90° sowie 10 mm für die Winkel 0° und -180°. Die Berechnung des Offsets $v_O(\alpha)$ in Abhängigkeit zu dem Griffwinkel α(in radiant) erfolgt nach:

$$v_O(\alpha_B) = 2\,\text{mm} + 10\,\text{mm} \cdot \frac{\left|\alpha \pm \frac{\pi}{2}\right|}{\frac{\pi}{2}}$$

Das Vorzeichen im Betrag entspricht dabei dem entgegengesetzten Vorzeichen aus α. Beide Offsets (2 mm und 10 mm) sind auf die GUI geführt und durch den Anwender parametrierbar. Sie ermöglichen im Anschluss eine Validierung der Kisten.

Validierung mit dem Offset

Box Nr.	1	2	3	4	5	6	7	8
Größe	Klein	Klein	Klein	Klein	Groß	Groß	Groß	Groß
Erfolgreich	5	5	5	5	5	5	5	5

Tabelle 13: Auflistung der erfolgreichen Transportbewegungen aller acht Boxen, sowie deren Größe.

Unterschiedliche Positionen und Ausrichtungen der acht Boxen werden durch eine Bildaufnahme[10] erfasst und im Anschluss der Greifvorgang initialisiert. Die Ergebnisse sind der Tabelle 13 zu entnehmen und zeigen, dass der Greifvorgang zu 100% erfolgreich ist.

[10] Die Aufnahmen sind dem Datenträger dieser Arbeit zu entnehmen

8.3 Kollisionserkennung

Die Kollisionserkennung findet unter den in Kapitel 6.5.4 genannten Einschräkungen statt und wird weiterhin nur teilweise validiert. Dazu werden über einem Zeitraum von 30 min vier Boxen in den Arbeitsbereich platziert und bewegt, um im Anschluss Kollisionen zu provozieren. Jeder Versuch eine Box zu bewegen wird unabhängig von einer existierenden Kollision als „Kollision“ oder „Kollisionsfrei“ bewertet und die daraus resultirerende „false-negative rate“ und „false-positive rate“ bestimmt. Die Roboter-Geschwindigkeit ist dabei von der Existenz eines menschlichen Armes im Arbeitsbereich abhängig und beträgt maximal $1\frac{\mathrm{m}}{\mathrm{s}}$ und minimal $0,4\frac{\mathrm{m}}{\mathrm{s}}$. Gleichzeitig wird die Breite der Kontur anhand der Geschwindigkeit des Roboters definiert und beträgt 55 px für $v_R = 1\frac{\mathrm{m}}{\mathrm{s}}$ sowie 35 px für $v_R = 0,4\frac{\mathrm{m}}{\mathrm{s}}$. Der Roboter wird durch den „Free-Drive“-Modus in die Ausgangsposition gebracht und bewegt Boxen in dem Arbeitsfeld von ihrer variierenden Ausgangposition in die Roboter-Position aus der Abbildung 39. Der Mensch steht während der gesamten Phase links neben dem Roboter und wirkt durch das Bewegen der Boxen in den Arbeitsablauf ein.
Während des Messzeitraumes wurden 157 Transportbewegungen ausgeführt, deren Bewegung wie folgt klassifiziert werden:

	Es existiert eine Kollision	Es existiert keine Kollision
Als Kollision klassifiziert	80	3
Als Kollisionsfrei klassifiziert	1	73

Tabelle 14: Ergebnisse der Kollisionserkennung, aus der Beobachtung von 157 Transportaufgaben.

Aus den Ergebnissen der Tabelle 14 resultiert eine FRR von:

$$FRR = \frac{1}{81} = 0,012$$

sowie eine FAR von

$$FAR = \frac{3}{76} = 0,039$$

Während der Durchführung der Messung entstand eine Situation, in der das System eine existierende Kollision nicht erkannt hat. In dieser wurde der Mensch aufgrund seiner Nähe zu dem Sicherheitsbereich (kleine geclusterte Rechtecke) und zu der Nähe des Roboters, als Roboter klassifiziert. Erst als der Roboter sich von dem Menschen entfernt hat, wurde der Arm korrekt klassifiziert und der Bewegungsablauf unterbrochen. Darüber hinaus wurden während der Durchführung drei falsche Kollisionen erfasst, die auf den Zusammenschluss der Box-Konturen, den Roboter-Konturen und der menschlichen Kontur zurückzuführen sind. In den genannten Situationen hätte der Roboter die Boxen so verschoben, dass diese mit dem Arm kollidiert wären. Diese Fälle werden aufgrund der Distanz zwischen dem Roboter und dem Arm als falsche Interpretation gewertet.

8.4 Performance

Die Erfassung der Performance des umgesetzten Systems basiert auf der Messung der Frame-Rate (Frames per second FPS) im System, da diese durch die Bildwiederholrate der Kinect die Grenze des Systems definiert. Das verwendete Computersystem besteht aus einer Intel i7 -6700K CPU, 8Gb Arbeitsspeicher, einem 64-Bit basierendem Windows 10 Betriebssystem und einer Nvidia GeForce GT730. Für die Gegenüberstellung der Laufzeit werden acht Situationen über einen Zeitraum von je einer Minute betrachtet. Die erfassten Werte der FPS werden anschließend aneinander gehängt und sind der Abbildung 52 zu entnehmen. Bei dem in der Abbildung 52 gezeigten FPS-Wert handelt es sich um eine Mittelwertbildung, da die unterschiedlichen Laufzeiten in einer Sekunde eine unterschiedliche Anzahl an Messungen erlauben.

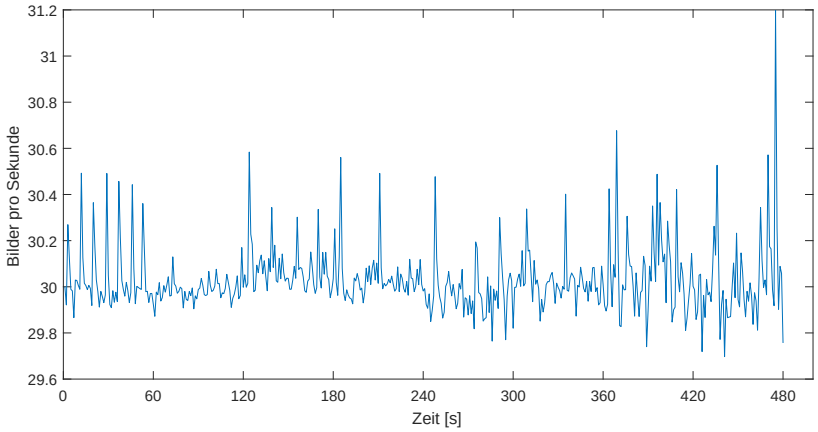


Abbildung 52: FPS des umgesetzten Systems unter diversen Belastungen.

Zeit [s]	Visu-alisierung	Erfassung im Tiefenbild	Erfassung im Farbbild	Bewegung	Arme im Arbeits-bereich	Transport	Kollision
0-60	x	-	-	-	-	-	-
61-120	x	x	-	-	-	-	-
121-180	x	x	-	x	-	-	-
181-240	x	x	x	-	-	-	-
241-300	x	x	x	x	-	-	-
301-360	x	x	x	-	-	x	-
361-420	x	x	x	x	x	-	-
421-480	x	x	x	x	x	x	x

Tabelle 15: Zuordnung der Belastungssituationen und der Zeitmessung aus Abbildung 52.

Die Messung der FPS in Abbildung 52 basiert auf den Systembelastungen der Tabelle 15 und lässt eine grobe Einschätzung der Performance zu. Die Spalte „Visualisierung“ definiert die Darstellung des Roboters und des Menschen durch Zylinder, während die Spalte „Bewegung“ eine permanente Bewegung der Boxen beschreibt. Die Belastung „Transport“ erfolgt durch eine permanente Roboterbewegung für den Transport der Boxen. Die Messphase mit provozierten Kollisionen ist der Spalte „Kollision“ zu entnehmen.
Die Messung aus Abbildung 52 zeigt, dass keiner der umgesetzten Klassen einen sichtbaren Einbruch der Bildwiederholrate verursacht und die maximal zu erreichende Systemperformance erreicht wird. Die minimale Reaktionszeit des Systems beträgt somit $t_{Rmin} = \frac{1\,\mathrm{s}}{29{,}7\,\frac{\mathrm{B}}{\mathrm{s}}} = 33{,}337\,\mathrm{ms}$.

9 Ergebnis

Innerhalb des Bearbeitungszeitraumes konnte ein funktionierendes, schnelles und visualisiertes System zur Erkennung und Klassifizierung eines Roboters, eines Menschen und einer Kollision umgesetzt werden. Die Entscheidung wurde zugunsten einer Modul basierten C++ Anwendung unter Verwendung der OGRE3D-Render-Engine und dem OpenCV- sowie QT-Framwork getroffen. Die OGRE3D-Engine bietet durch eine ausführliche Dokumentation, einer großen Community sowie einer einfachen Schnittstelle zu dem QT-Framework Vorteile und konnte sich gegenüber der Irlicht OSG und QT3D-Engine durchsetzen. Die Engine ermöglicht eine einfache Visualisierung der Punktwolke und ist nahtlos in die GUI intigriert.
Für die Berichtigung der Schräglagenkorrektur wurde ein neuer Ansatz vorgestellt, der auf Grundlage der Standardabweichung in einem Tiefenbild die Schräglage der Kamera für die Bildverarbeitung korrigieren kann. Dieser Ansatz ist gegenüber dem Verfahren zur Approximation der Ebenengleichung langsamer und wies in der Validierung einen geringen Versatz auf, weshalb der Ansatz über die Ebenengleichung Verwendung fand.
Die Betrachtung der Kinect als Bildquelle hat gezeigt, dass eine Kantenerkennung in dem originalem Tiefenbild aufgrund der Mittelwertbildung benachbarter Pixel deutlich erschwert ist. Weiterhin kann nur eine geringe Schwellwertgrenze in dem Tiefenbild verwendet werden, da relevante Kanten der Transportboxen früh entfernt werden. Der Versuch die Ortsinformation aus dem Coordinate-Mapper des Kinect-SDKs auf eine Matrix abzubilden, minderte die Möglichkeit Kanten zu erkennen und war nicht zielführend, weshalb der Ansatz verworfen wurde.
Aus den Untersuchungen heraus wurde sich dazu entschieden das Tiefenbild für die Vorsegmentierung zu verwenden, indem die Hu-Momente der Konturen als Klassifikationskriterium verwendet wurden. Im Vergleich zu dem ebenfalls umgesetzten Ansatz der Kantenerkennung durch die Hough-Transformation, konnte in dem Ansatz mit den Hu-Momenten eine höhere Zuverlässigkeit in der Klassifizierung, eine deutlich bessere Performance und eine geringere Genauigkeit erreicht werden.
Für die Erfassung der unbunten Transportboxen im Farbbild wurde durch eine Ergänzung der Boxen mit weinrotem Vlies ein geeignetes Merkmal für die Erfassung der Griffposition gefunden, da die Materialeigenschaften eine diffuse und gute Reflektion im Tiefenbild sowie eine klare Erkennung im HSV-Farbraum aufweisen.
Die Umsetzung der Kreuzkorrelations- und der Differentialbildung im HSV-Farbraum ergab, dass nur geringe Unterschiede in der Performance festzustellen sind. Die Kreuzkorrelation ist in der Erkennung der roten Markierungen sowohl robuster als auch genauer und findet daher Anwendung in der finalen Applikation. Die Erkennung der roten Markierungen ist skalierbar umgesetzt, weshalb der Anwender die Anzahl der Messpunkte bestimmen kann.
Für die Erkennung der menschlichen Körperteile konnte auf Grundlage der konvexen Hülle das Clustern der Konturpunkte in dem Arbeitsbereich als geeignete Methode verifiziert werden, um einen Arm in mehrere Abschnitte zu zerlegen. Die Betrachtung der Ortsinformation einer in OpenCV erfassten Kontur wies keine eindeutigen Informationen über die Position der Hand auf und wurde verworfen. Dem gegenüber kann der erste Cluster einer Kontur als Hand interpretiert werden und als Schnittstelle zwischen Roboter und Mensch dienen.
Eine Gegenüberstellung des K-Means- und des EM-Algorithmus hat gezeigt, dass der K-Means-Algorithmus gegenüber dem EM-Algorithmus deutlich performanter ist. Eine Aussage über die Genauigkeit ist nicht möglich, da die in OpenCV implementierte Funktion des EM-Algorithmus auf den Initialisierungswerten des K-Means-Clustering basiert. Aufgrund der besseren Performance wurde der K-Means Algorithmus in der Software angewendet.
Für die Klassifizierung der Konturen von Mensch und Roboters konnte ermittelt werden, dass die Approximation eines Rechteckes auf Grundlage der geclusterten Konturpunkte mit einer anschließenden Prüfung der Schnittmenge zwischen dem vereinfachten Robotermodell eine geeignete Methode ist. Diese Methode ist aufgrund der Trennung von Konturpunkten außerhalb des Arbeitsbereiches mit Einschränkungen verbunden, die ein kontrolliertes Einfahren des Roboters in den Arbeitsbereich voraussetzen. Zusätzlich führen Arbeiten des Roboters oberhalb des Arbeitsbereiches zu einer Fehlinterpretation.
Weitere Untersuchungen haben gezeigt, dass die Betrachtung der klassifizierten Konturen als Kriteri-

um einer Kollision verwendet werden kann, um den Roboter vor der Kollision anzuhalten. Für eine frühzeitige Erkennung ist die Breite der Kontur in Abhängigkeit von der Roboter-Geschwindigkeit zu vergrößern. Die Erkennung ist aufgrund der Systemkalibrierung im zweidimensionalen Raum auf die Höhe der Arbeitsebene beschränkt.
Darüber hinaus bietet die erstellte Applikation die Möglichkeit diverse Systemparameter der Algorithmen durch den Anwender zu variieren. Die Visualisierung des Roboters und des Menschen in der GUI erfolgt wahlweise durch Zylinder oder farblich markierte Datenpunkte der Punktwolke.
Für die Kalibrierung der Transformationsvorschriften zwischen den Koordinantensystemen konnte eine automatisierte und in die Software integrierte Kalibrierung umgesetzt werden.
Diverse Validierungen haben außerdem ergeben, dass der umgesetzte Zustandsautomat für den Roboter in der Lage ist Transportboxen zu greifen und zu bewegen. Der UR5-Roboter besitzt dabei eine erhöhte Ungenauigkeit, weshalb die Greifpositionen nur mithilfe eines Offsets erfolgreich erreicht werden können.

10 Ausblick

Das in dieser Thesis umgesetzte System repräsentiert die Grundstruktur für eine Mensch-Roboter-Kollaboration. Das umgesetzte Programm kann, dank seiner modular gestalteten Struktur, einfach ergänzt werden. Bereits in der vorliegenden Fassung kann eine kollisionsfreie Trajektorienplanung mit Bezug zu der Arbeitsfläche implementiert werden. Eine Ergänzung des Kalibriervorganges um die dritte Dimension erlaubt darüber hinaus eine kollisionsfreie Trajektorienplanung auf jeder Ebene, die im Erfassungsbereich des Sensors liegt.
Die Genauigkeit der Kollisionserkennung kann durch einen pixelweisen Vergleich mit dem Roboter gesteigert werden. Wird zusätzlich ein Kalman-Fillter verwendet, können Kollisionen abhängig von der Geschwindigkeit und Bewegungsrichtung der Person und des Roboters erkannt werden. Diese erlauben eine höhere Genauigkeit und höhere Robotergeschwindigkeiten, sofern sie zuverlässig sind. Eine Alternative ist im Ansatz „Maschinelles lernen“ zu finden. So kann geprüft werden, inwiefern ein System lernen kann Kollisionen zu erkennen und rechtzeitig zu verhindern. Die in dieser Arbeit aufgezeigten Abweichungen durch die Kinect können mithilfe weiterer Sensoren, die seitlich auf das Arbeitsfeld gerichtet sind, behoben werden und die „mixed Pixel“ der bestehenden Kinect eliminieren. Gleichzeitig stellt die Ergänzung um weitere Kameras eine dichtere Informationsquelle bereit. Die Befestigung der Kinect an das letzte Gelenk des Roboters kann darüber hinaus den näheren Arbeitsbereich scannen und dadurch eine noch genauere Abbildung der Umgebung erstellen. Dieser Ansatz geht dafür mit Einschränkungen im Bewegungsfreiraum des Roboters einher. Für einen Abgleich der zusätzlichen Kameras stehen grundlegende Funktionen in dem OpenCV-Framework bereit. Die Gewinnung der zusätzlichen Informationen kann außerdem die Bildverarbeitung im Farbbild ersetzen und so weitere Ressourcen freigeben.
Die Kinect verfügt über bisher ungenutzte Ressourcen wie das Mikrofon und die im Kinect-SDK enthaltene Spracherkennung. Mit diesen Ressourcen wäre eine Kontrolle der Arbeitsaufträge realisierbar, wodurch die Interaktion zwischen Roboter und Mensch um den Aspekt der Sprache erweitert wird.
Im Kontext der Industrie 4.0 ergeben sich weitere interessante Gebiete, die aktuelle Problemstellungen aufgreifen. So kann ein zweiter identischer Arbeitsplatz verwendet werden, um eine Kollaboration zwischen zwei Systemen zu realisieren, die eine gemeinsame Datenbank (Cloud) als Grundlage für die Arbeitsabläufe verwenden. Des Weiteren besteht, durch das im Jahr 2017 eröffnete Automatisierungslabor an der HAW, die Möglichkeit das Cloud-System um Informationen aus einer Produktionsstraße zu ergänzen und den Menschen durch den Roboter in den Prozess einzubinden. Dabei könnte der Roboter die Sicherheitsgrenzen der Fertigungsstraße umgehen und so eine Schnittstelle zwischen dem Menschen und der Produktionsstraße ermöglichen.

Literatur

[Alf] Alff, Mischa A.: *Finding the Right Graphics Library for Your Projects*. https://destrock.com/b/s/finding-the-right-graphics-library/, Abruf: 2017-08-02

[AV07] *Kapitel* k-means++: the advantages of careful seeding. In: Arthur, David ; Vassilvitskii, Sergei: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics Philadelphia, 2007, S. 1027–1035

[BB06] Burger, Wilhelm ; Burge, Mark J.: *Digitale Bildverarbeitung, Eine Einführung mit Java und ImageJ*. Springer, 2006. http://dx.doi.org/13978-3-540-30940-6. http://dx.doi.org/13978-3-540-30940-6

[Ber11] Bernshausen, Dipl.-Ing. J.: *Integration von 3D- Time-of-Flight -Kameras in Applikationen zur sicheren Steuerung autonomer Roboter*, Universität Siegen, Diss., 2011

[BNC+16] Beyl, Tim ; Nicolai, Philip ; Comparetti, Mirko D. ; Raczkowsky, Jörg ; Momi, Elena ; Wörn, Heinz: Time-of-flight-assisted Kinect camera-based people detection for intuitive human robot cooperation in the surgical operating room. In: *International Journal of Computer Assisted Radiology and Surgery* 11 (2016), Juli, Nr. 7, 1329. http://dx.doi.org/10.1007/s11548-015-1318-7. – DOI 10.1007/s11548–015–1318–7

[Bro02] Brocke, Martin: *Statistische Ereignisdetektion in Bildfolgen*, Ruprecht-Karls-Universität Heidelberg, Diss., 2002

[Can86] Canny, J.: A Computational Approach to Edge Detection. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8 (1986), Nov, Nr. 6, S. 679–698. http://dx.doi.org/10.1109/TPAMI.1986.4767851. – DOI 10.1109/TPAMI.1986.4767851. – ISSN 0162–8828

[CAP+12] Culjak, Ivan ; Abram, David ; Pribanic, Tomislav ; Dzapo, Hrvoje ; Cifrek, Mario: *A brief introduction to OpenCV*. May 2012

[CGMS15] Corti, Andrea ; Giancola, Silvio ; Mainetti, Giacomo ; Sala, Remo: *A metrological characterization of the Kinect V2 time-of-flight camera*. 10 2015

[DK16] Dahlkemper, Prof. Dr.-Ing. ; Kölzer, Prof. Dr.-Ing.: *Vorlesungsskript Angewandte industrielle Bildverarbeitung*. 2016

[DKML16] Drath, Rainer ; Krüger, Martin W. ; Matthias, Bjoern ; Listmann, Kim D.: Das Internet der Dinge, Dienste und Menschen – Auswirkungen von Industrie 4.0 auf den Menschen am Beispiel kollaborativer Roboter. In: *ResearchGate* (2016)

[dMG+17] de Gea Fernández, José ; Mronga, Dennis ; Günther, Martin ; Knobloch, Tobias ; Wirkus, Malte ; Schröer, Martin ; Trampler, Mathias ; Stiene, Stefan ; Kirchner, Elsa ; Bargsten, Vinzenz ; Bänziger, Timo ; Teiwes, Johannes ; Krüger, Thomas ; Kirchner, Frank: Multimodal sensor-based whole-body control for human-robot collaboration in industrial settings. In: *Robotics and Autonomous Systems* 94 (2017), 102 - 119. http://dx.doi.org/http://dx.doi.org/10.1016/j.robot.2017.04.007. – DOI http://dx.doi.org/10.1016/j.robot.2017.04.007. – ISSN 0921–8890

[Ebe03] Ebert, Dirk: *Bildbasierte Erzeugung kollisionsfreier Transferbewegungen für Industrieroboter*, Technische Universität Kaiserslautern, Diss., 2003

[FP11] Frati, Valentino ; Prattichizzo, Domenico: *Using Kinect for hand tracking and rendering in wearable haptics*. IEEE World Haptics Conference 2011, 2011

[Gec11] Gecks, Thorsten: *Sensorbasierte, echtzeitfähige Online-Bahnplanung für die Mensch-Roboter-Koexistenz*, Universität Bayreuth, Diss., 2011

[GJK00] Galambos, C. ; J.Matas ; Kittler, J.: Progressive Probabilistic Hough Transform for line detection. In: *Computer Vision and Image Understanding* 78 (2000), S. 119–137

[Goe17] Goetze, Dr. B.: Methode der kleinsten Quadrate zur Approximation von Punktmengen durch Ebenen / Gesellschaft zur Förderung angewandter Informatik e.V. Forschungsbereich Graphing. 2017 (V. 3.0). – Forschungsbericht

[HLCH12] Hansard, Miles ; Lee, Seungkyu ; Choi, Ouk ; Horaud, Radu P.: *Time-of-Flight Cameras*. Springer London, 2012 `http://www.ebook.de/de/product/20528159/miles_hansard_seungkyu_lee_ouk_choi_radu_patrice_horaud_time_of_flight_cameras.html`. – ISBN 978–1–4471–4658–2

[HNCM05] Hansen, Pierre ; NgaiBernard, Eric ; Cheung, K. ; Mladenovic, Nenad: Analysis of Global k-Means, an Incremental Heuristic for Minimum Sum-of-Squares Clustering. In: *Journal of Classification* 22 (2005), S. 287–310. `http://dx.doi.org/10.1007/s00357-005-0018-3`. – DOI 10.1007/s00357–005–0018–3

[Hoc16] Hoch, Frank: *Konzeption und Entwicklung einer Steuerungsarchitektur für die Roboter-Mensch- Interaktion*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2016

[Hof06] Hofemann, Nils: *Videobasierte Handlungserkennung für die natürliche Mensch-Maschine-Interaktion*, Universität Bielefeld, Diss., 2006

[Hu62] Hu, Ming-Kuei: Visual pattern recognition by moment invariants. In: *IRE Transactions on Information Theory* 8 (1962), February, Nr. 2, S. 179–187. `http://dx.doi.org/10.1109/TIT.1962.1057692`. – DOI 10.1109/TIT.1962.1057692. – ISSN 0096–1000

[Jäh12] Jähne, Bernd: *Digitale Bildverarbeitung*. Springer Berlin Heidelberg, 2012 `http://www.ebook.de/de/product/20512144/bernd_jaehne_digitale_bildverarbeitung.html`. – ISBN 978–3–642–04952–1

[JS14] John Sell, Patrick O.: *THE XBOX ONE SYSTEM ON A CHIP AND KINECT SENSOR*. 03 2014

[Kla10] Klawonn, Frank: Grundlagen zweidimensionaler Darstellungen. Version: Januar 2010. `http://dx.doi.org/10.1007/978-3-8348-9679-7_2`. In: *Grundkurs Computergrafik mit Java*. Springer, Januar 2010. – DOI 10.1007/978–3–8348–9679–7_2. – ISBN 978–3–8348–1223–0

[Kle16] Kleborn, Stephan: *Evaluierung einer Microsoft Kinect v2 zur gesten- und sprachbasierten Steuerung eines Roboterarmes*. August 2016

[MDHMN+16] Markis, DI A. ; DI Harald Montenegro, MSc ; Neuhold, Ing. M. ; Oberweger, Ing. A. ; Schlosser, DI C. ; Schwald, DI C. ; Sihn, Prof. Dr.-Ing. W. ; Fabian Ranz, M.Sc. ; Edtmayr, DI T. ; Hold, Dipl.-Wirtsch.-Ing. P. ; Reisinger, DI G.: Sicherheit in der Mensch-Roboter- Kollaboration / TÜV AUSTRIA Holding AG and Fraunhofer Austria Research GmbH. 2016. – Forschungsbericht

[Ott11] Otto, Markus: Lineare Algebra. Version: jan 2011. `http://dx.doi.org/10.1007/978-3-8274-2456-3_2`. In: *Rechenmethoden für Studierende der Physik im ersten Jahr*. Springer, jan 2011. – DOI 10.1007/978–3–8274–2456–3_2. – ISBN 978–3–8274–2455–6

[Que] Quesenberry, Andrew: *Ogre3D vs. OpenSceneGraph vs. Irrlicht vs. Panda3D vs. Monogame vs.OpenGL Free for all.* http://qbit-ent.blogspot.de/2015/03/ogre3d-vs-openscenegraph-vs-irrlicht-vs.html, Abruf: 2017-07-31

[RN12] Russel, Stuart ; Norvig, Peter: *Künstliche Intelligenz.* Bd. 3. Auflage. Pearson Studium, 2012 http://www.ebook.de/de/product/19148036/peter_norvig_stuart_russell_kuenstliche_intelligenz.html. – ISBN 978–3–86894–098–5

[Rob05] Robots, Universal: *UR5 Technische Daten*, 2005. https://www.universal-robots.com/media/1514579/101081_199917_ur5_technical_details_web_a4_art03_rls_de.pdf, Abruf: 2017-08-02

[RRA10] Rocha, Rafaela V. (Hrsg.) ; Rocha, Rodrigo V. (Hrsg.) ; Araújo, Regina B. (Hrsg.) ; Federal University of São Carlos Department of Computer Science, Brazil (Veranst.): *Selecting the Best Open Source 3D Games Engines.* 2010

[Sch05] Schürmann, Tim: 3D-Engine Ogre Künstlicher Horizont. In: *Linux-Magazin* 11 (2005). http://www.linux-magazin.de/Ausgaben/2005/11/Kuenstlicher-Horizont

[Sch13] Schwarzer, Roland: *Markerlose Oberflächenkontruktion mithilfe von Mehrkamerasystemen*, Hochschule für Technik und Wissenschaft Dresden, Diplomarbeit, 2013

[scr] scrawl: *Ogre3D vs. OpenSceneGraph.* http://scrawl.bplaced.net/blog/?p=442, Abruf: 2017-07-31

[Sko] Skovsgaard, Lars: *Handling Singularity.* http://www.zacobria.com/universal-robots-zacobria-forum-hints-tips-how-to/singularity/

[tea] team, OpenCV: *About.* http://opencv.org/about.html, Abruf: 2017-08-02

[Wela] Welbourne, Edward: *Qt History.* https://wiki.qt.io/Qt_History, Abruf: 2017-04-23

[Welb] Welbourne, Edward: *Using 3D engines with Qt.* http://wiki.qt.io/Using_3D_engines_with_Qt, Abruf: 2017-05-03

[wik] wikipedia: *HSV-Farbraum.* https://de.wikipedia.org/wiki/HSV-Farbraum, Abruf: 2017-08-01

[WS16] Wasenmueller, Oliver ; Stricker, Didier: *Comparison of Kinect v1 and v2 Depth Images in Terms of Accuracy and Precision.* 2016

11 Anhang

Der Anhang dieser Arbeit befindet sich auf einem Datenträger, der bei Herrn Prof. Dr.-Ing. Jochen Maaß einsehbar ist.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____