HAW HAMBURG

BACHELOR THESIS
Adem Can Agdas

Fusion von Kamera- und Punktwolkendaten in Simulation und Realwelt

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Adem Can Agdas

Fusion von Kamera- und Punktwolkendaten in Simulation und Realwelt

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Tim Tiedemann
Zweitgutachter: Prof. Dr. Peer Stelldinger

Eingereicht am: 16. November 2022

Adem Can Agdas

Thema der Arbeit

Fusion von Kamera- und Punktwolkendaten in Simulation und Realwelt

Stichworte

Fusion, Kamera, Lidar

Kurzzusammenfassung

Diese Arbeit thematisiert die Fusion von Kameradaten und 3D-Punktwolkendaten, indem den einzelnen 3D-Punkten in einer Punktwolke Farben basierend auf den Kameradaten zugeordnet werden. Mithilfe der intrinsischen und extrinsischen Parameter der Kamera werden die einzelnen 3D-Punkte der Punktwolke in das Kamerabild zurückprojiziert, wodurch dann eine Farbe für den jeweiligen 3D-Punkt aus dem Kamerabild abgelesen wird. In dieser Arbeit wurde ein Toolset unter ROS entwickelt, welches diesen Vorgang implementiert. Im Rahmen dessen wurde unter anderem eine Simulation in der Unity3D-Engine erstellt, die jeweils mehrere Kameras und einen Lidar simuliert. Das Toolset wurde jeweils auf die Simulation und Echtweltgeräte angewendet und die Ergebnisse evaluiert.

Adem Can Agdas

Title of Thesis

Camera and Point Cloud Data Fusion in Simulation and Real World

Keywords

Fusion, Camera, Lidar

Abstract

This work addresses the fusion of camera data with a 3D point cloud by assigning colors to the individual 3D points in the point cloud based on the camera data. With the intrinsic and extrinsic properties of the camera known, individual 3D points of the point cloud can be re-projected onto the image, which makes it possible to read a color from that image that is associated with that particular 3D point. In this work a ROS toolset was developed that implements this procedure as well as a Unity3D project that can simulate multiple cameras and a lidar. Subsequently, the toolset was applied to the simulation and real world devices and the results evaluated.

Inhaltsverzeichnis

Abbildungsverzeichnis **vi**

1 Einleitung **1**
1.1 Ziele einer Fusion . . . 2
1.2 Sensordatenverarbeitung . . . 3
1.2.1 Messen . . . 3
1.2.2 Wahrnehmen . . . 4
1.2.3 Fehler . . . 5
1.3 Fusion von Kamera und Punktwolke . . . 6

2 Theorie **7**
2.1 Kameramodell . . . 7
2.1.1 Lochkameramodell . . . 7
2.1.2 Physische Kameras . . . 11
2.2 Physische Kameramatrix . . . 12
2.2.1 Affine Störquellen . . . 12
2.2.2 Nicht affine Störquellen . . . 15
2.2.3 Radiale Verzeichnung . . . 15
2.2.4 Tangentiale Verzeichnung . . . 19
2.2.5 Intrinsische Parameter . . . 21
2.3 Abbilden von 3D-Punkten in das Kamerabild . . . 22
2.3.1 Extrinsische Parameter . . . 23
2.4 Zusammenfassung . . . 25

3 Umsetzung **26**
3.1 Versuchsaufbau . . . 26
3.1.1 Realwelt . . . 27
3.1.2 Simulation . . . 29

3.2 Berechnung der Parameter 32
3.2.1 Intrinsische Parameter 32
3.2.2 Extrinsische Parameter 33
3.3 Systemaufbau 35

4 Evaluierung der Ergebnisse **38**
4.1 Evaluierung der simulierten und realen Geräte 38
4.1.1 Simulation der Kameras 38
4.1.2 Simulation des Lidars 40
4.2 Evaluierung der intrinsischen Kalibrierung 44
4.3 Evaluierung der extrinsischen Kalibrierung 45
4.4 Evaluierung der Fusion 48

5 Bewertung und Ausblick **57**
5.1 Relevante Arbeiten und mögliche Erweiterungen 57

6 Schlusswort **60**

Literaturverzeichnis **61**
Selbstständigkeitserklärung 64

Abbildungsverzeichnis

1.1 Prozess einer Sensordatenverarbeitung. Quelle: (Winner u. a. (2015), S. 443, Abbildung 24.1) . . . 3

2.1 Lochkameramodell. Quelle: (Kaehler und Bradski (2016) S. 639, Abbildung 18.1) . . . 8
2.2 Lochkameramodell mit Bildebene vor der Lochblende. Quelle: (Kaehler und Bradski (2016), S. 640, Abbildung 18.2) . . . 9
2.3 Rechteck-Eigenschaften der Bildsensorelemente einer physischen Kamera. Quelle: (Förstner und Wrobel (2016), S. 471, Abbildung 12.9) . . . 13
2.4 Kameratransformation in OpenCV. Quelle: (OpenCVDocs) . . . 14
2.5 Verzeichnung durch eine Linse Quelle: (Kaehler und Bradski (2016) S. 645, Abbildung 18.3) . . . 16
2.6 Verschiedene Arten von Verzeichnung Quelle: (OpenCVDocs) . . . 16
2.7 Undistortion Map für radiale Verzeichnung. Die Pfeile zeigen die Verschiebung aufgrund der Verzeichnung an. Quelle: (Kaehler und Bradski (2016), S. 646, Abbildung 18.4) . . . 18
2.8 Anwendung einer Undistortion Map. Da es sein kann, dass ein Pixel auf eine nicht ganzzahlige Koordinate abgebildet wird, wird der Pixel mit anteilhafter Intensität auf umliegende Pixel verteilt. Quelle: (Kaehler und Bradski (2016), S. 679, Abbildung 18.18) . . . 19
2.9 Tangentiale Verzeichnung. Quelle: (Kaehler und Bradski (2016), S. 647, Abbildung 18.5) . . . 20
2.10 Undistortion Map für radiale und tangentiale und Verzeichnung. Quelle: (Kaehler und Bradski (2016)), S. 648, Abbildung 18.6 . . . 21
2.11 Zusammenhang zwischen Welt-, Kamera- und Bildkoordinatensystem. Quelle: (Förstner und Wrobel (2016), S. 462, Abbildung 12.5) . . . 22
2.12 Umwandeln der Objektkoordinaten in Kamerakoordinaten. Quelle: (Kaehler und Bradski (2016), S. 650, Abbildung 18.8) . . . 23

3.1 Versuchsaufbau in Realwelt. 28
3.2 Visualisierung der Sensordaten. Oben links: Basler Kamera. Unten links: Allied Vision Kamera. Rechts: Blickfeld Lidar. Zu beachten ist, dass die Scan-Laser des Blickfeld Lidars im Bild der Allied Vision Kamera sichtbar sind, da diese in einem Spektrum sind, die von der Allied Vision Kamera detektiert werden. 29
3.3 Simulierte Szene in Unity. 30
3.4 Simulation eines Lidars in Unity durch Raycasts. 30
3.5 Visualisierung der simulierten Sensoren. Es wurde hier nur das Schachbrett modelliert ohne Hintergrundobjekte oder ähnliches. 31
3.6 Perspective N-Point (PNP) Problem Konzept. Quelle: (Kaehler und Bradski (2016), S. 701, Abbildung 19.4) 34
3.7 Systemdiagramm der Fusionsarchitektur 36

4.1 Vergleich der simulierten Kamera und der realen Kamera. 39
4.2 Bestätigung des Abstands von der Kamera zum Schachbrett durch einen Entfernungsmesser. 40
4.3 Messung des Schachbretts mit dem Blickfeld Lidar. 41
4.4 Vergleich des simulierten Lidars mit dem Blickfeld Lidar. Die rot gezeigten Punkte stammen vom simulierten Lidar und die farbigen vom realen Blickfeld Lidar 42
4.5 Verzerrung des Lidars bei hoher Auflösung: Links und Mitte: Messung des Lidars von zwei Personen, die stillstehen. Rechts: Messung des Lidars von zwei Personen, die sich während der Messung nach vorne bewegen 43
4.6 Vergleich der Abbildung von 3D-Punkten auf das Kamerabild mit guten intrinsischen Parametern (links) und schlechten (rechts) 44
4.7 Optimale Ausrichtungen des Schachbretts für eine gute Kamerakalibrierung 45
4.8 Annäherung an die extrinsischen Parameter durch das Minimieren einer Kostenfunktion. Das globale Minimum repräsentiert hierbei die tatsächlichen extrinsischen Parameter. Durch die vorher genannten Fehlerquellen kann es jedoch sein, dass das Minimum von den tatsächlichen extrinsischen Parametern abweicht. Quelle: (Bouain u. a. (2020), Abbildung 2) 47
4.9 Ergebnis der Fusion: Lidar-Punktwolke und Basler Kamera 48
4.10 Ergebnis der Fusion: Lidar-Punktwolke und Allied Vision Kamera 48
4.11 Ergebnis der Fusion: Simulierte Kamera und simulierter Lidar 49
4.12 Versuchsaufbau mit einer Person, die ein Schachbrett hält 50

4.13 Ergebnis der Fusion mit der Basler Kamera und der Lidar-Punktwolke im alternativen Versuchsaufbau 50
4.14 Ergebnis der Fusion mit der Basler Kamera und der Lidar-Punktwolke im alternativen Versuchsaufbau (2) 51
4.15 Ergebnis der Fusion mit der Allied Vision Kamera und der Lidar-Punktwolke im alternativen Versuchsaufbau 51
4.16 Ergebnis der Fusion mit der Allied Vision Kamera und der Lidar-Punktwolke im alternativen Versuchsaufbau (2) 52
4.17 Versuchsaufbau in Simulation 53
4.18 Messungen der Sensoren im simulierten Versuchsaufbau 54
4.19 Ergebnis der Fusion vom Bild von Kamera 1 und dem Lidar 54
4.20 Ergebnis der Fusion vom Bild von Kamera 2 und dem Lidar 55

´

1 Einleitung

Damit autonome Systeme, wie etwa autonome Fahrzeuge, korrekt und zuverlässig arbeiten können, müssen diese ihre Umgebung so gut wie möglich wahrnehmen können. Dies kann durch den Einsatz von verschiedenen Arten von Sensoren bewerkstelligt werden, wie beispielsweise Kameras, oder Distanzmessungssensoren wie Radar oder Lidar. Werden mehrere solcher Sensoren im selben System zum Einsatz gebracht, müssen die Daten dieser Sensoren sinnvoll zusammengeführt werden. Diesen Prozess nennt man Fusion oder auch Datenfusion. Durch die Fusion der Messungen der einzelnen Sensoren kann die Umgebung besser und akkurater wahrgenommen und modelliert werden, was zu einer höheren Robustheit und Korrektheit des Systemverhaltens führt.

Diese Arbeit behandelt konkret die Fusion einer 3D-Punktwolke, die von einem 3D-Lidar-Sensor stammt, mit RGB-Farbbildern, die von einer oder mehreren Kameras stammen. Im Endergebnis wird jedem 3D-Punkt der Punktwolke ein oder mehrere RGB-Werte basierend auf den Kameradaten zugeordnet.

Es wurde ein Toolset in (ROS) entwickelt, welches diesen Prozess umsetzt. Es wurde außerdem eine Simulation mit (Unity) erstellt, in der Kameras und Lidars simuliert werden können. Das Toolset wurde anschließend auf die Simulation und Echtweltgeräte angewendet.

Diese Arbeit hat folgenden Aufbau: Im folgenden Teil des ersten Kapitels werden grundlegende Ziele und Konzepte einer Fusion behandelt und deren Relevanz in dieser Arbeit. Im zweiten Kapitel werden die theoretischen Grundlagen der in dieser Arbeit implementierten Kamera-Lidar Fusion beschrieben. Im dritten Kapitel wird die Umsetzung der Fusion mit dem ROS-Toolset und der Unity-Simulation erklärt. Im vierten Kapitel werden die Ergebnisse der Fusion evaluiert. Im fünften Kapitel wird ein Ausblick für mögliche weiterführende Konzepte gegeben.

1.1 Ziele einer Fusion

Wie bereits erwähnt ist es das Ziel einer Fusion, die Messungen von mehreren Einzelsensoren gewinnbringend zu vereinen, um eine bessere Repräsentation der Umwelt zu erlangen. In (Winner u. a. (2015) S. 441) werden die grundlegenden Konzepte einer Fusion vorgestellt. Insebsondere werden vier grundlegende Ziele einer Datenfusion definiert:

- Redundanz: Beispiel: Mehrere gleichartige Sensoren, die denselben Abschnitt wahrnehmen, um Störsignale und Unsicherheiten zu reduzieren
- Komplementarität: Beispiel: Mehrere verschiedenartige Sensoren, die auf verschiedenen physikalischen Konzepten basieren und denselben Abschnitt wahrnehmen um mehr Eigenschaften der Umgebung wahrzunehmen
- Zeitliche Aspekte: Beispiel: Erhöhung der Messgeschwindigkeit durch abwechselnd messenden Sensoren
- Kostenreduktion

Der wichtigste Punkt für diese Arbeit ist hierbei Komplementarität. Durch die zusätzliche Zuordnung von Farbwerten zu den Punkten der Punktwolke wird ein höheres Maß an Informationsgehalt gewonnen, was für Systeme, die diese Daten weiterverarbeiten, relevant ist. Dies wird ein Kapitel 5 näher behandelt.

Ein ebenfalls relevanter Punkt für diese Arbeit ist Redundanz. Das Toolset, welches im Rahmen dieser Arbeit entwickelt wurde, ermöglicht es, die Daten von mehreren Kameras mit verschiedenen räumlichen Positionen auf dieselbe Punktwolke abzubilden. Wenn es sich hierbei um gleichartige RGB-Kameras handelt, ist beispielsweise der Aspekt der Redundanz erfüllt. Außerdem kann ein zusätzliches Grad an Komplementarität erreicht werden, indem Kameras eingesetzt werden, die verschiedene Spektren wahrnehmen. Dadurch können einzelnen Punkten der Punktwolke mehrere Farbwerte zugeordnet werden, die jeweils auf verschiedenen Spektren basieren.

Der zeitliche Aspekt wird in dieser Arbeit vernachlässigt. Es werden keine Annahmen über Messgeschwindigkeit oder Latenz gemacht, da der Aufbau in dieser Arbeit eher prototypisch war. Systeme, die eine kritische Echtzeitanforderung haben, müssten dies berücksichtigen.

Der Kostenaspekt wurde ebenfalls vernachlässigt.

1.2 Sensordatenverarbeitung

Um zu verstehen wie genau eine Fusion von Daten umgesetzt werden kann, ergibt es Sinn, zunächst zu betrachten, wie Sensordaten von einzelnen Sensoren verarbeitet werden. In (Winner u. a. (2015) ab S. 442) wird beschrieben, wie eine Sensordatenverarbeitung generell abläuft (siehe Abbildung 1.1).

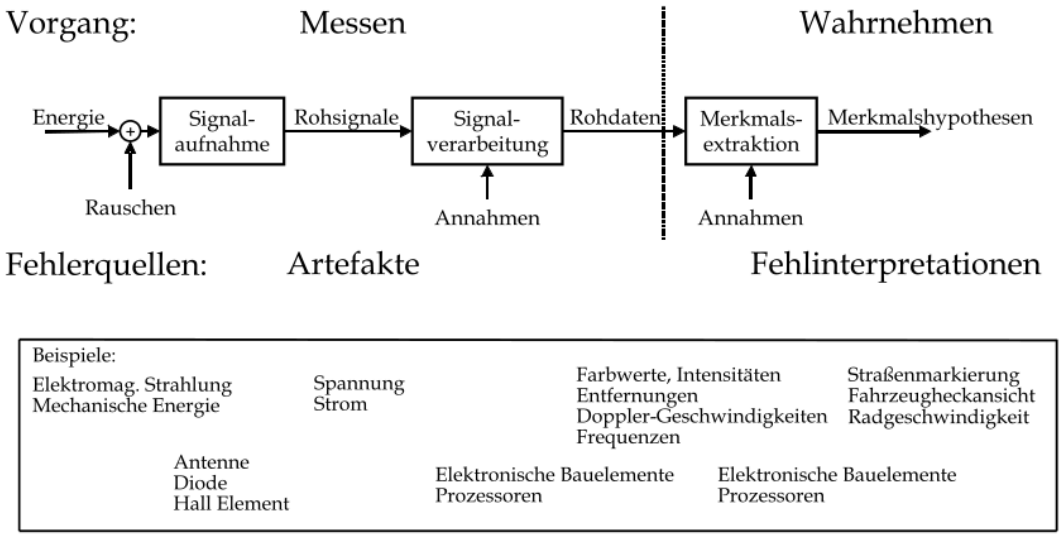

Abbildung 1.1: Prozess einer Sensordatenverarbeitung. Quelle: (Winner u. a. (2015), S. 443, Abbildung 24.1)

1.2.1 Messen

Der Prozess wird hierbei in zwei Bereiche unterteilt, *Messen* und *Wahrnehmen*. Im ersten Schritt des Messens wird Energie aus der Umwelt vom Sensor in elektrische Signale umgewandelt. Im beispielhaften Fall einer ordinären Kamera wäre dies zum Beispiel die Konvertierung der vom Kamera-Bildsensor detektierten elektromagnetischen Strahlung im Spektrum der vom menschlichen Auge sichtbaren Farben in Intensitätswerte. Dieser erste Schritt wird in (Winner u. a. (2015)) als *Signalaufnahme* beschrieben. Die resultierenden Werte werden *Rohsignale* genannt. Die Rohsignale allein sind jedoch nicht für die Weiterverarbeitung geeignet, da es sich um willkürliche Werte abhängig von der Funktionsweise des Sensors handelt. Diese Rohsignale müssen zunächst in ein einheitliches

generelles Format gebracht werden, damit auch andere Applikationen mit diesen Daten arbeiten können. Dieser Schritt wird in (Winner u. a. (2015)) *Signalverarbeitung* genannt, in dem Rohsignale in sogenannte *Rohdaten* umgewandelt werden und ist der letzte Schritt des Messvorgangs. Um Rohsignale in Rohdaten umzuwandeln müssen physikalische Annahmen über den Sensor und die resultierenden Rohsignale gemacht werden. Im Beispiel der Kamera wäre das Format der Rohdaten zum Beispiel das 8Bit-RGB Format, in dem für jeden Wert des jeweiligen Farbspektrums ein Wert zwischen 0 und 255 zugeordnet wird. Wenn nun ein Rohsignal vorliegt, und die maximalen Intensitätswerte für das Signal bekannt sind, kann dieses Rohsignal auf Werte zwischen 0 und 255 normalisiert werden, um das Rohdatenformat zu erhalten und den Messvorgang abzuschließen.

1.2.2 Wahrnehmen

Im nächsten Schritt des *Wahrnehmens* werden aus den Rohdaten sogenannte *Merkmale* extrahiert, aus denen dann *Objekthypthosen* getroffen werden. Im Beispiel einer Kamera könnte dies ein neuronales Netz sein, welches beispielsweise darauf trainiert ist, Autos zu erkennen und zu einem gegebenen Bild 2D-Bounding Boxen für die Position eines Autos im Bild zu schätzen.

Der Vorgang des *Warhnehmens* befindet sich hierbei auf einer höheren abstrakten Ebene als das Messen. Während es beim Messen darum geht, die Messwerte eines Sensors in eine einheitliche Form zu bringen, geht es beim Wahrnehmen darum, aus diesen Daten *Objekte* zu modellieren, mit denen beispielsweise Algorithmen, die eine Aktorik kontrollieren, Entscheidungen treffen können. Ein Fahralgorithmus kann beispielsweise entscheiden, ob sich auf einer Straße Fußgänger befinden und darauf reagierend rechtzeitig abbremsen. Ein *Modell* kann dabei mehr sein als eine simple Bilderkennung, es kann eine gesamte Modellierung eines Fahrzeugs in 3D samt Informationen über die aktuelle Geschwindigkeit und Fahrrichtung sein.

Um solche, wie sie in (Winner u. a. (2015)) genannt werden, *Objekthypothesen* treffen zu können, müssen die Ergebnisse von mehreren Sensoren in einer Fusion vereint werden. Die Fusion kann hierbei an verschiedenen Stellen im Prozess der Sensordatenverarbeitung stattfinden und auf verschiedene Arten.

1.2.3 Fehler

(Winner u. a. (2015) S.441) setzt fest:

"Oberstes Ziel der Datenfusion ist es, Daten von Einzelsensoren so zusammen zu führen, dass Stärken gewinnbringend kombiniert und/oder einzelne Schwächen reduziert werden."

Die *Stärken*, auf die hier verwiesen werden, sind die Ziele, auf die in Kapitel 1.1 dieser Arbeit eingegangen wurde. Die *Schwächen* beziehen sich auf die Fehler, die im Prozess der Sensordatenverarbeitung entstehen. In jedem Schritt der Sensordatenverarbeitung gelangen Fehler ins System, die berücksichtigt werden müssen (siehe Abbildung 1.1).

Bei der Signalaufnahme beispielsweise können auch so genannte *Rauschsignale* empfangen werden, die das eigentliche Objekt, welches am Ende wahrgenommen werden soll, überlagern. Bei einer Kamera beispielsweise können dies Flecken am Objektiv sein, die das Bild verdecken.

Wie bereits erwähnt werden Rohsignale eines Sensors in Rohdaten umgewandelt, indem physikalische Annahmen über den Sensor und die Rohsignale gemacht werden. Ein weiterer Fehler, der in den Verarbeitungsprozess eindringen kann, sind Fehler, die entstehen, wenn diese Annahmen verletzt werden. Diese Fehler werden nach (Winner u. a. (2015)) Artefakte genannt. Ein Beispiel hierfür bei Kameras ist der sogenannte *Rolling Shutter Effect*. Bei der Umwandlung von den Intensitätswerden in ein Bild wird angenommen, dass alle Werte, die einem Bild zugeordnet wurden, zum selben Zeitpunkt aufgenommen wurden. Falls die Kamera jedoch die Werte der einzelnen Elemente sequenziell abliest (beispielsweise reihenweise von oben nach unten), dann stammen die Pixel aus einem bestimmten Bereich des Bildes von einem Zeitpunkt vor oder nach diesem Zeitpunkt in einem anderen Bereich des Bildes. Falls sich die Szene in dieser Zeit merklich verändert (Ob durch Bewegung der Kamera oder durch sich im Bild bewegende Objekte) kommt es zu einer bildlichen Verzerrung (englisch: distortion). Der *Rolling Shutter Effekt* kommt in einer ähnlichen Form auch in dieser Arbeit bei Lidarmessungen vor, näheres dazu in Kapitel 4.1.2.

Fehler können auch bei der Extraktion von Merkmalen auftreten. In dieser Arbeit, beispielsweise, werden einzelnen Punkten einer Punktwolke Farben verliehen, indem diese Punkte auf das Kamerabild abgebildet werden. Das geschieht unter der Annahme, dass das Kamerabild die Geometrie der Szene korrekt wiederspiegelt. Jedoch ist es so, dass

durch die Kameralinse das Bild optisch verzerrt wird (auch Verzeichnung genannt). Falls diese Verzeichnung nicht korrigiert wird, wäre die Zuordnung der Farben auf die Punkte in der Punktwolke fehlerhaft, da sie die Umgebung nicht korrekt wiedergibt (näheres hierzu in Kapitel 2.2.3). (Winner u. a. (2015)) nennt diese Art von Fehler *Fehlinterpretation*.

1.3 Fusion von Kamera und Punktwolke

Um Kameradaten mit einer Puktwolke zu fusionieren, werden die einzelnen 3D-Punkte der Punktwolke in das Kamerabild abgebildet, wodurch die Farben für die Punkte abgelesen werden und mit diesen Farben assoziiert werden. Mit Hinblick auf die vorher behandelten Aspekte liegen sowohl die Kameradaten als auch die Lidar 3D-Punktwolke in Form von Rohdaten vor. Durch die Fusion wird der Punktwolke ein zusätzliches Merkmal, und zwar eine Farbe in Form eines RGB-Wertes verliehen, indem die Rohdaten der Kamera mit den Rohdaten des Lidars kombiniert werden. Das daraus resultierende Ergebnis ist jedoch keine höhere Objekthypothese, wie sie in (Winner u. a. (2015)) beschrieben wird. Das Ergebnis der Fusion liegt hier eher immer noch in Rohdatenform vor, da die Punkte der Punktwolke nicht verändert wurden, es wurde lediglich ein zusätzliches Merkmal, Farbe, hinzugefügt. Der Fusionsprozess in dieser Arbeit ist also weder strikt dem Messvorgang noch dem Wahrnehmungsvorgang zuzuordnen, sondern irgendwo dazwischen. Zwei Formen von Rohdaten werden hierbei in eine neue Form von Rohdaten gebracht, indem die Rohdaten der Kamera als Merkmal den Rohdaten des Lidars zugeorndet werden, mit der höhere Verarbeitungsalgorithmen unter Umständen besser arbeiten können. Wie die resultierenden Rohdaten möglicherweise weiterverarbeitet werden könnten, wird in Kapitel 5 diskutiert.

2 Theorie

Um zu verstehen, wie die 3D-Punkte einer Punktwolke in ein Kamerabild abgebildet werden können, ist es notwendig, zu verstehen, wie eine Kamera die Geometrie einer Szene wiedergibt, um diese rekonstruieren zu können. Hierfür muss näher betrachtet werden, wie das Kameramodell funktioniert.

2.1 Kameramodell

(Kaehler und Bradski (2016) S. 637) beschreibt die generelle Funktionsweise einer optischen Kamera. Eine Lichtquelle strahlt Licht in Form von Strahlen aus, dieses Licht wird von Objekten in der Welt reflektiert, welches dann vom Bildsensor der Kamera detektiert wird. Abhängig von der Oberfläche des Objekts werden dabei verschiedene Spektren des Lichts unterschiedlich absorbiert, wodurch verschiedene Helligkeiten und Farben wahrgenommen werden.

Neben der Oberflächeneigenschaft ist in einem Kamerabild auch die Geometrie der Szene relevant. Im Hinblick darauf ist eine Kamera ein Konstrukt, welches 3D-Koordinaten, die ein Objekt in der Welt beschreiben, in 2D-Pixel Koordinaten umwandelt, die eine Position in einem Bild beschreiben.

Es gibt verschiedene Kameramodelle, die die Geometrie einer Szene auf unterschiedliche Weise wiedergeben. Die häufigste Form, in der auch das menschliche Auge vertreten ist, die Mehrheit der Kameramodelle in der Computergrafik, als auch die Kameras, die in dieser Arbeit verwendet wurden, ist das perspektivische Kameramodell.

2.1.1 Lochkameramodell

Das einfachste perspektivische Kameramodell ist das Lochkameramodell (siehe Abbildung 2.1)

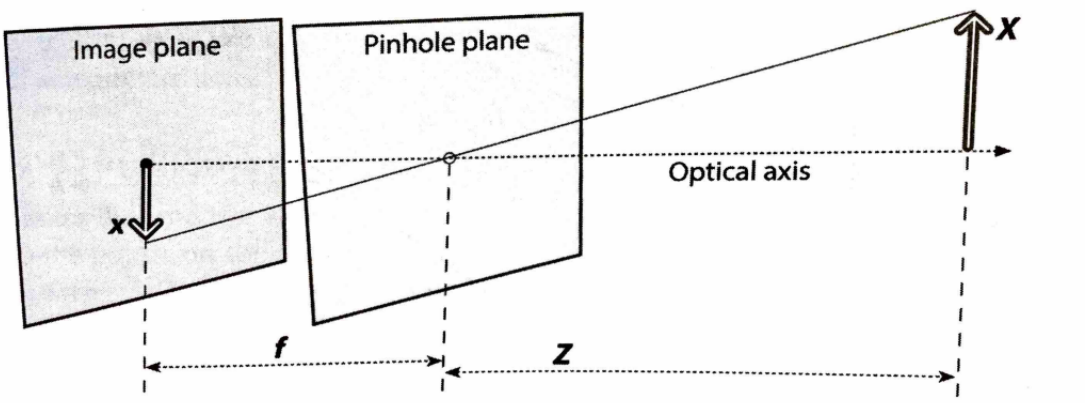

Abbildung 2.1: Lochkameramodell. Quelle: (Kaehler und Bradski (2016) S. 639, Abbildung 18.1)

Im Lochkameramodell fällt jeder Lichtstrahl aus der Szene zunächst durch eine Lochblende (englisch *pinhole*) und durch dieses auf eine Ebene, die sich hinter der Lochblende befindet, die sogenannten *Bildebene* (englisch *image plane*). Folglich stehen die Bilder, die auf die Bildebene projiziert werden, auf dem Kopf. Die Gerade, die durch die Lochblende und die Mitte der Bildebene geht, wird *optische Achse* (englisch *optical axis*) genannt. Angenommen f ist der Abstand der Bildebene zur Lochblende, die sogenannte *Brennweite* (englisch *focal length*), Z der Abstand des Objektes zu der Lochblende und X die Höhe des Objektes relativ zur optischen Achse, so kann die Höhe der Abbildung des Objekts auf der Bildebene folgendermaßen berechnet werden:

$$-x = f * \frac{X}{Z}$$

Diese Formel kann auch für die y-Komponente des Objekts, also der Weite relativ zur optischen Achse verwendet werden, wodurch das Objekt dann vollständig auf die Bildebene projiziert werden kann.

Damit das Bild nicht auf dem Kopf steht, kann die Bildebene statt hinter der Lochblende auch vor der Lochblende platziert werden (siehe Abbildung 2.2):

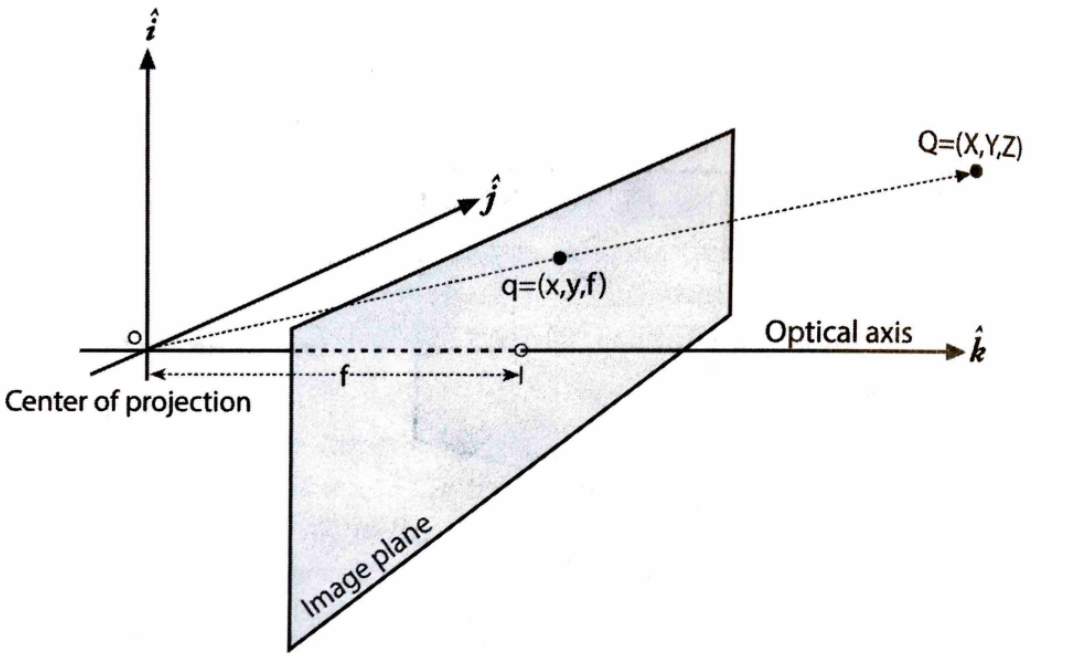

Abbildung 2.2: Lochkameramodell mit Bildebene vor der Lochblende. Quelle: (Kaehler und Bradski (2016), S. 640, Abbildung 18.2)

Die Geometrie der Szene wird durch diese Änderung nicht verändert und auch die Formel zur Berechnung der Abbildungshöhe bleibt gleich, nur das Vorzeichen fällt für die Berechnung weg:

$$x = f * \frac{X}{Z}$$

Wenn man hierbei von einem normalisierten Koordinatensystem ausgeht, bei der die Brennweite immer $f = 1$ ist, so bleibt nur noch:

$$x = \frac{X}{Z}$$

Um die Position eines 3D-Punktes auf der Bildebene zu berechnen muss also nur jeweils ihre x und y Komponente durch ihre z Komponente geteilt werden. In Systemen, in denen $f \neq 1$ ist, stellt die Brennweite also praktisch nur einen Skalierungsfaktor dar.

Diese Form des Lochkameramodells ist die einfachste Form einer perspektivischen Projektion, die häufig in virtuellen Grafikprogrammen verwendet wird. In (Akenine-Möller u. a. (2018) S. 96) wird dieser Prozess so beschrieben, wie er in (OpenGL) umgesetzt wird.

In Computerprogrammen wird hier konventionellerweise mit *Transformationsmatrizen* gearbeitet, da sie die Berechnung dieser Modelle vereinfachen.

In (Akenine-Möller u. a. (2018) S. 96, Formel Nr. 4.68) wird hierfür folgende (homogene) Projektionssmatrix zur Verfügung gestellt:

$$P_p = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix} \tag{2.1}$$

d ist in diesem Fall die Brennweite und das Vorzeichen ergibt sich daraus, dass die z-Achse in OpenGL in die entgegengesetzte Richtung zur Blickrichtung der Kamera zeigt. Um einen Punkt p also auf die Bildebene zu projizieren, muss er mit dieser Matrix multipliziert werden. Dass diese Matrix lediglich die vorher beschriebene Projektion erzielt, wird in (Akenine-Möller u. a. (2018) Formel Nr. 4.69, S. 97) gezeigt:

$$q = P_p p = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ -p_z/d \end{pmatrix} \Rightarrow \begin{pmatrix} -dp_x/p_z \\ -dp_y/p_z \\ -d \\ 1 \end{pmatrix} \tag{2.2}$$

Das Endergebnis zeigt, dass die Werte p_x und p_y des Punktes, der projiziert werden soll, durch den p_z Wert dividiert wurden, um die x und y Werte der Projektion zu berechnen, zusätzlich multipliziert mit der Brennweite.

Nachdem der Punkt auf die Bildebene projiziert wird, liegen die Koordinaten der Projektion nach wie vor im Koordinatensystem der Kamera. Damit das Objekt gerendert werden kann, werden aber Pixelkoordinaten benötigt (beispielsweise Werte zwischen $(0, 0)$ und $(1080, 1920)$ für ein Full-HD Bild). Dies wird in vielen Computergrafikframeworks, unter anderem auch OpenGL folgendermaßen erreicht (Akenine-Möller u. a. (2018), S. 99, Formel 4.75):

$$P_{OpenGL} = \begin{pmatrix} c/a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & -\frac{f'+n'}{f'-n'} & -\frac{2f'n'}{f'-n'} \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{2.3}$$

c ist hierbei eine Konstante die sich aus $c = 1.0/tan(\phi/2)$ zusammen setzt. ϕ ist hierbei der manuell festgelegte Bildwinkel der Kamera in Grad. a ist das Seitenverhältnis des Bildes und n' und f' definieren je die *Near- und Far-Clipping Plane* der Kamera, die in der virtuellen Computergrafik den Bereich festlegen, in der Objekte gerendert werden.

Diese Matrix beschreibt das Verhalten einer virtuellen Kamera komplett. Mit ihr können 3D-Punkte einer Szene direkt in Pixel-Koordinaten umgewandelt werden. Man nennt diese Form von Matrix daher auch *Kameramatrix*.

2.1.2 Physische Kameras

Die Methode der perspektivischen Transformation mit definiertem Bildwinkel und Clipping Planes ist in der Computergrafik weitverbreitet, für die Zwecke in dieser Arbeit aber eher irrelevant. Das Ziel dieser Arbeit ist es, die 3D-Punkte einer Punktwolke in ein physisches Kameramodell abzubilden. Das physische Kameramodell unterscheidet sich von dem virtuellen Kameramodell in einigen Punkten.

Im virtuellen Kameramodell wird ein Bildwinkel definiert und eine darauf basierende Kameramatrix berechnet. Der Bildwinkel einer physischen Kamera hingegen ist eine intrinsische Eigenschaft, die vom Bildsensor der Kamera und der Einstellung der Linse abhängt. Die Kameramatrix einer physischen Kamera ist also eine intrinsische Eigenschaft, die durch spezielle Heuristiken geschätzt werden muss. Nur wenn die Kameramatrix, die eine Kamera beschreibt, bekannt ist, können durch das Bild dieser Kamera Rückschlüsse auf etwaige Farbeigenschaften von Objekten in der Szene gemacht werden, da nur so die Geometrie der Szene rekonstruiert werden kann.

Außerdem ist eine virtuelle Kamera eine idealisierte Form und ignoriert gewisse Fehleinflüsse unter der physische Kameras leiden, beispielsweise radiale und tangentiale Verzeichnung, auf die (Kaehler und Bradski (2016) Kapitel 18) und (Förstner und Wrobel (2016) Kapitel 12) näher eingehen.

2.2 Physische Kameramatrix

Während die Kameramatrix von einer virtuellen Kamera meist mit einer 4x4 Matrix angegeben wird, wird die Kameramatrix einer physischen Kamera eher in 3x3 angegeben. Die (immer noch homogene) 3x3 Kameramatrix, die einen 3D-Punkt lediglich auf die Bildebene projiziert, sieht folgendermaßen aus:

$$P = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

Dass diese Matrix lediglich dieselbe Projektion wie die Matrix in Formel Nr. 2.1 durchführt, wird hier gezeigt:

$$P = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} p_x * f \\ p_y * f \\ p_z \end{pmatrix} = \begin{pmatrix} \frac{fp_x}{p_z} \\ \frac{fp_y}{p_z} \\ 1 \end{pmatrix} \tag{2.5}$$

Formel 2.4 beschreibt jedoch nur die Projektion eines 3D-Punktes auf die Bildebene. Die vollständige Kameramatrix einer physischen Kamera, die einen 3D-Punkt in 2D-Pixelkoordinaten umwandelt, hat folgendes Format (Kaehler und Bradski (2016) S. 641):

$$M = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.6}$$

Statt eines einzelnen Parameters für die Brennweite, gibt es jetzt zwei, f_x und f_y. Außerdem wurden zwei weitere Parameter, c_x und c_y hinzugefügt.

2.2.1 Affine Störquellen

In Formel Nr. 2.3 für die Kameramatrix einer virtuellen Kamera sieht man, dass einer der beiden Brennweiten-Werte mit einem zusätzlichen Parameter a, dem Seitenverhältnis des Bildes, multipliziert wird. Würde dies nicht geschehen, würde das Bild, welches von

der Kameramatrix erzeugt wird, in die Höhe oder Breite verzerrt werden, da eine quadratische Projektion auf ein nicht quadratisches Bild abgebildet werden würde. Dies trifft auch auf eine physische Kamera zu, jedoch gibt es einen weiteren Grund, weshalb physische Kameras zwei Parameter für die Brennweite haben: Selbst wenn das Bild, auf das die Projektion abgebildet wird, quadratisch wäre (ist der Fall, wenn das Verhältnis der Dimensionen der Anzahl der Bildsensorelemente 1:1 ist, zum Beispiel 1000x1000), hätte die Kameramatrix dennoch verschiedene Werte für f_x und f_y, da die einzelnen Sensorelemente des Bildsensors einer physischen Kamera nicht immer komplett quadratisch sind (siehe Abbildung 2.3).

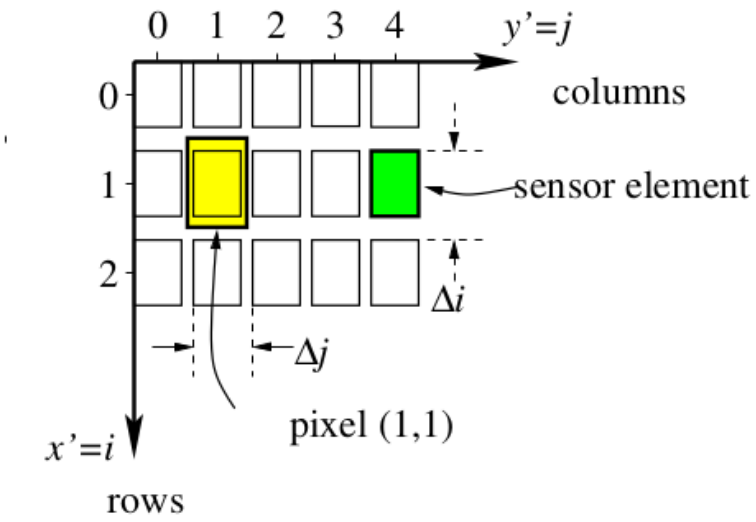

Abbildung 2.3: Rechteck-Eigenschaften der Bildsensorelemente einer physischen Kamera. Quelle: (Förstner und Wrobel (2016), S. 471, Abbildung 12.9)

Die Parameter f_x und f_y sind also je von der Höhe und Breite der einzelnen Bildsensorelemente abhängig. In (Kaehler und Bradski (2016) S. 641) wird explizit angegeben, dass gilt:

$$f_x = F * s_x, \;\; f_y = F * s_y$$

Wobei F die tatsächliche Brennweite ist und s_x und s_y jeweils die Höhe und Breite der Sensorelemente, in einer Einheit von Pixel pro Millimeter. Nach (Kaehler und Bradski

(2016)) kann weder die tatsächliche Brennweite noch die Größe der Sensorelemente direkt gemessen werden, was bedeutet, dass eine Heuristik, die die Kameramatrix einer physischen Kamera schätzt, nur Annäherungen an die tatsächlichen Werte von f_x oder f_y berechnen kann.

In einer virtuellen Kameramatrix, wie beispielsweise in Formel Nr. 2.3 ist das Endergebnis der Transformation in einem Format, in der die Bildkoordinaten in einem System enthalten sind, in dem die Mitte des Bildes (Schnittpunkt der optischen Achse mit der Bildebene, auch *Hauptpunkt* genannt (englisch *principal point*)) den Ursprung bildet und die Grenzen (Ränder des Bildes) durch $x = 1, -1$ und $y = 1, -1$ definiert sind. Dies ist in der Computergrafik praktisch, da durch diese "normalisierten Bildkoordinaten" die tatsächlichen Pixelkoordinaten aufgrund der aktuellen Größe des Fensters berechnet werden können, die sich unter Umständen während der Laufzeit öfters ändern. Eine physische Kameramatrix hingegen hat immer dieselbe (native) Auflösung, da die Anzahl der Bildsensorelemente fest ist. Für eine Kameramatrix ergibt es also mehr Sinn, das Bild in ein Koordinatensystem abzubilden, das die endgültigen Pixelkoordinaten im Bild beschreibt. In (OpenCV) ist es hierbei Konvention, dass der obere linke Rand den Ursprung bildet, die y-Achse nach unten und die x-Achse nach rechts zeigt. Für eine Kamera mit Auflösung von 1920x1080 wäre die obere linke Ecke bei den Koordinaten $(0, 0)$ und die untere rechte Ecke bei $((1080, 1920)$ (siehe Abbildung 2.4).

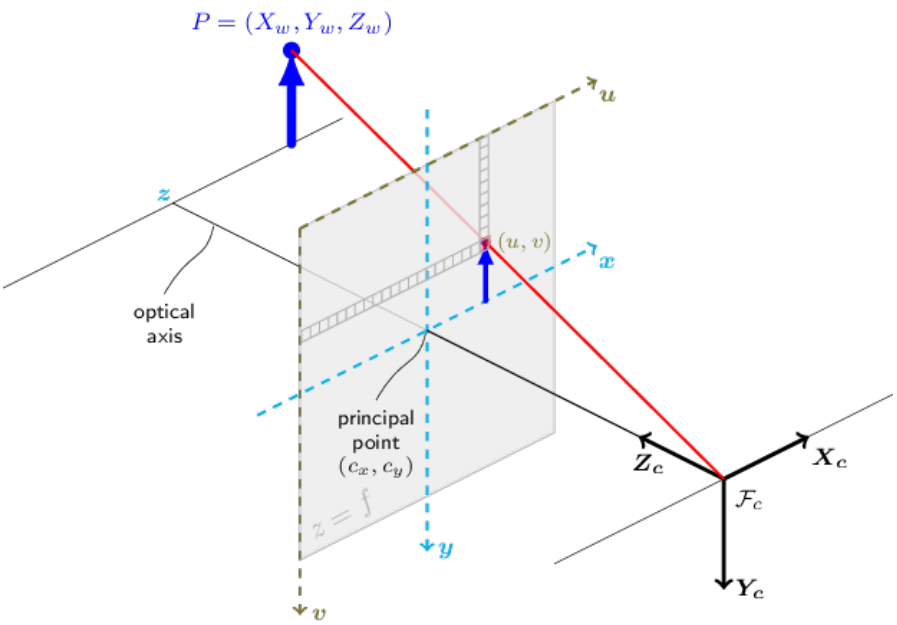

Abbildung 2.4: Kameratransformation in OpenCV. Quelle: (OpenCVDocs)

Bei physischen Kameras ist es aber grundlegend der Fall, dass die Mitte des Bildes nicht ganz auf der optischen Achse liegt, da diese Art von Präzision im Bauvorgang einer Kamera einfach nicht möglich ist. Das bedeutet, dass die Mitte des Bildes geringfügig von der optischen Achse abweicht. Eine Heuristik, die also die Kameramatrix einer Kamera schätzt, muss dies berücksichtigen. Hierfür gibt es die zusätzlichen Parameter c_x und c_y, die die Mitte des Bildes also zurück auf den Hauptpunkt verschieben. Ähnlich wie f_x und f_y können diese Werte nicht gemessen oder berechnet werden, sondern müssen von einer Heuristik geschätzt werden.

2.2.2 Nicht affine Störquellen

Die rechteckige Eigenschaft der Bildsensorelemente und die Abweichung der Bildmitte von der optischen Achse beeinflussen das endgültige Bild der Kamera in einer Weise, die *affin* ist. Das bedeutet, dass die Art und das Ausmaß der Störung an jeder Stelle im Bild gleich ist. Diese affinen Störquellen werden praktisch in allen physischen Kameramatrizen modelliert, damit sie die Geometrie der Szene richtig wiederspiegeln. Es gibt aber auch *nicht affine Störquellen*, die das Kamerabild beeinflussen können. Diese Störquellen zu modellieren ist optional und hängt davon ab, wie stark diese Effekte auftreten. Es gibt zwei dieser Störquellen, die auch in dieser Arbeit relevant sind und zwar die *radiale* und *tangentiale Verzeichnung* (englisch *radial and tangential distortion*). Diese Störquellen zu modellieren ist komplizierter und kann nicht durch das simple Hinzufügen von weiteren Parametern in eine Kameramatrix erreicht werden. Stattdessen werden diese Störquellen auf eine Weise modelliert, in der das Kamerabild so vorbearbeitet wird, so dass diese Störquellen weitesegehend neutralisiert werden, wodurch die Geometrie der Szene nach wie vor korrekt von einer Kameramatrix beschrieben wird.

2.2.3 Radiale Verzeichnung

Die Rate mit der eine physische Kamera Bilder aufnehmen kann, ist meist daran gekoppelt, wie lange es dauert, bis sich genügend Licht auf den Bildsensoren gesammelt hat, sodass ein ausreichender Wert für die Intensität abgelesen werden kann. Je mehr Licht dabei auf die Sensorelemente fällt, desto schneller können Bilder ausgelesen werden. Deswegen werden die meisten Kameras mit *Linsen* ausgestattet, die das Licht aus der Umgebung auf den Bildsensor der Kamera bündeln. Hierdurch wird das Bild, welches auf den Bildsensor fällt, verzerrt. Man nennt diesen Effekt auch *radiale Verzeichnung.*

Wegen der Funktionsweise einer Linse ist hierbei das Ausmaß der Verzeichnung in der Mitte des Bildes kleiner und an den Rändern größer (siehe Abbildung 2.5 und 2.6)

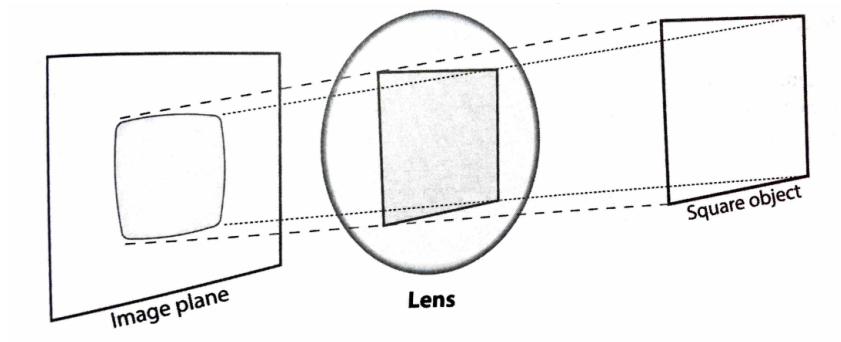

Abbildung 2.5: Verzeichnung durch eine Linse Quelle: (Kaehler und Bradski (2016) S. 645, Abbildung 18.3)

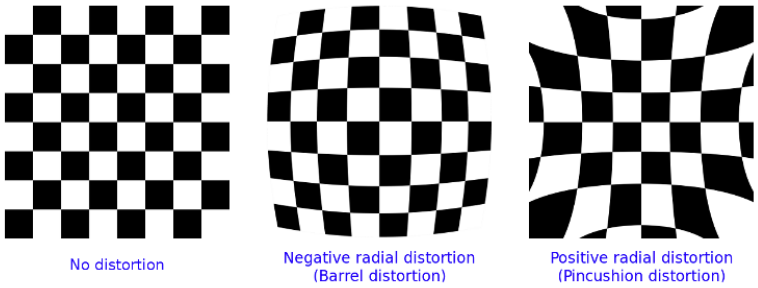

Abbildung 2.6: Verschiedene Arten von Verzeichnung Quelle: (OpenCVDocs)

Der Effekt einer solchen Verzeichnung ist, dass Linien, die in der Szene eigentlich gerade sind, im Bild dann gebogen auftauchen.

Um diese Verzeichnung zu modellieren schätzt eine Heuristik, die eine Kameramatrix aufgrund eines Bildes schätzt, auch die sogenannte *Verzeichnungsparameter* (englisch *distortion parameter*). Durch sie kann diese Verzeichnung charakterisiert und rückgängig gemacht werden. Durch die Verzeichnung werden im Grunde Pixel im Bild auf der Geraden, die sich zwischen ihrer Position und der Mitte des Bildes befindet, entweder zur Mitte hin oder weg verschoben. Je weiter der Pixel dabei von der Mitte des Bildes entfernt ist, desto größer ist die Verschiebung. Wenn also für jeden Pixel ein Transformationsvektor ermittelt werden kann, der den Pixel wieder an die korrekte Position verschiebt, kann die Geometrie der Szene wiederhergestellt werden.

Die mathematische Funktion, die aus den Koordinaten eines Pixels die korrigierten Koordinaten basierend auf den Verzeichnungsparametern berechnet, wird mit Taylorreihen approximiert. Die Anzahl der Terme der Reihe ist dabei beliebig, meist werden um die Funktion genügend zu approximieren aber nur zwei benötigt, welche durch zwei Verzeichnungs-Parameter beschrieben werden, in Fällen von besonders starker Verzerrung auch drei. Hierbei gilt nach (Kaehler und Bradski (2016) S. 646):

$$x_{corrected} = x * (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{corrected} = y * (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

k_1, k_2 und k_3 stellen hierbei die Verzeichnungsparameter für die radiale Verzeichnung dar. x und y sind die Koordinaten im verzeichneten Bild, $x_{corrected}$ und $y_{corrected}$ sind die neuen Koordinaten im korrigierten Bild. Die Funktion, die hierbei durch Taylorreihen approximiert wird, ist eine Funktion, die das Maß der Verschiebung basierend auf der Entfernung des Punktes zur Mitte des Bildes (r) berechnet. Die Richtung, in die verschoben werden muss, ist immer bekannt, und zwar Richtung Mitte (falls das Ergebnis positiv ist, sonst von der Mitte weg).

Hierdurch kann ein 2D-Array an Vektoren erstellt werden, welche in OpenCV eine *undistortion map* genannt wird (siehe Abbildung 2.7).

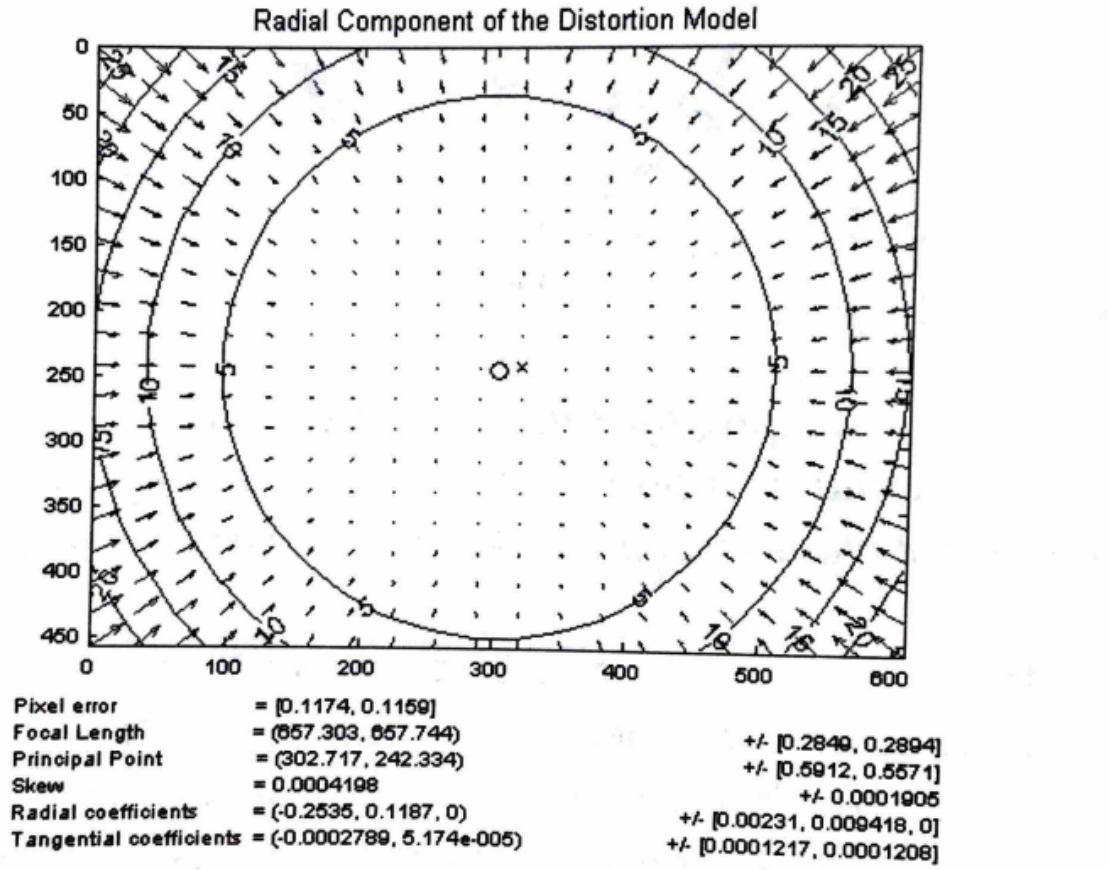

Abbildung 2.7: Undistortion Map für radiale Verzeichnung. Die Pfeile zeigen die Verschiebung aufgrund der Verzeichnung an. Quelle: (Kaehler und Bradski (2016), S. 646, Abbildung 18.4)

Wenn diese Map auf ein verzeichnetes Bild angewendet wird, werden alle Pixel im Bild auf eine neue Position im korrigierten Bild verschoben, sodass die Kameramatrix die Geometrie der Szene wieder korrekt beschreibt (siehe Abbildung 2.8). Diese Map wird komplett durch die Verzeichnungsparameter für die radiale Verzeichnung beschrieben.

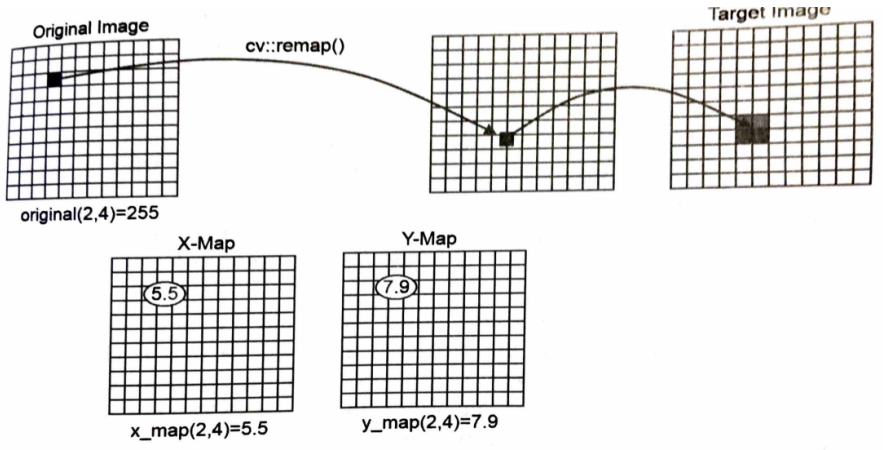

Abbildung 2.8: Anwendung einer Undistortion Map. Da es sein kann, dass ein Pixel auf eine nicht ganzzahlige Koordinate abgebildet wird, wird der Pixel mit anteilhafter Intensität auf umliegende Pixel verteilt. Quelle: (Kaehler und Bradski (2016), S. 679, Abbildung 18.18)

2.2.4 Tangentiale Verzeichnung

Neben der radialen Verzeichnung gibt es auch die *tangentiale Verzeichnung*. Tangentiale Verzeichnungen treten dann auf, wenn der Bildsensor der Kamera nicht genau parallel zur Linse ist (auch folglich von bautechnischen Limitationen). Dadurch wird das Licht durch die Linse unregelmäßig auf den Bildsensor gebündelt, was zu einer Verzerrung führt (siehe Abbildung 2.9)

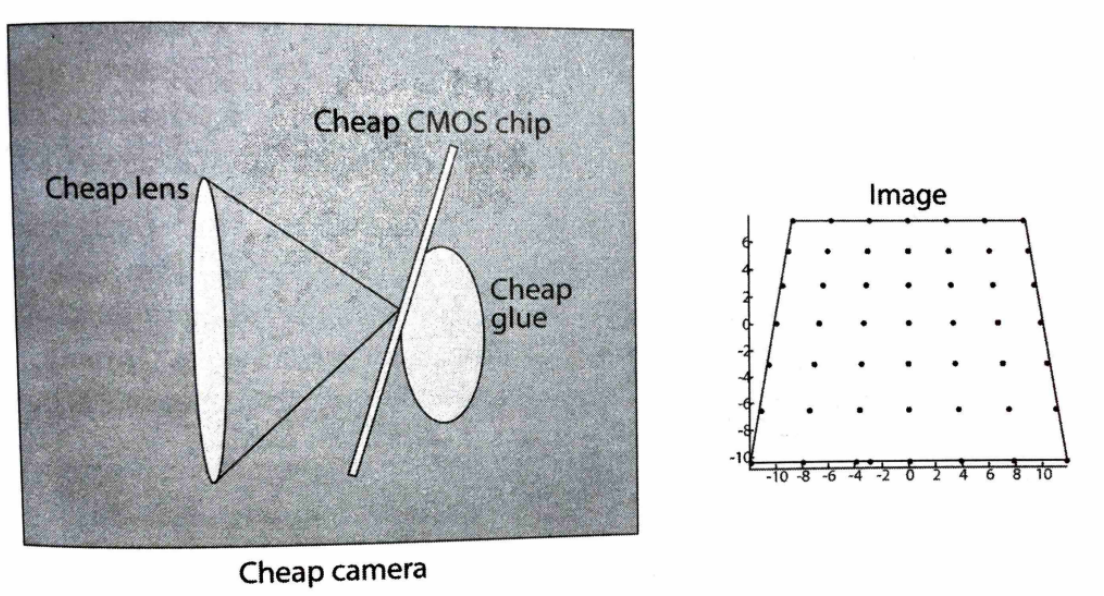

Abbildung 2.9: Tangentiale Verzeichnung. Quelle: (Kaehler und Bradski (2016), S. 647, Abbildung 18.5)

Die korrigierten Werte für eine tangentiale Verzeichnung werden folgendermaßen berechnet (Kaehler und Bradski (2016), S. 647):

$$x_{corrected} = x + [2p_1xy + p_2(r^2 + 2x^2)]$$

$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

Die tangentiale Verzeichnung wird ähnlich wie die radiale Verzeichnung berechnet, für die tangentiale Verzeichnung benötigt man aber immer nur genau zwei Parameter, p_1 und p_2.

Da die Korrektur für beide Verzeichnungsarten ähnlich berechnet wird, werden beide Formen mit derselben *undistortion map* beschrieben. Die Map, die beide Arten von Verzerrung komplett beschreibt, wird von 4 Parametern charakterisiert. Sie wird meist in OpenCV in Form von einer 1x4 Matrix mit folgender Form angegeben.

$$[k_1, k_2, p_1, p_2]$$

Ein Beispiel für eine Undistortion Map, die eine radiale und tangentiale Verzeichnung beschreibt, kann in Abbildung 2.10 betrachtet werden.

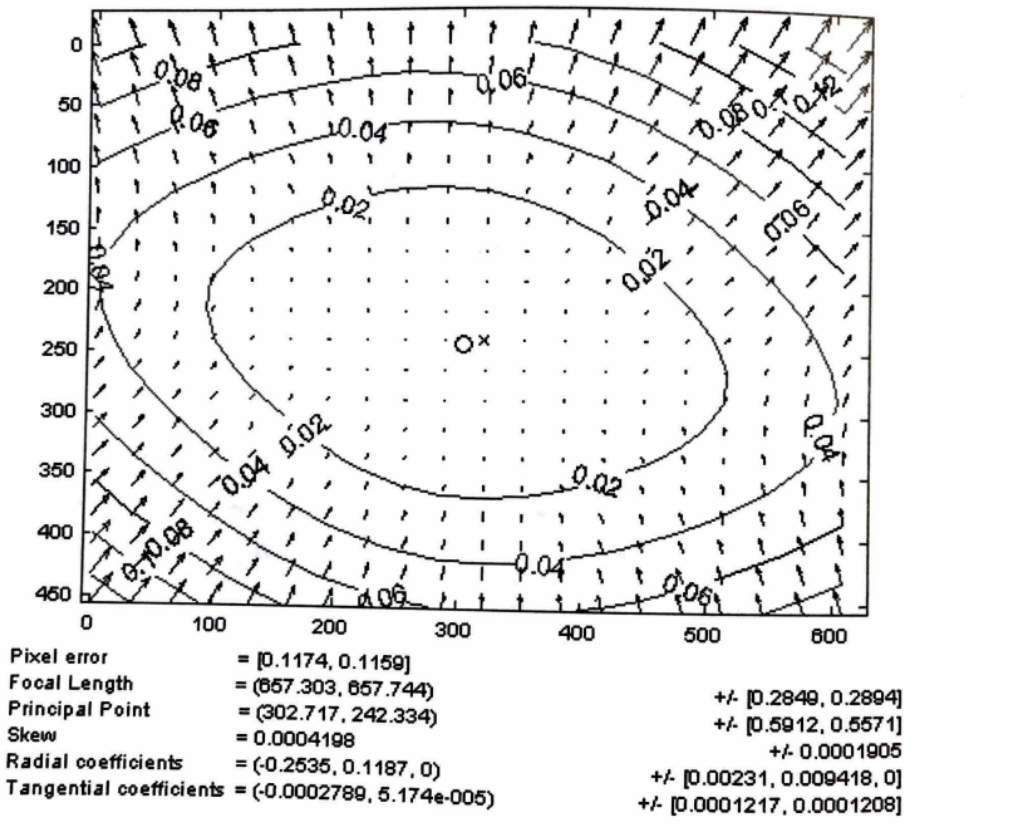

Abbildung 2.10: Undistortion Map für radiale und tangentiale und Verzeichnung. Quelle: (Kaehler und Bradski (2016)), S. 648, Abbildung 18.6

2.2.5 Intrinsische Parameter

Mit den bis jetzt genannten Parametern lässt sich die Geometrie einer Kamera ausreichend rekonstruieren. Wenn also die acht Parameter f_x, f_y, c_x, c_y, k_1, k_2, p_1, p_2 bekannt sind, dann ist es möglich, aufgrund des Kamerabildes Rückschlüsse über die Oberflächeneigenschaften in der Szene zu machen. Diese Parameter werden auch die *intrinsischen* Parameter der Kamera genannt.

2.3 Abbilden von 3D-Punkten in das Kamerabild

Mit einer vollständigen Kameramatrix und einem korrigierten Bild, welches die Verzeichnung neutralisiert, ist es nun möglich, 3D-Punkte auf das Kamerabild abzubilden, um so Aussagen über die Oberflächeneigenschaften des Punktes basierend auf dem Kamerabild zu machen. Jedoch besteht das Problem, dass sich die Kameramatrix und die 3D-Punkte der Punktwolke in verschiedenen Koordinatensystemen befinden. Die 3D-Punkte der Punktwolke stammen aus dem Sensorkordinatensystem des Lidars und das Bild der Kamera aus dem Sensorkoordinatensystem der Kamera.

Zu beachten ist hierbei, dass eine Kameramatrix, egal ob physisch oder virtuell, immer davon ausgeht, dass 3D-Punkte, die mit der Kameramatrix abgebildet werden sollen, im *Kamerakoordinatensystem* bereitgestellt werden. Hierbei handelt es sich um ein Koordinatensystem, bei dem sich der Urpsrung im Brennpunkt der Kamera befindet und die Achsen auf der Orientierung der Kamera basieren. Das konventionelle Kamerakoordinatensystem in OpenCV, welches auch in dieser Arbeit verwendet wird, ist in Abbildung 2.4 dargestellt. Wenn also 3D-Punkte aus dem Lidar-Koordinatensystem vorliegen, müssen diese zunächst in das jeweilige Kamerakoordinatensystem umgewandelt werden, damit die 3D-Punkte auf das Kamerabild abgebildet werden können. Abbildung 2.11 zeigt den Zusammenhang zwischen Welt- und Kamerakoordinatensystem.

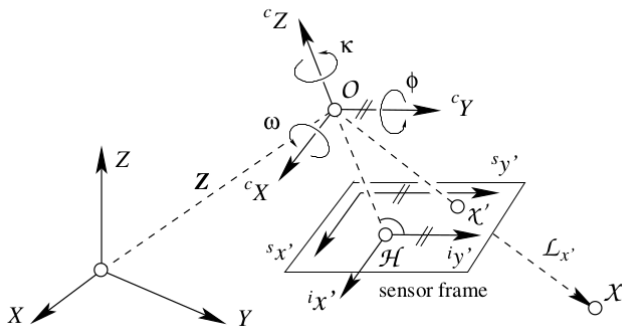

Abbildung 2.11: Zusammenhang zwischen Welt-, Kamera- und Bildkoordinatensystem. Quelle: (Förstner und Wrobel (2016), S. 462, Abbildung 12.5)

Auch wenn mehrere Kameras benutzt werden, um Oberflächeneigenschaften von verschiedenen Kameras zu erhalten, kann dies bewerkstelligt werden, indem der jeweilige

3D-Punkt zunächst in das jeweilige Kamerasystem konvertiert wird, um dann die jeweilige Kameramatrix anzuwenden. Das Lidar-Koordinatensystem fungiert in diesem Fall als Basis, von dem aus die 3D-Punkte in jedes beliebige Kamerakoordinatensystem übertragen werden können.

2.3.1 Extrinsische Parameter

Die *intrinsischen* Parameter einer Kamera, also die Kameramatrix und die Verzeichnungsparameter, beschreiben, wie eine Kamera die Geometrie einer Szene wiedergibt. Um jedoch 3D-Punkte aus einem externen Koordinatensystem in der Kamera abzubilden, müssen außerdem die *extrinsischen* Parameter zwischen dem externen Koordinatensystem und dem Kamerakoordinatensystem bekannt sein. Hierbei handelt es sich je um eine Translation (Verschiebung) und eine Rotation (Orientierung), die bei Anwendung auf einen Punkt im Koordinatensystem des Objektes, den Punkt in das Koordinatensystem der Kamera überträgt. Abbildung 2.12 zeigt hierbei die Anwendung der extrinsischen Parameter auf einen Punkt in Objektkoordinaten, sodass der Punkt in das Kamerakoordinatensystem übertragen wird.

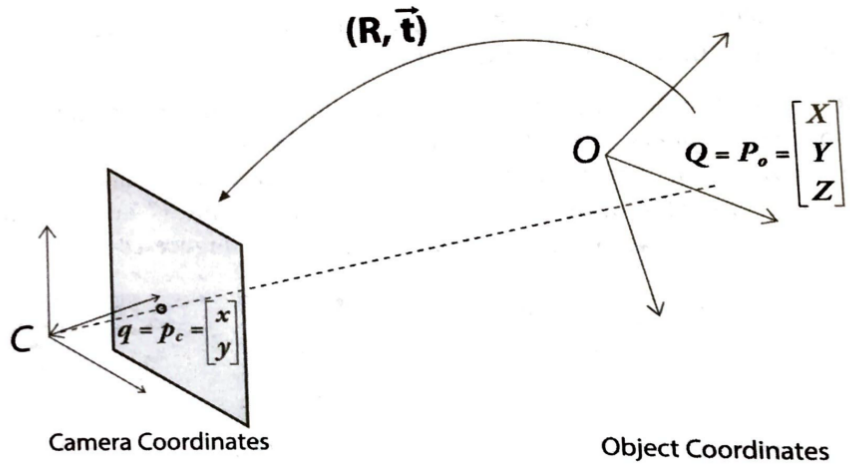

Abbildung 2.12: Umwandeln der Objektkoordinaten in Kamerakoordinaten. Quelle: (Kaehler und Bradski (2016), S. 650, Abbildung 18.8)

Nach (OpenCVDocs) handelt es sich bei den extrinsischen Parametern hierbei um einen 3x1 Translationsvektor

$$t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

und eine 3x3 Rotationsmatrix

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

wodurch die extrinsischen Parameter dann durch eine 4x4 Matrix mit der Form

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

beschrieben werden können.

Wenn ein 3D-Punkt P_w nun also in Weltkoordinaten vorliegt, kann der Punkt in das Kamerakoordinatensystem übertragen werden, indem berechnet wird:

$$P_c = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} * P_w$$

P_c ist hierbei der 3D-Punkt im Kamerakoordinatensystem, der benutzt werden kann, um den Punkt mithilfer der Kameramatrix in das Kamerabild abzubilden.

Für die extrinsischen Parameter wird für gewöhnlich in OpenCV jedoch nicht immer die gesamte 3x3 Matrix für die Rotation angegeben. In (Kaehler und Bradski (2016) S. 643) wird hierfür die *Rodrigues Transformation* eingeführt, mit der eine Rotation statt mit einer 3x3 Matrix mit einem 1x3 Vektor $r = \begin{bmatrix} r_x & r_y & r_z \end{bmatrix}$ beschrieben werden kann. Die Richtung des Vektors gibt hierbei die Achse an, um die rotiert wird und die Länge das Vektors das Ausmaß, um welches rotiert wird. Da diese Repräsentation die Anzahl der Parameter etwas reduziert und diese Darstellung beliebig in die 3x3 Form hin und zurück geformt werden kann, wird diese Form der Darstellung grundsätzlich über die 3x3 Darstellung bevorzugt.

2.4 Zusammenfassung

Um also einen 3D-Punkt, der in einem Koordinatensystem, der unabhängig vom Koordinatensystem einer Kamera ist, in das Bild dieser Kamera abbilden zu können, sodass aufgrund des Bildes der Kamera Oberflächeneigenschaften für den 3D-Punkt gemacht werden können, benötigt man die *intrinsischen* Parameter der Kamera und die *extrinsischen* Parameter zwischen dem Kamerakoordinatensystem und dem Weltkoordinatensystem.

Diese umfassen hierbei die vierzehn (skalaren) Parameter f_x, f_y, c_x, c_y, k_1, k_2, p_1, p_2, t_x, t_y, t_z, r_x, r_y und r_z.

Sind diese Parameter bekannt, kann dieser Prozess umgesetzt werden.

3 Umsetzung

In diesem Kapitel wird die Umsetzung der in Kapitel 2 geschilderten Konzepte in dieser Arbeit beschrieben. Die Software, die die in Kapitel 2 geschilderte Transformation eines 3D-Punktes auf ein Kamerabild projiziert, sodass für den Punkt ein Farbwert abgelesen wird, ist Teil eines in dieser Arbeit entwickelten ROS-Packages. (ROS) (Robot Operating System) ist ein weitverbreitetes und vielverwendestes Framework in der Robotik. Es stellt eine Plattform zur Verfügung, in der viele Funktionalitäten, die relevant in der Robotik sind (unter anderem auch Sensordatenverarbeitung), in sogenannten "Packages" zur Verfügung gestellt werden können. Dementsprechend wurde in dieser Arbeit ein solches ROS-Package erstellt, welches die Fusion einer 3D-Punktwolke mit einem Kamerabild implementiert. Dieses Package kann unter dem Git Repository unter diesem Link: (https://github.com/Adem-Can/ilm_ros_pkg) gefunden werden. Das Package wurde unter Ubuntu 20.04 LTS mit der ROS Version "Noetic Ninjemys" in C++ erstellt.

Außerdem wurde eine Simulation in Unity erstellt, die eine Kamera und einen Lidar in einer virtuellen Umgebung simulieren kann. Das Repository für das Unity-Projekt kann unter diesem Link gefunden werden: (https://github.com/Adem-Can/UnityROSSim).

3.1 Versuchsaufbau

Das ROS-Package, welches in dieser Arbeit erstellt wurde, unterscheidet grundsätzlich nicht zwischen simulierten Geräten oder echten Geräten. Außer einer Plattform, in der verschiedene Packages bereitgestellt werden können, stellt ROS auch eine Infrastruktur zur Verfügung, die einen Message-Broker implementiert, in der Teilnehmer dieses Sytems, sogenannte "ROS-Nodes", Nachrichten untereinander mit einem "Pub/Sub Pattern" untereinander austauschen können. Für den Zweck von gewöhnlicher Software sind

”ROS-Nodes” lediglich ordinäre Betriebssystems-Prozesse, die auf C++-Programmen basieren, die das ROS C++-Framework benutzen, um mit der ROS-Infrastruktur zu kommunizeren. Diese Nachrichten in der ROS-Infrastruktur haben vordefinierte Typen, die vielgenutzte Formate, die in der Robotik relevant sind, implementieren, darunter auch für Kamerabilder und 3D-Punktwolken. Die beiden vordefinierten Nachrichtentypen, die diese Formate implementieren und in dieser Arbeit exklusiv genutzt werden sind `sensor_msgs/Image` und `sensor_msgs/PointCloud2`. Das Package, welches in dieser Arbeit erstellt wurde, erwartet die Kameradaten und Lidardaten in diesen Formaten und berechnet darauf basierend eine Punktwolke, deren Punkte mit einer oder mehreren Farbwerten assoziiert wurden. Hierfür wurde in ROS ein benutzerdefinierter Nachrichtentyp, `ilm_ros_pkg/ColoredPointcloud` definiert.

3.1.1 Realwelt

Für den Realwelt-Aufbau wurden in dieser Arbeit drei Geräte verwendet:

- Eine Allied Vision G-130 VSWIR T1 Kamera
- Eine Basler a2a1920-51gcpro Kamera
- Ein Blickfeld Cube 1 Lidar

Die Geräte wurden alle an einer speziell entworfenen Halterung nebeneinander fixiert. Abbildung 3.1 und 3.2 zeigt den Versuchsaufbau und die Visualisierungen der jeweiligen Sensordaten.

Jedes dieser Geräte wird mit einem intern ausgestatteten ROS-Treiber geliefert, der die Daten des jeweilgen Sensors in den vorher erwähnten Nachrichtenformaten bereitstellt.

Bei der Allied Vision Kamera handelt es sich hierbei um eine SWIR-Kamera, die ein Schwarzweiß-Bild basierend auf einem Lichtspektrum, der sich im Infrarotbereich befindet, bereitstellt. Die Basler Kamera nimmt Bilder im normalen vom menschlichen Auge sichtbaren Farbspektrum auf und stellt dementsprechende Farbbilder bereit. Der Blickfeld Lidar misst Entfernungen basierend auf einem Laserstrahl im Infrarotbereich. Um eine Entfernung zu messen, wird hierbei die Zeit gemessen, die ein Lichtstrahl braucht, um von einer Oberfläche reflektiert und vom Sensor des Lidars wieder detektiert zu werden. Darauf basierend wird eine Entfernung geschätzt. Basierend auf dieser Entfernung und der Richtung des Laserstrahls kann so ein 3D-Punkt abgeleitet werden. Mithilfe eines

beweglichen Spiegels, der im Lidar-Sensor verbaut ist, kann so ein 3D-Raum weitesgehend gescannt werden, woraus dann eine Punktwolke abgeleitet wird.

Abbildung 3.1: Versuchsaufbau in Realwelt.

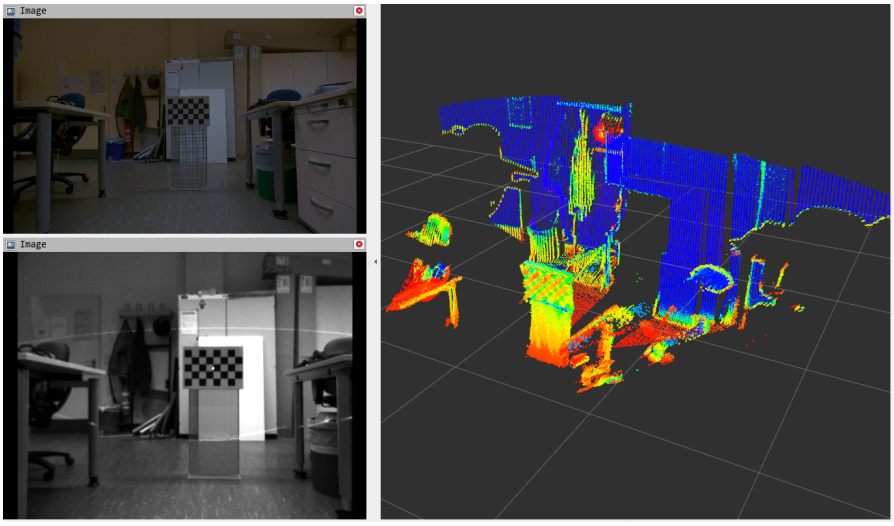

Abbildung 3.2: Visualisierung der Sensordaten. Oben links: Basler Kamera. Unten links: Allied Vision Kamera. Rechts: Blickfeld Lidar. Zu beachten ist, dass die Scan-Laser des Blickfeld Lidars im Bild der Allied Vision Kamera sichtbar sind, da diese in einem Spektrum sind, die von der Allied Vision Kamera detektiert werden.

Abbildung 3.1 zeigt hier den Aufbau mit diesen Geräten und Abbildung 3.2 zeigt eine Visualisierung der erhaltenen Sensordaten im Standard-Visualisierungstool von ROS, ”RViz”. Die Szene wurde hier mit einem zu allen Geräten parallelen 8x5 Schachbrett ausgestattet, deren Quadrate eine Größe von 5cm haben und welches genau 2m von der Halterung entfernt ist.

3.1.2 Simulation

Im Rahmen der Arbeit wurde auch eine virtuelle Szene mit der (Unity) 3D-Engine erstellt, die eine Kamera und auch einen Lidar in ähnlicher Weise, wie die Echtweltgeräte simulieren kann. Applikationen in der Unity-Engine werden mit C# entwickelt, jedoch kann auch von Unity aus mit der ROS-Infraktruktur kommuniziert werden, da Unity C#-Wrapper für das ROS-Framework bereitstellt. Dadurch können ROS-Messages, die aus der ROS-Laufzeit stammen auch in Unity empfangen werden und ROS-Messages, die von Unity gesendet werden, können in der ROS-Laufzeit empfangen werden.

Abbildung 3.3 zeigt hier den Aufbau der Szene in Unity. Hierbei wurde das Schachbrett aus dem Aufbau in Abbildung 3.1 und 3.2 mit den selben Dimensionen modelliert.

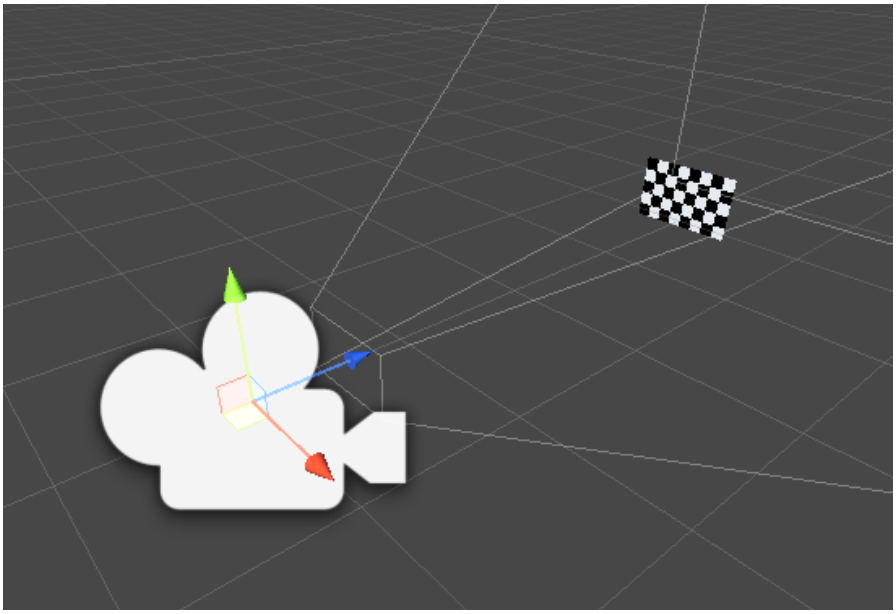

Abbildung 3.3: Simulierte Szene in Unity.

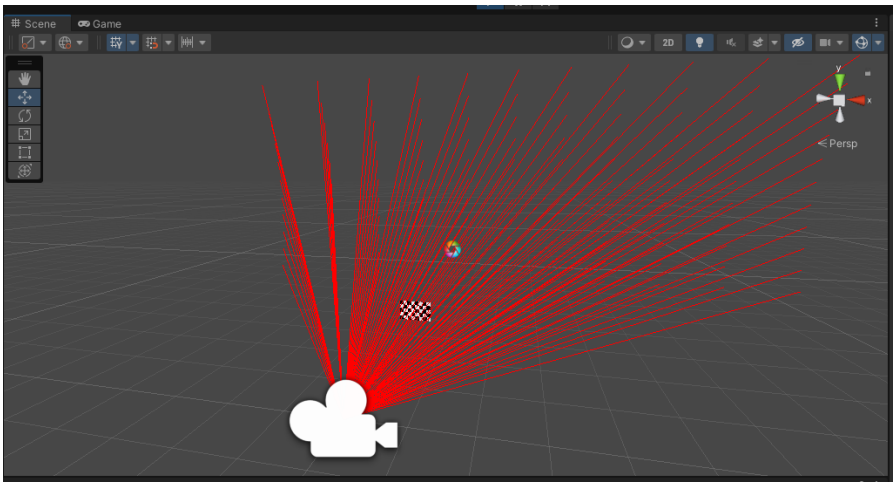

Abbildung 3.4: Simulation eines Lidars in Unity durch Raycasts.

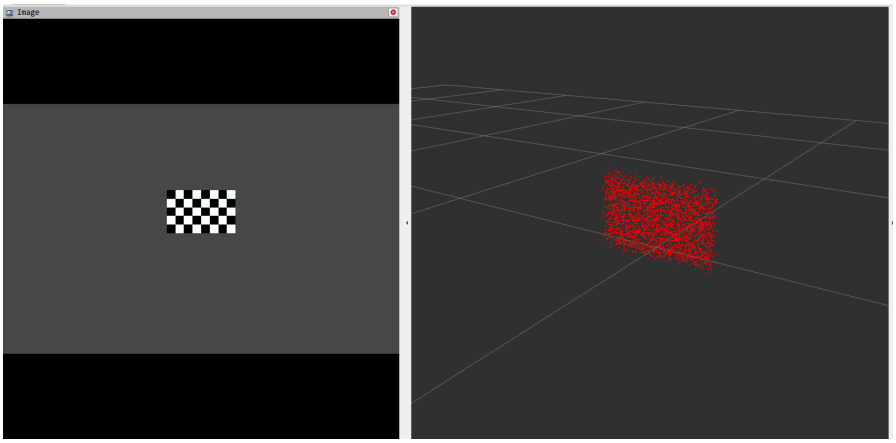

Abbildung 3.5: Visualisierung der simulierten Sensoren. Es wurde hier nur das Schachbrett modelliert ohne Hintergrundobjekte oder ähnliches.

Folgende Aspekte wurden unter anderem in der simulierten Unity-Szene umgesetzt:

Für die Kamera:

- Bereitstellung der Kameradaten im exakt gleichen Format wie die Basler Kamera oder die Allied Vision Kamera. (`sensor_msgs/Image`)
- Simulieren einer radialen/tangentialen Verzeichnung durch durch Unity bereitgestellte Post-Processing Effekte
- Möglichkeit, mehrere Kameras in derselben Szene mit verschiedenen Positionen zu simulieren
- Einstellbare Auflösung und intrinsiche Parameter, sodass die Kameras den Echtweltkameras nachgestellt werden können

Für den Lidar:

- Bereitstellung der Lidardaten im exakt gleichen Format wie der Blickfeld Lidar (`sensor_msgs/PointCloud2`)
- Generieren einer Punktwolke durch multiple Raycasts in der 3D-Szene (siehe Abbildung 3.4)
- Einstellbare Auflösung und Blickwinkel, sodass der Lidar den Echtweltlidar nachstellen kann

- Einstellbare ”Unsicherheit” der einzelnen 3D-Punkte

3.2 Berechnung der Parameter

Wie in Kapitel 2 besprochen wurde, kann eine Fusion von einer Punktwolke und einem Kamerabild nur durchgeführt werden, wenn die intrinsischen und extrinsischen Parameter der Kamera bekannt sind. Im Folgenden wird erklärt, wie diese Parameter in dieser Arbeit berechnet wurden. Es ist hierbei egal, ob die Daten von den realen Geräten stammen oder der Simulation, da die genannten Konzepte auf beide gleichermaßen zutreffen. Unterschiede zwischen den Ergebnissen der Realwelt und Simulation werden in Kapitel 4 diskutiert.

3.2.1 Intrinsische Parameter

Die Heuristik, die die intrinsischen Parameter einer Kamera (inklusive Kameramatrix und Verzeichnungsparameter) schätzt, wird *Kalibrierung* genannt. Ein kalibriertes Kamerabild ist demnach ein Bild, aus der die Verzeichnung basierend auf den Verzeichnungsparametern, neutralisiert wurde und dessen Kameramatrix bekannt ist. Das (OpenCV) Framework stellt von sich aus Funktionen bereit, die eine Kalibrierung durchführen. Die Methodik, mit der diese Kalibrierung durchgeführt wird, basiert hierbei auf (Zhang (2000)). Hierbei wird der Heuristik eine Menge an Bildern der Kameras mitgegeben, in der ein Schachbrett in verschiedenen Blickwinkeln zu sehen ist. Da bekannt ist, dass das tatsächliche Schachbrett-Objekt in Wirklichkeit komplett quadratisch ist, kann aufgrund der Unregelmäßigkeiten im Bild auf die intrinsischen Parameter der Kamera geschlossen werden. Wie genau dieser Prozess abläuft wird in (Zhang (2000)) und (Kaehler und Bradski (2016) ab S. 665) detaillierter geschildert. Für die Zwecke in dieser Arbeit kann die Kalibrierung als Black-Box betrachtet werden, die basierend auf den Kameradaten eine Schätzung für die intrinsischen Werte der Kamera liefert. Es gibt aber auch einige Punkte, die zu beachten sind, um die Qualität der Kalibrierung zu erhöhen, welche in Kapitel 4 besprochen werden.

Das ROS-Framework stellt standardmäßig ein ROS-Package bereit, welches diese Kalibrierung durchführt. Es erwartet das Bild der Kamera in Form von `sensor_msgs/Image` Nachrichten und sucht nach Schachbrettern in diesem Bild, um die Kamera zu kalibrieren. Es handelt sich um das *Camera Calibration* Package (siehe `http:`

//wiki.ros.org/camera_calibration, zuletzt aufgerufen: 15.11.2022). Das Ergebnis der Kalibrierung sind die Verzeichnungsparameter und die Kameramatrix, die die intrinsischen Parameter der Kamera darstellen.

Es gibt zusätzlich ein ROS-Package, welches die Verzeichnung eines Bildes basierend auf den Verzeichnungsparametern, wie in Kapitel 2.2.4 besprochen wurde, neutralisiert. Es handelt sich hierbei um das *Image Processing* Package (siehe http://wiki.ros.org/image_proc, zuletzt aufgerufen: 15.11.2022). Wenn dieses Package mit einem Kamerabild und dazu korrespondierenden intrinsischen Parametern der Kamera in Form von `sensor_msgs/CameraInfo` Messages versorgt wird, dann stellt das Package das Bild der Kamera mit neutralisierter Verzeichnung bereit.

Tatsächlich ist es so, dass die meisten Kameras, die einen ROS-Treiber bereit stellen, intern diese beiden Packages verwenden, um eine Kalibrierung durchzuführen und die Verzeichnung des Bildes neutralisieren, darunter auch die Basler und Allied Vision Kamera. Nach einer Kalibrierung können diese Kameras dadurch automatisch so konfiguriert werden, sodass das Bild direkt unverzeichnet vorliegt.

3.2.2 Extrinsische Parameter

Neben den intrinsischen Parametern müssen auch die extrinsischen Parameter zwischen dem Kamerakoordinatensystem und dem Lidarkoordinatensystem herausgefunden werden. Das Lidarkoordinatensystem fungiert hier als Referenzkoordinatensystem und gilt als "Weltkoordinatensystem". Es gibt hierfür zwei Möglichkeiten.

- Die Abstände zwischen den fixierten Geräten messen und direkt ablesen.
- Die extrinsischen Parameter mit dem "Perspective-N-Point" Algorithmus basierend auf 2D-3D Korrespondenzen berechnen.

Der erste Punkt bietet sich an, wenn die Geräte untereinander von der Position her so fixiert sind, sodass ihre Koordinatensysteme alle dieselbe Ausrichtung haben (parallel) und so nur die Translation untereinander abgelesen werden muss (wie es in dieser Arbeit der Fall ist). Ansonsten kann theoretisch auch die Orientierung nachgemessen werden, macht das Ergebnis aber fehleranfälliger. Für eine Simulation ist hierfür natürlich kein Messen erforderlich, da die absoluten Positionen und Orientierungen aller Geräte direkt in Unity abgelesen werden können.

Der zweite Punkt stellt eine generalisierte Form dar und basiert allein auf den Sensordaten, ohne von vornherein etwas vom Aufbau zu wissen. In (Kaehler und Bradski (2016) S. 701) wird hierfür das Konzept des *Perspective N-Point Problems* eingeführt (siehe Abbildung 3.5)

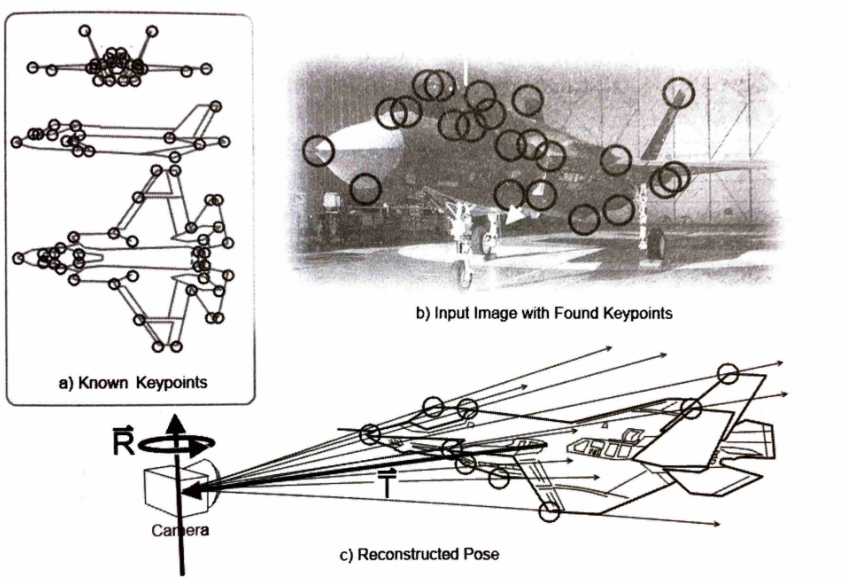

Abbildung 3.6: Perspective N-Point (PNP) Problem Konzept. Quelle: (Kaehler und Bradski (2016), S. 701, Abbildung 19.4)

Wenn einzelne 3D-Punkte eines Objektes vorliegen und dieselben Punkte in einem Kamerabild, dessen intrinsische Parameter bekannt sind, lokalisiert werden können, kann auf die extrinsische Beziehung zwischen dem Objektkoordinatensystem und dem Kamerakoordinatensystem geschlossen werden. Da durch die Kameramatrix die Geometrie der Szene durch das Bild korrekt wiedergegeben wird, können durch die Bildpositionen "inverse Strahlen" in den Weltkoordinatensystem aufgestellt werden, die zunächst die Eigenschaften des Kamerakoordinatensystems erfüllen. Die 3D-Punkte des Objektes liegen bereits im Weltkoordinatensystem vor. Nun kann duch den Abstand der Strahlen, die durch die einzelnen Bildpositionen erstellt wurden, zu den dazugehörigen 3D-Objekt Punkten eine Kostenfunktion aufgestellt werden, die durch die Summe all dieser Abstände zwischen den Strahlen und der 3D-Punkte des Objektes definiert ist. Wenn all diese inversen Strahlen nun allesamt affin verschoben und rotiert werden, sodass sie die tatsächliche Position der Kamera im Weltkoordinatensystem darstellen, würde diese Kostenfunktion gegen null gehen. Die extrinsischen Parameter zwischen dem Kamera- und

Weltkoordinatensystem kann also ermittelt werden, indem diese Kostenfunktion, durch das Modifizieren der extrinsischen Parameter reduziert wird.

Dies wird mithilfe eines speziellen *Least Squares* Algorithmus erreicht und zwar dem *Levenberg-Marquardt-Algorithmus*, siehe (Levenberg (1944)).

Das (OpenCV) Framework stellt hierfür eine Funktion bereit, die genau dieses Konzept implementiert, und zwar die `solvePnPRefineLM()` Funktion (siehe (OpenCVDocs)), die in dieser Arbeit genutzt wird.

Wie auch bei der Kamerakalibrierung kann dieser Algorithmus als Black-Box betrachtet werden, der aufgrund eines Kamerabildes, ihrer intrinsischen Parameter, einer Menge an 3D-Punkten und die zugehörigen Bildpositionen dieser Punkte im Kamerabild die extrinsischen Parameter der Kamera berechnet. Auch hier gibt es jedoch wieder einige Punkte, die zu beachten sind, um die Qualität des Ergebnisses zu erhöhen, was in Kapitel 4 besprochen wird.

3.3 Systemaufbau

Das ROS-Package, welches im Rahmen dieser Arbeit erstellt wurde, beinhaltet vier Programme (ROS-Nodes), die jeweils verschiedene Aufgaben im Fusionsprozess erfüllen.

Darunter:

- camInfoPublisher
- pairPicker
- extrinsicCalibrator
- fuser

Diese können im ”src” Ordner des git-Repositories gefunden werden.

Der Zusammenspiel dieser Programme im Fusionsprozess wird im Systemdiagramm in Abbildung 3.7 gezeigt:

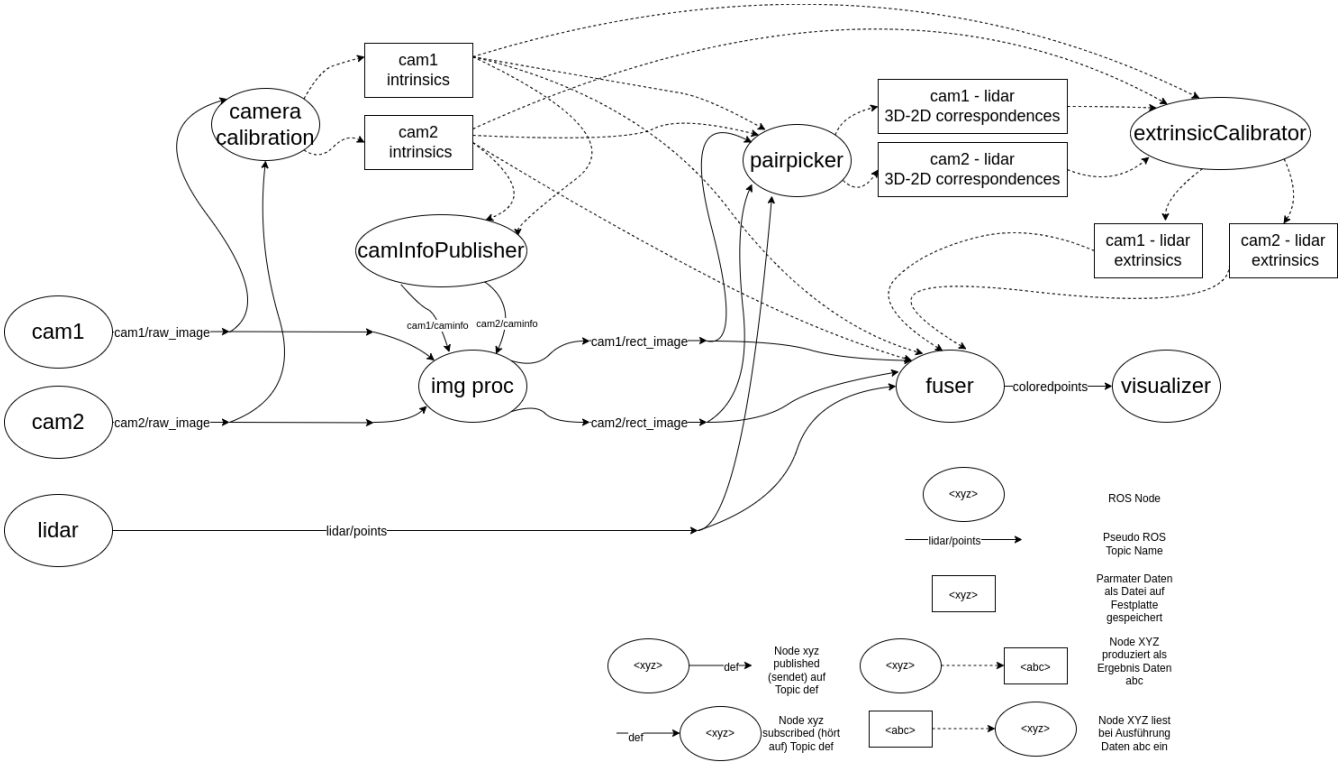

Abbildung 3.7: Systemdiagramm der Fusionsarchitektur

Im folgenden eine Beschreibung des Fusionsprozesses unter Einbezug der ROS-Nodes.

In diesem Beispiel wird von zwei simulierten Kameras un Unity (die auch eine Verzeichnung simulieren) ausgegangen, die die Verzeichnung der Kamerabilder nicht automatisch korrigieren. Diese Kameras veröffentlichen ihre Kameras also in verzeichneter Form. Konventionsmäßig wird ein solches Bild in ROS ”raw” genannt. Dieses ”rohe” Bild wird an den ROS Camera Calibration Node gegeben, um die intrinsischen Parameter der Kameras zu schätzen. Da es sich bei den Unity Kameras nicht um vollwertige Kamera-Treiber handelt, die das Bild automatisch korrigieren, muss dies manuell durchgeführt werden. Das Ergebnis der Kalibrierung liegt in Form einer .yaml Datei vor, die dem *camInfo-Publisher* übergeben wird. Hierbei handelt es sich um einen simplen Node, der diese Datei einfach nur ausliest und kontinuierlich über einen ROS Topic veröffentlicht. Dies ist notwendig, da der ROS Image Processing Node die intrinsischen Parameter als ROS Message erwartet, um das Bild zu korrigieren. Dieser Schritt wird von den Realwelt Geräten automatisch ausgeführt, wenn sie mit dem ROS Camera Calibration Node kalibriert werden.

Der restliche Teil ist sowohl bei simulierten als auch echten Geräten gleich. Das Bild, das jetzt in korrigierter Form ohne Verzeichnung vorliegt, wird konventionsmäßig ”rectified” genannt. Nun müssen die extrinsischen Parameter der Kamera bestimmt werden. Hierfür werden neben dem korrigiertem Bild und den intrinsischen Parametern aber noch genügend (mindestens drei) 3D-2D Korrespondenzen benötigt. Diese können mit dem *pairpicker* Node erstellt werden. Mit einem gegebenem Kamerabild, ihren intrinsischen Parametern und einer Punktwolke vom Lidar kann ein Nutzer manuell die Bildkoordinaten von ausgewählten 3D-Punkten der Punktwolke im Kamerabild ablesen und die jeweiligen Werte in einer Datei abspeichern. Diese Dateien stellen die 3D-2D-Korrespondenzen dar, mit denen der *extrinsicCalibrator* den zuvor erwähnten Levenberg–Marquardt Algorithmus ausführen kann, der von OpenCV implementiert ist, um die extrinsischen Parameter der Kamera zu berechnen.

Die intrinsischen und extrinsischen Parameter der Kamera, das korrigierte Bild der Kamera und die Punktwolke des Lidars werden dann dem *fuser* Node übergeben, der die Fusion durchführt und als Ergebnis die Punktwolke zurückgibt, in der jedem Punkt der Punktwolke eine Farbe basierend auf dem Kamerabild zugeorndet wurde.

Da der Standardvisualisierer von ROS, RViz, jedoch keine Möglichkeit hat, die benutzerdefinierte farbige Punktwolke zu visualisieren, wurde noch eine spezielle Szene in Unity bereitgestellt, die solche Punktwolken visualisieren kann. Sie kann als Unity Szene im Unity-Projekt-Repository dieser Arbeit gefunden werden.

4 Evaluierung der Ergebnisse

In diesem Kapitel werden die Ergebnisse dieser Arbeit evaluiert. Hierbei wird unter anderem das Verhalten der Simulation mit dem Verhalten der Echtweltgeräte basierend auf der Szene aus Abbildung 3.1 und 3.2 verglichen, die in der Simulation nachgestellt wurde (siehe Abbildung 3.5), verglichen. Außerdem wird jeweils die Qualität der intrinsischen und extrinsischen Kalibrierung evaluiert. Daraufhin wird das Gesamtergebnis der Fusion, die farbige Punktwolke, die in verschiedenen Szenen generiert wurde, evaluiert.

4.1 Evaluierung der simulierten und realen Geräte

Das Ziel der Simulation in dieser Arbeit ist es, die Daten der realen Sensorgeräte möglichst genau nachzustellen.

4.1.1 Simulation der Kameras

Um eine Kamera in der Simulation nachstellen zu können müssen hierfür folgende Aspekte berücksichtigt werden:

- Die (native) Auflösung der Kamera
- Der Bildwinkel der Kamera
- Das Verzeichnungsverhalten der Kamera

Sowohl die Auflösung und der Bildwinkel einer Kamera können un Unity einfach eingestellt werden. Die native Auflösung der Kamera wird direkt vom Treiber bereitgestellt und der Bildwinkel kann mithilfe ihrer intrinsischen Parameter berechnet werden. Die

Verzeichnung der Kamera kann jedoch nicht aufgrund ihrer Verzeichnungsparameter direkt nachgestellt werden, da die Simulation einer Verzeichnung in Unity diskrete Skalierungsparameter statt genauer Verzeichnungsparameter nutzt und außerdem nur radiale Verzeichnung aufgrund einer Linse simulieren kann. Für diesen Zweck wurde die simulierte Kamera mit einer geringfügigen Verzeichnung simuliert, die in etwa der Verzeichnung der Basler-Kamera entspricht.

Jeweils die simulierte und die reale Kamera wurden gleichzeitig in Betrieb genommen und gleichzeitig in RViz visualisiert, um ihren Output zu vergleichen. Das Ergebnis kann in Abbildung 4.1 gesehen werden.

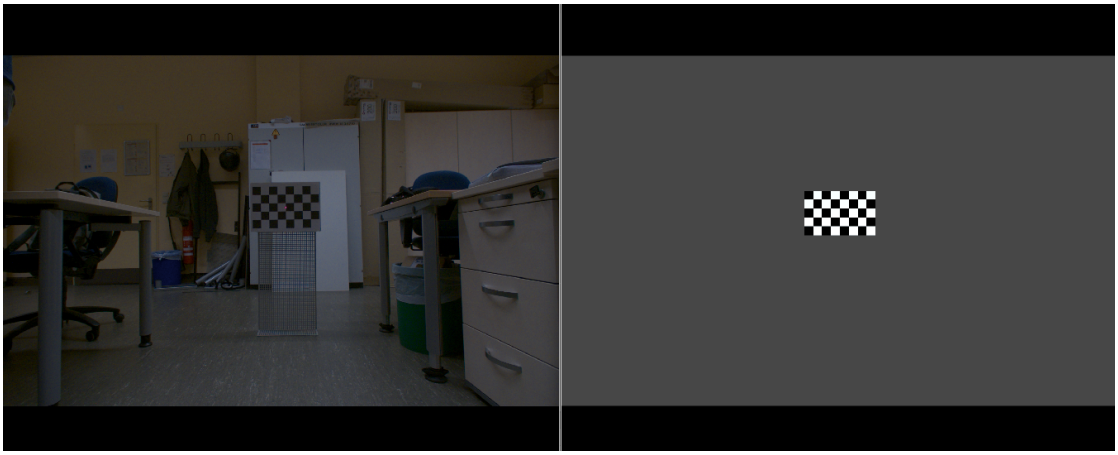

Abbildung 4.1: Vergleich der simulierten Kamera und der realen Kamera.

Man sieht, dass das Schachbrett in beiden Bildern nahezu die gleiche Größe hat. Wie bereits in Kapitel 3.1.1 erwähnt wurde, ist das 8x5 Schachbrett mit einer Quadratgröße von 5cm sowohl in Simulation als auch in Realwelt genau 2m von den Geräten entfernt. Der Abstand im realen Aufbau wurde hiermit mit einem Bosch PLR 50 C Entfernungsmesser bestätigt (siehe Abbildung 4.2)

Abbildung 4.2: Bestätigung des Abstands von der Kamera zum Schachbrett durch einen Entfernungsmesser.

Die Eignung der simulierten Kamera für die Nachstellung der Basler Kamera ist in dieser Arbeit hiermit gegeben.

4.1.2 Simulation des Lidars

Um einen Lidar in der Simulation nachstellen zu können müssen hierfür folgende Aspekte berücksichtigt werden:

- Die Auflösung des Lidars
- Der Bildwinkel des Lidars
- Die Unsicherheit des Lidars

Ähnlich wie bei einer Kamera kann auch bei einem Lidar eine Auflösung und ein Bildwinkel eingestellt werden. Die Auflösung definiert hierbei, wieviele jeweils vertikale und horizontale Scanlinien benutzt werden sollen, um die Szene abzutasten. Durch den Blickwinkel wird hierbei beeinflusst, welche Teile der Szene abgetastet werden. Dadurch wird auch indirekt die Dichte der Abtastung für den Bereich der Szene festgelegt.

Die 3D-Punkte, die durch den realen Lidar bereitgestellt werden, obliegen hierbei einem Rauschen, was dazu führt, das selbst wenn der Lidar fixiert ist und still steht und keine Änderungen in der Szene auftreten die einzelnen Scans leicht unterschiedliche Koordinaten mit jedem Scan messen.

Beim Blickfeld Lidar, der in dieser Arbeit verwendet wurde, wurde festgestellt, dass gemessene 3D-Punkte von einem Objekt, welches etwa zwei Meter vom Lidar entfernt ist, eine Unsicherheit von etwa 4cm haben. Gemessene 3D-Punkte von einem Objekt, welches etwa 4.5 Meter vom Lidar entfernt ist, haben eine Unsicherheit von etwa 5mm, also ist die Unsicherheit größer, je näher das Objekt am Lidar ist.

Neben diskreten 3D-Punkten liefert der Blickfeld Lidar mit jeder Messung außerdem einen Intensitätswert, der von der jeweils reflektierten Oberfläche gemessen wurde. In der Szene, die in Abbildung 3.1 und 3.2 gezeigt wird, werden vom Blickfeld Lidar verschiedenen Instensitäten für die weißen und schwarzen Bereiche des Schachbretts wahrgenommen. (siehe Abbildung 4.3)

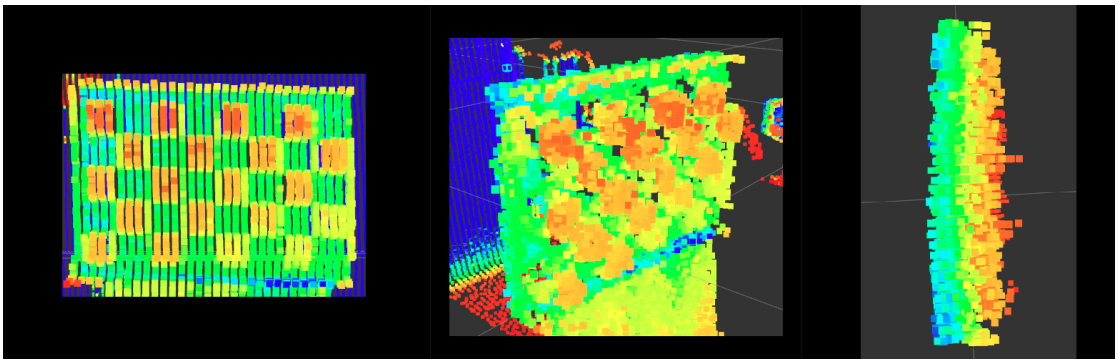

Abbildung 4.3: Messung des Schachbretts mit dem Blickfeld Lidar.

Es lässt sich erkennen, dass die Bereiche der dunklen Felder durchschnittlich näher zum Lidar hin gemessen wurden, als die weißen Felder. Durchschnittlich weichen die Positionen von Punkten mir niedrigerer und höherer Intensität etwa 6cm voneinander ab.

Der Lidar, der in Unity simuliert wurde hat eine einstellbare Auflösung und Blickwinkel. Es kann jedoch nur eine generelle globale Unsicherheit eingestellt werden, anders als eine entfernungsbasierte Unsicherheit, wie es beim Blickfeld Lidar der Fall ist. Außerdem wird keine Intensität aufgrund der Oberflächen simuliert. Dies ist in Abbildung 3.5 sichtbar. In RViz wird mit der Farbe von visualisierten Punkten der jeweilige Intensitätswert signalisiert, die der Lidar gemessen hat, der simulierte Unity-Lidar liefert hierbei immer einen Dummy-Wert von 0.

Um den simulierten Lidar mit dem Blickfeld-Lidar zu vergleichen wurde ähnlich wie in Kapitel 4.1.1 die Punktwolke des realen Lidars gleichzeitig mit der Punktwolke des simulierten Lidars aufgenommen, der die gleiche Szene nachstellt, wie die Szene, die der Blickfeld-Lidar abtastet. Der simulierte Lidar wurde hierfür mit derselben Auflösung und demselben Blickwinkel wie der Blickfeld-Lidar eingestellt. Das Ergebnis ist in Abbildung 4.4 zu sehen.

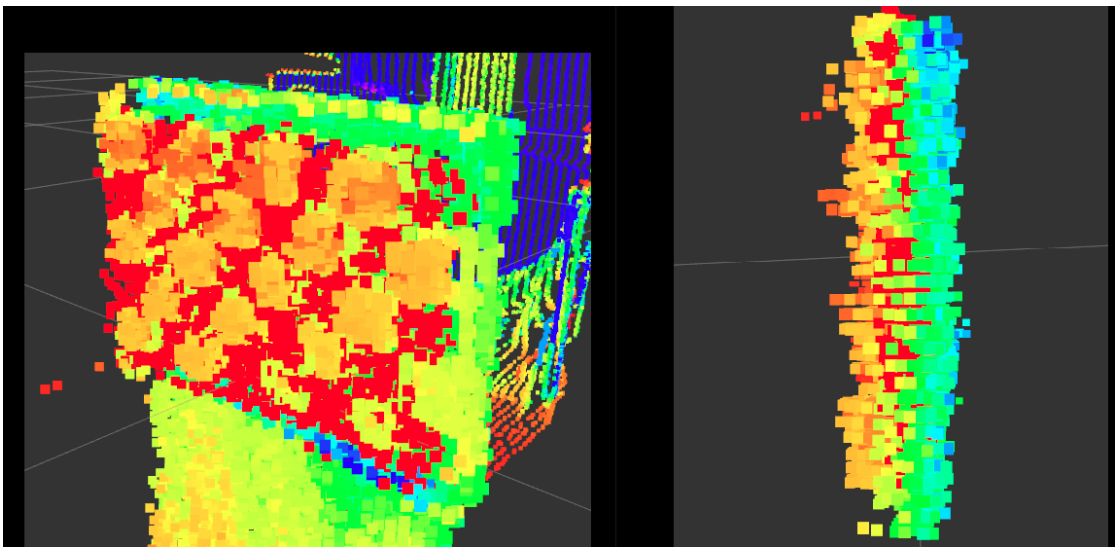

Abbildung 4.4: Vergleich des simulierten Lidars mit dem Blickfeld Lidar. Die rot gezeigten Punkte stammen vom simulierten Lidar und die farbigen vom realen Blickfeld Lidar

Man sieht, dass die detektierten Punkte beider Geräte für das Schachbrett überlappen. Der simulierte Lidar (rote Punkte) liefert hierbei jedoch nur immer denselben Dummy-Wert für eine Intsensität (0) und auch der Fakt, dass dunklen Bereiche des Schachbretts beim Blickfeld-Lidar etwas näher detektiert wurden als die hellen Bereiche, wird von der

Simulation nicht berücksichtigt. Dennoch stellt der Unity-Lidar für die Zwecke dieser Arbeit eine geeignete Lösung dar, um den realen Blickfeld-Lidar zu simulieren.

Ein letzter Punkt, der noch beachtet werden muss, ist die Abtastrate des Lidars. Da der reale Blickfeld-Lidar die Szene abtastet, indem ein eingebauter Spiegel den Laserstrahl in verschiedene Richtungen reflektiert, bedeutet dies, dass die Szene sequentiell, also zeitversetzt, abgetastet wird. Je höher die eingestellte Auflösung des Lidars ist, dauert es umso länger, die Intensitäten der Laserstrahlen für eine vollständige Messung zurückzulesen. Wenn sich die Objekte in dieser Szene in dieser Zeit verändern oder der Lidar bewegt wird, kann es zu einer Verzerrung der Messungen kommen, ähnlich wie er bei Kameras eintritt, wenn sie unter dem *Rolling Shutter Effect* leiden, was in Kapitel 1.3 angesprochen wurde.

Beim Blickfeld Lidar dauert das vollständige Abtasten der Szene bei einer Auflösung von 399x175 Scanlinien und einem Blickfeld von 30 Grad x 70 Grad etwa zwei Sekunden. Hierbei kann eine deutliche Verzerrung auftreten, beispielsweise, wenn Personen durch das Bild laufen (siehe Abbildung 4.5)

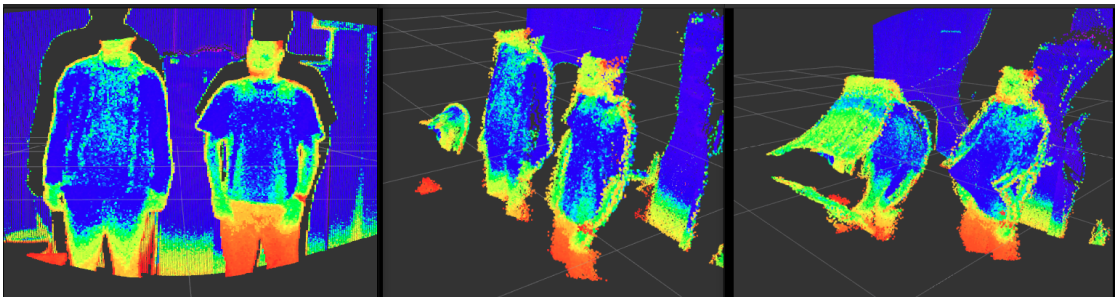

Abbildung 4.5: Verzerrung des Lidars bei hoher Auflösung: Links und Mitte: Messung des Lidars von zwei Personen, die stillstehen. Rechts: Messung des Lidars von zwei Personen, die sich während der Messung nach vorne bewegen

Diese Art der Verzerrung wird ebenfalls nicht vom Unitylidar berücksichtigt, da die Punkte des simulierten Lidars allesamt simultan aufgenommen werden. Falls eine Anwendung diese Art von Verzerrung modellieren möchte, müsste dies berücksichtigt werden. In dieser Arbeit ist dies aber eher irrelevant.

4.2 Evaluierung der intrinsischen Kalibrierung

Die Qualität der intrinsischen Kalibrierung hat einen direkten Einfluss auf die Korrektheit der Fusion, da durch sie die Geometrie der Kamera beschrieben wird. In dieser Arbeit kann die ungefähre Korrektheit einer Kameramatrix abgelesen werden, indem ein Lidar ohne Unsicherheit simuliert wird, dessen Punkte dann in das Kamerabild aufgrund der intrinsischen Parmameter abgebildet werden. Da die extrinsischen Parameter in der Simulation genau bekannt sind, können mögliche Abweichungen also nur durch inkorrekte intrinsische Parameter entstehen. Abbildung 4.6 zeigt hierbei ein derartiges Beispiel.

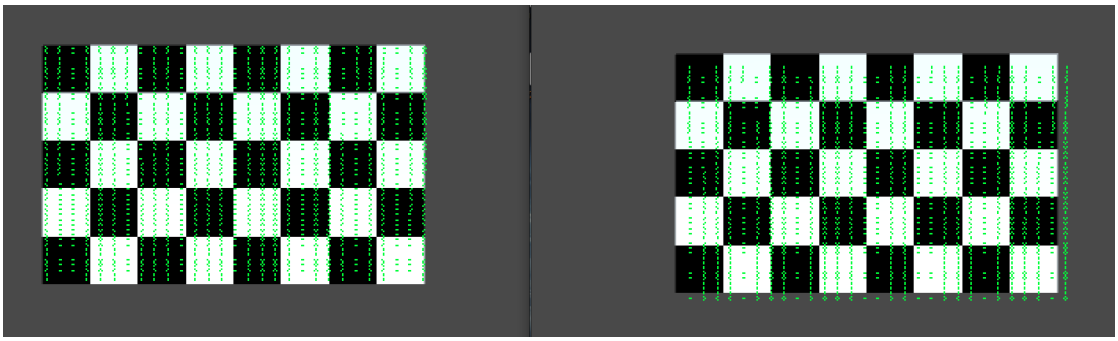

Abbildung 4.6: Vergleich der Abbildung von 3D-Punkten auf das Kamerabild mit guten intrinsischen Parametern (links) und schlechten (rechts)

Man sieht, dass mit guten intrinsischen Parametern alle 3D-Punkte auf Pixel im Bild abgebildet werden, in denen das Schachbrett zu sehen ist. Mit schlechten intrinsischen Parametern hingegen, erkennt man, dass eine geringfügige Abweichung auftritt, da die Geometrie der Kamera nicht ganz korrekt wiedergegeben wird.

Die Qualität einer Kalibrierung ist hierbei grundsätzlich von drei Punkten abhängig.

- Die Anzahl an Samples
- Abwechslungsreiche Positionen und Orientierungen des Schachbretts in den Samples
- Samples, die möglichst nah an der Kamera sind

Wie gesagt, wird für die Bestimmung der intrinsischen Parameter das ROS Camera-Calibration Node verwendet, welches auf der Heuristik von OpenCV basiert. Hierfür

werden multiple Kamerabilder benötigt, in denen ein Schachbrett zu sehen ist. Je abwechslungsreicher dabei die Orientierung und Position des Schachbretts ist, desto besser fällt die Kalibrierung aus. Dabei haben Samples, bei denen das Schachbrett sehr nah an der Kamera ist, den größten Einfluss. Für die Kalibrierung sollte deswegen das Schachbrett je in einer Pose aufgenommen werden in der das Schachbrett parallel, leicht nach rechts, links, oben und unten geneigt ist und das noch mal von naher, mittlerer und ferner Position. (siehe Abbildung 4.7)

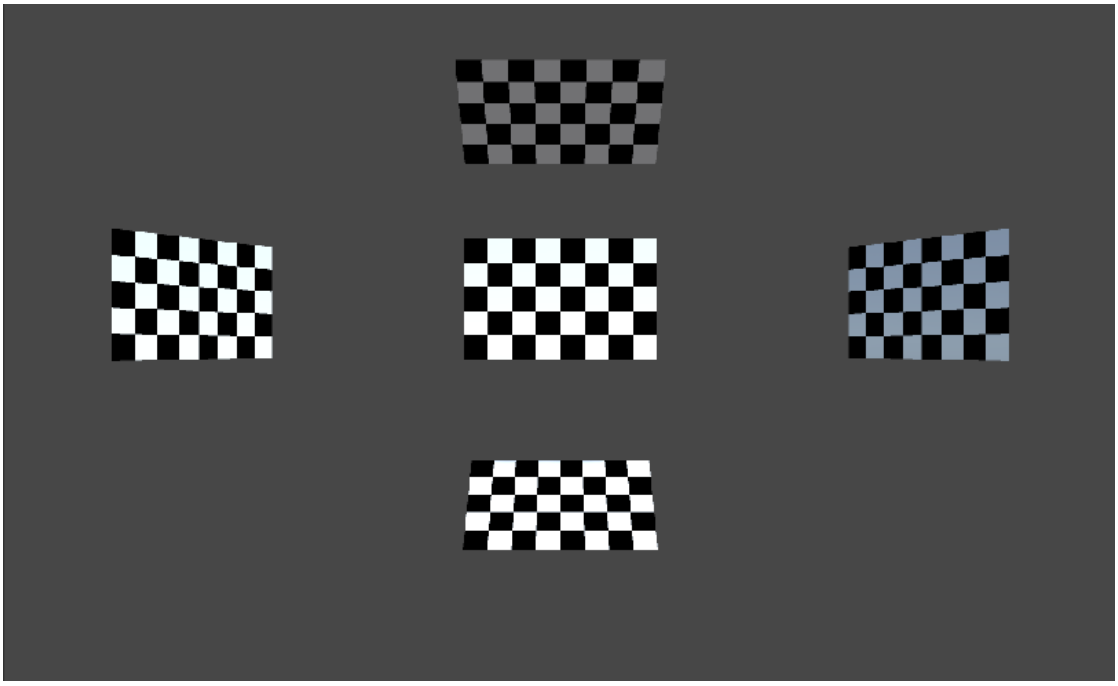

Abbildung 4.7: Optimale Ausrichtungen des Schachbretts für eine gute Kamerakalibrierung

4.3 Evaluierung der extrinsischen Kalibrierung

Wie bereits in Kapitel 3.2.2 gesagt, gibt es zwei Möglichkeiten, die extrinsischen Parameter der Kamera zu bestimmen, entweder durch direktes Messen oder durch Berechnung basierend auf einem Least-Squares Algorithmus.

Falls, wie in dieser Arbeit, alle Koordinatensysteme der Geräte parallel fixiert sind und die Baudimensionen bekannt sind, ist es besser, diese direkt abzulesen und einzutragen. In Systemen, in der die Baudimensionen jedoch nicht bekannt sind und verschiedene

Orientierungen zwischen den Koordinatensystemen herrschen, ist dies jedoch keine Möglichkeit.

Die Ergebnisse, die der Levenberg Marquardt-Algorithmus liefert ist nämlich abhängig von der Kameramatrix, den Verzeichnungsparametern, der 3D-Punktwolke und den 2D-3D Korrespondenzen zwischen der Punktwolke und dem Kamerabild. Durch jeden dieser Punkte gelangen Fehler in den Prozess:

- Wenn die Kameramatrix eine schlechte Qualität hat, dann werden 3D-Punkt möglicherweise nicht korrekt auf das Bild abgebildet. Außerdem hat dies auch einen Einfluss auf den Informationsgehalt der 3D-2D Korrespondenzen.
- Wenn die Verzeichnungsparameter schlecht geschätzt sind, dann ist die Qualität der Rekonstruktion der Geometrie der Szene dementsprechend schlechter, was zu ungenaueren Ergebnissen führt
- Die Punkte der Punktwolke, die durch einen realen Lidar geliefert werden haben eine gewisse Unsicherheit. Wenn zu wenig 3D-2D Korrespondenzen vorliegen, kann es sein, dass das globale Minimum der Kostenfunktion für die Berechnung der extrinsischen Parameter stark vom tatsächlichen Zustand abweicht (siehe Abbildung 4.8).

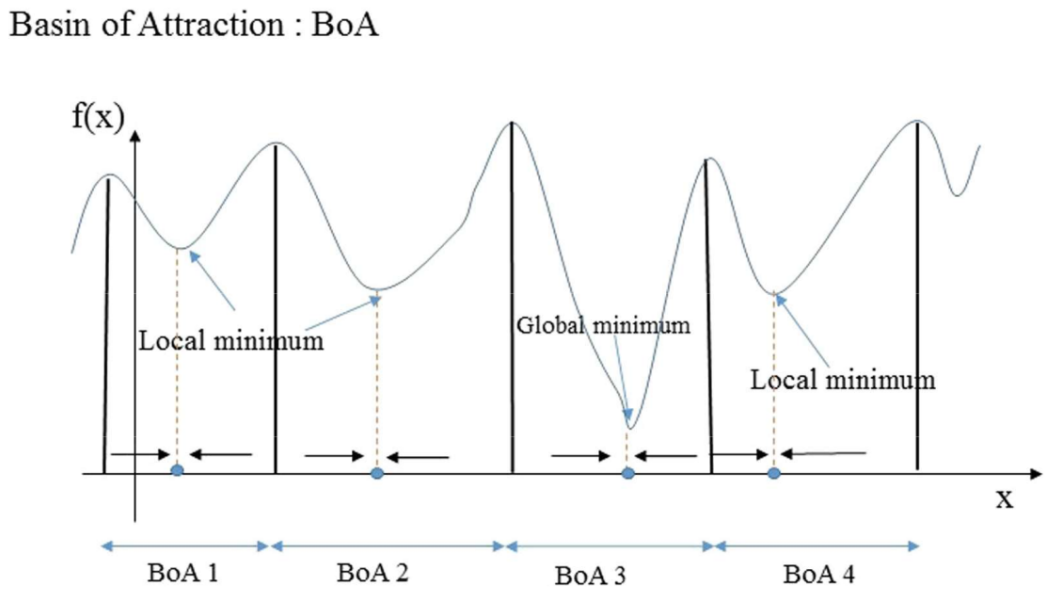

Abbildung 4.8: Annäherung an die extrinsischen Parameter durch das Minimieren einer Kostenfunktion. Das globale Minimum repräsentiert hierbei die tatsächlichen extrinsischen Parameter. Durch die vorher genannten Fehlerquellen kann es jedoch sein, dass das Minimum von den tatsächlichen extrinsischen Parametern abweicht. Quelle: (Bouain u. a. (2020), Abbildung 2)

Um diese Art der extrinsischen Kalibrierung zu evaluieren wurde eine Unity-Simulation mit zwei identischen Kameras und einem Lidar erstellt.

Die beiden Kameras wurden je 4 Meter nebeneinander platziert und aufgrund der gelieferten Messungen wurden mithilfe des in dieser Arbeit entwickelten ROS-Nodes *pairPicker* 3D-2D-Korrespondenzen abgeleitet mit denen dann mit dem *extrinsicCalibrator* die extrinsischen Parameter berechnet wurden. Selbst mit einer Vielzahl an extrahierten 3D-2D-Korrespondenzen wurden Ergebnisse erzielt, in denen die extrinsischen Parameter bis zu 5cm von der tatsächlichen Poisiton verschoben waren. Deswegen wurden in dieser Arbeit die Abstände zwischen den Geräten direkt gemessen.

4.4 Evaluierung der Fusion

Wie in Abbildung 4.6 beispielhaft zu sehen ist, wurde die Fusion durchgeführt, indem die 3D-Punkte auf Basis aller bisher genannten Konzepte auf das Kamerabild abgebildet wurden, um eine Farbeigenschaft für diesen Punkt abzulesen. Das Ergebnis dieser Fusion ist in den folgenden Abbildungen zu sehen.

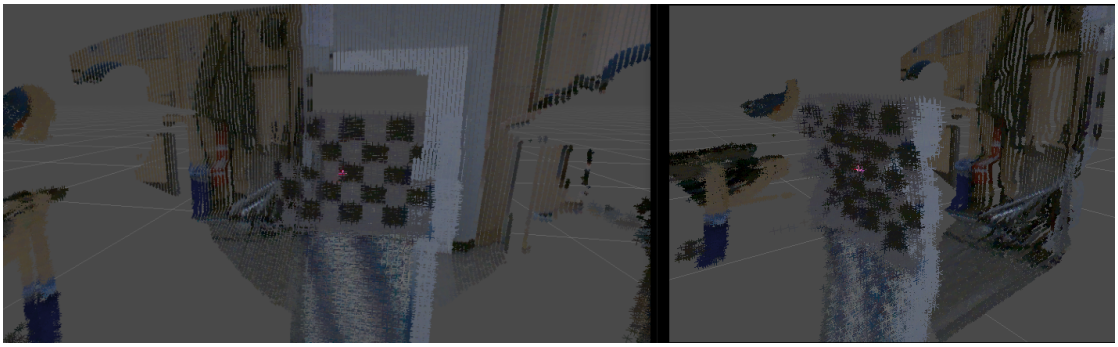

Abbildung 4.9: Ergebnis der Fusion: Lidar-Punktwolke und Basler Kamera

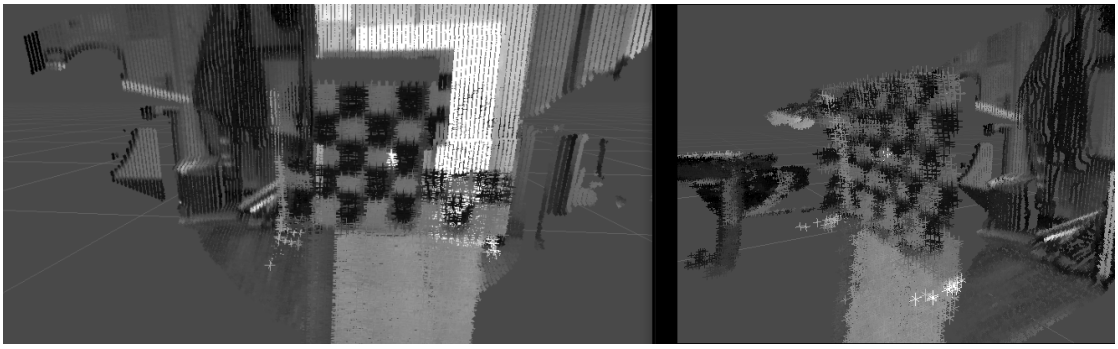

Abbildung 4.10: Ergebnis der Fusion: Lidar-Punktwolke und Allied Vision Kamera

In den Abbildungen 4.9 und 4.10 kann das Ergebnis der Fusionen jeweils mit der Punktwolke und der Basler Kamera und mit der Punktwolke und der Allied Vision Kamera betrachtet werden. Es handelt sich hierbei um den Versuchsaufbau aus Abbildung 3.1 und 3.2. Man sieht, dass die Fusion größtenteils erfolgreich umgesetzt wurde, jedoch ist auch eine leichte räumliche Verschiebung bemerkbar, wodurch einige Teile des Schachbretts auf die hinterliegende Wand abgebildet wurden. Dies ist ein Effekt, der durch die vorher diskutierten Abweichungen und Fehler durch die intrinsische und extrinsische Kalibrierung verursacht wird.

In Abbildung 4.11 wird das Ergebnis der Fusion mit dem simulierten Lidar und der simulierten Kamera gezeigt. Es handelt sich um den Versuchsaufbau, der in Abbildung 3.3, 3.4 und 3.5 gezeigt wird, der den Aufbau in Abbildung 3.1 und 3.2 in der Simulation nachstellt.

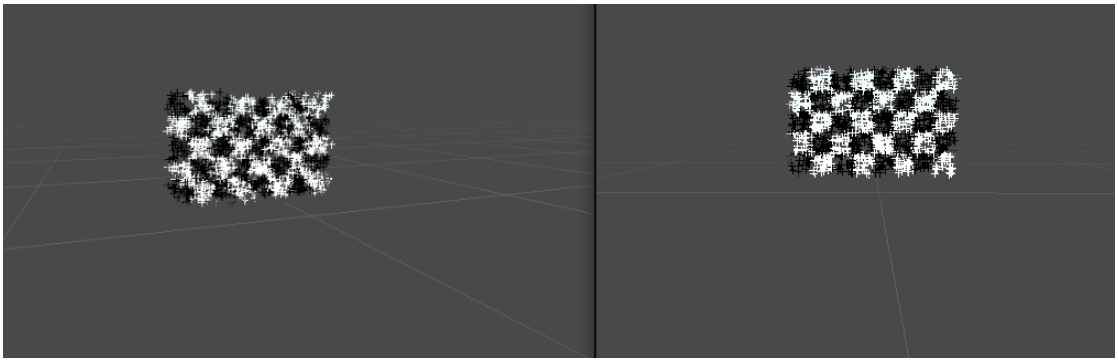

Abbildung 4.11: Ergebnis der Fusion: Simulierte Kamera und simulierter Lidar

In Abbildung 4.11 kann betrachtet werden, dass die Fusion mit den simulierten Geräten gut funktioniert hat.

In Abbildung 4.12 wird ein Versuchsaufbau gezeigt, in der eine Person ein Schachbrett hält.

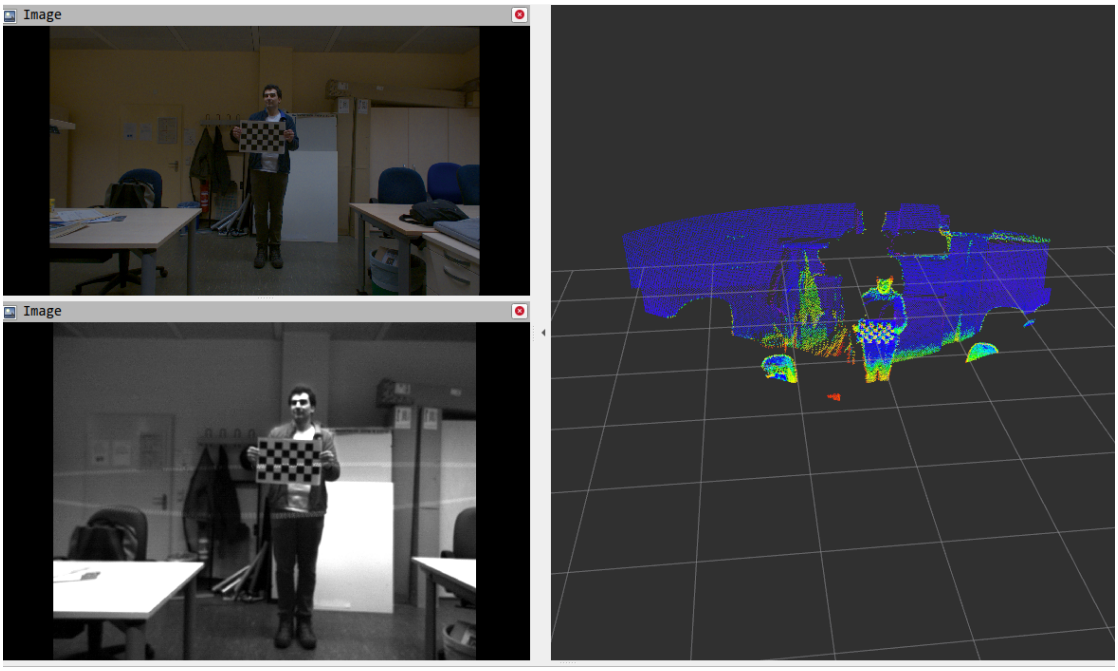

Abbildung 4.12: Versuchsaufbau mit einer Person, die ein Schachbrett hält

Das Ergebnis der Fusion in diesem Aufbau wird in den folgenden Abbildungen gezeigt.

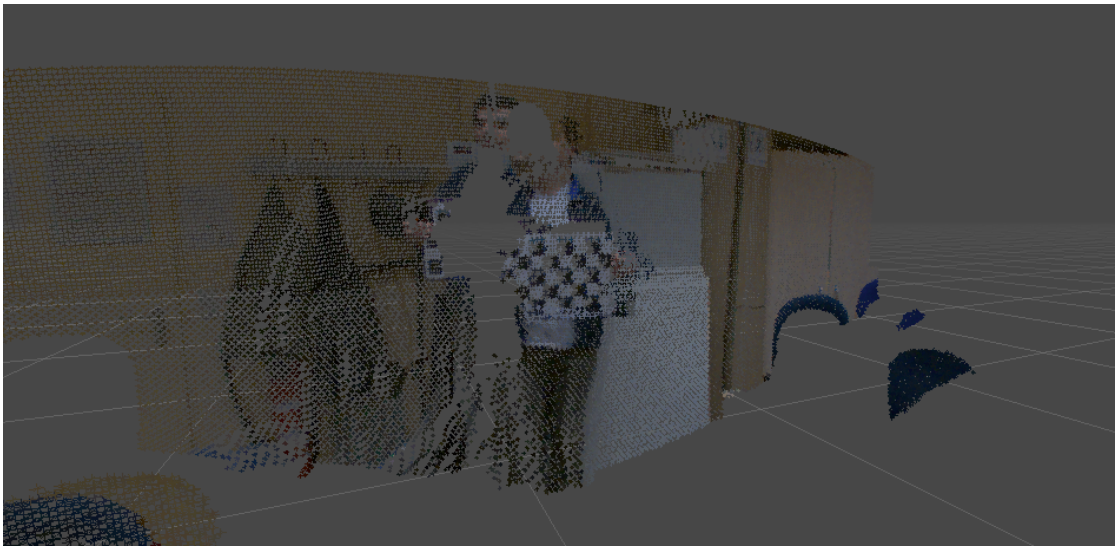

Abbildung 4.13: Ergebnis der Fusion mit der Basler Kamera und der Lidar-Punktwolke im alternativen Versuchsaufbau

Abbildung 4.14: Ergebnis der Fusion mit der Basler Kamera und der Lidar-Punktwolke im alternativen Versuchsaufbau (2)

Abbildung 4.15: Ergebnis der Fusion mit der Allied Vision Kamera und der Lidar-Punktwolke im alternativen Versuchsaufbau

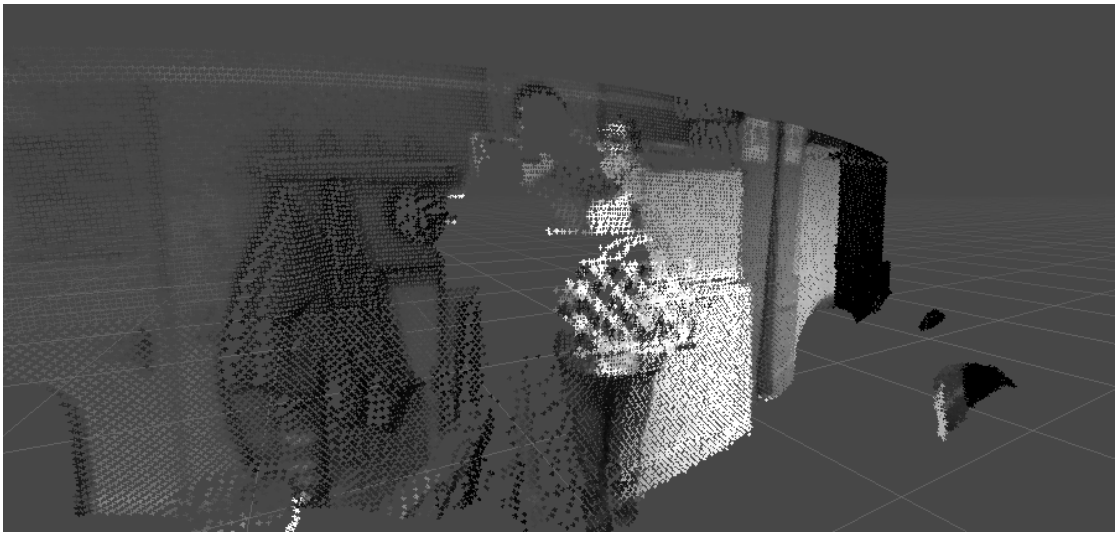

Abbildung 4.16: Ergebnis der Fusion mit der Allied Vision Kamera und der Lidar-Punktwolke im alternativen Versuchsaufbau (2)

In Abbildung 4.13 und 4.14 kann man sehen, dass die Zuordnung der Farben auf die Punktwolke leicht fehlerhaft ist. Teile des Gesichts der Person und der (vom Bild aus) linke Teil des Schachbretts wurden auf die hinterliegende Wand statt auf die korrekte Position abgebildet. Daraus kann man erkennen, dass irgendwo im intrinsischen oder extrinsischen Kalibrierungsprozess eine größere Abweichung/Fehler aufgetreten sein muss.

Derselbe Fehler kann auch in der Fusion der Allied Vision Kamera mit der Punktwolke betrachtet werden, jedoch in deutlich geringerem Ausmaß. Zwar werden offensichtlich einige Bildpixel an die falsche Position abgebildet, aber das Ergebnis ist deutlich besser als die Fusion mit der Basler Kamera. Dies bedeutet, dass die Kalibrierung der intrinsischen und extrinsischen Parameter mit der Allied Vision Kamera besser ausgefallen ist als bei der Basler Kamera.

Zuletzt wurde ein weiterer Versuchsaufbau getestet, der komplett in der Simulation stattfindet. Er ist in Abbildung 4.17 zu sehen.

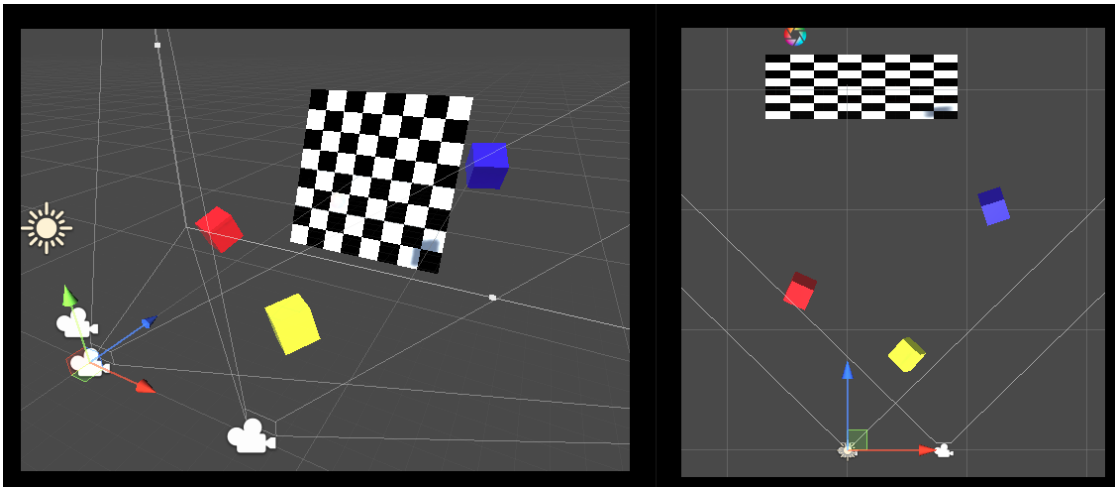

Abbildung 4.17: Versuchsaufbau in Simulation

Hierbei wurden zwei identische Kameras jeweils nebeneinander platziert mit einem Abstand von 4m. Die linke Kamera (Kamera 1) ist an der selben Position wie der Lidar, wobei der Blickwinkel des Lidars etwas kleiner als die der Kameras ist. Außerdem befinden sich in der Szene ein etwas nach hinten geneigtes Schachbrett sowie drei farbige Würfel.

Der simulierte Lidar wurde hierbei ohne Unsicherheit konfiguriert, was bedeutet, dass an jeder Position der 3D-Punkte der Punktwolke auch eine Oberfläche existiert. Außerdem wurden die Kameras ohne Verzeichung simuliert und die extrinsischen Parameter wurden direkt aus der Simulation abgelesen.

Die Messungen der Sensoren können in Abbildung 4.18 betrachtet werden.

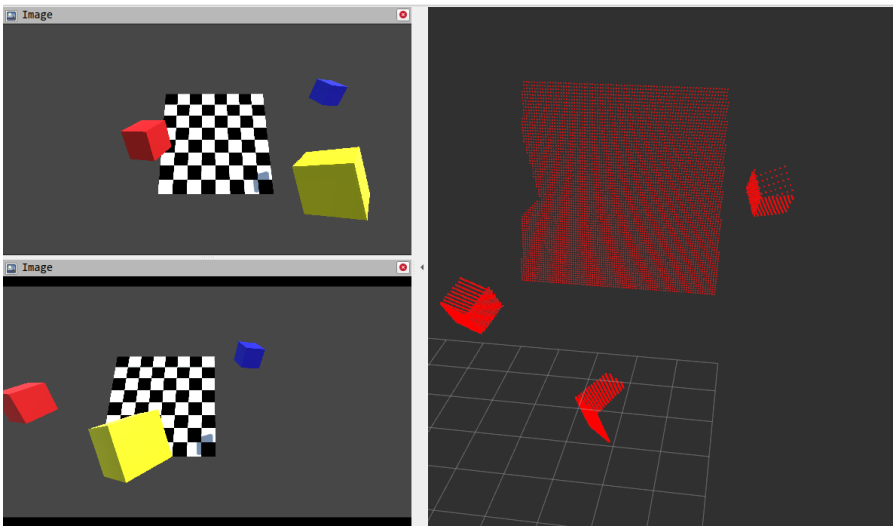

Abbildung 4.18: Messungen der Sensoren im simulierten Versuchsaufbau

In den Abbildungen 4.19 und 4.20 ist das Ergebnis der Fusion beider Kameras mit dem Lidar abgebildet.

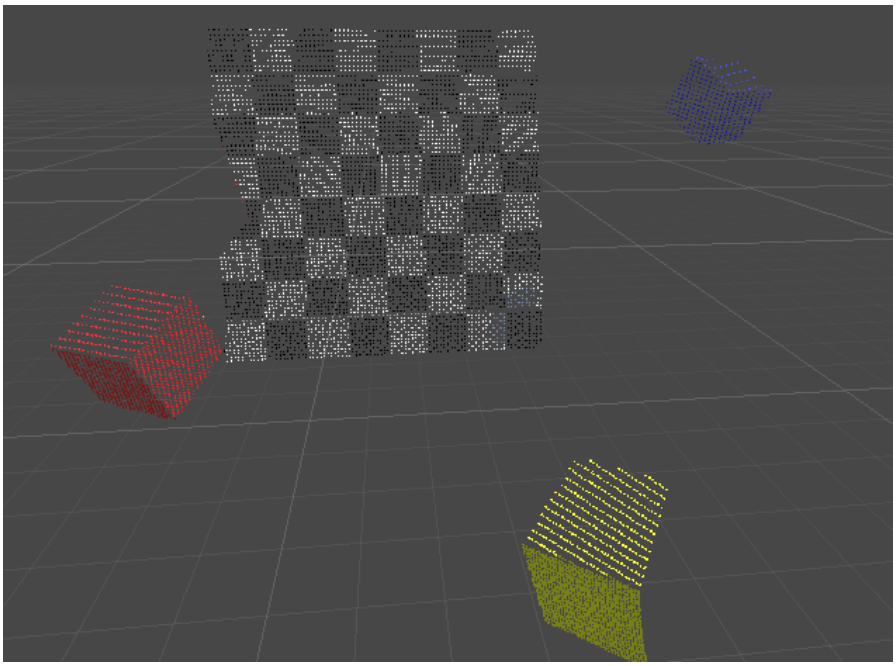

Abbildung 4.19: Ergebnis der Fusion vom Bild von Kamera 1 und dem Lidar

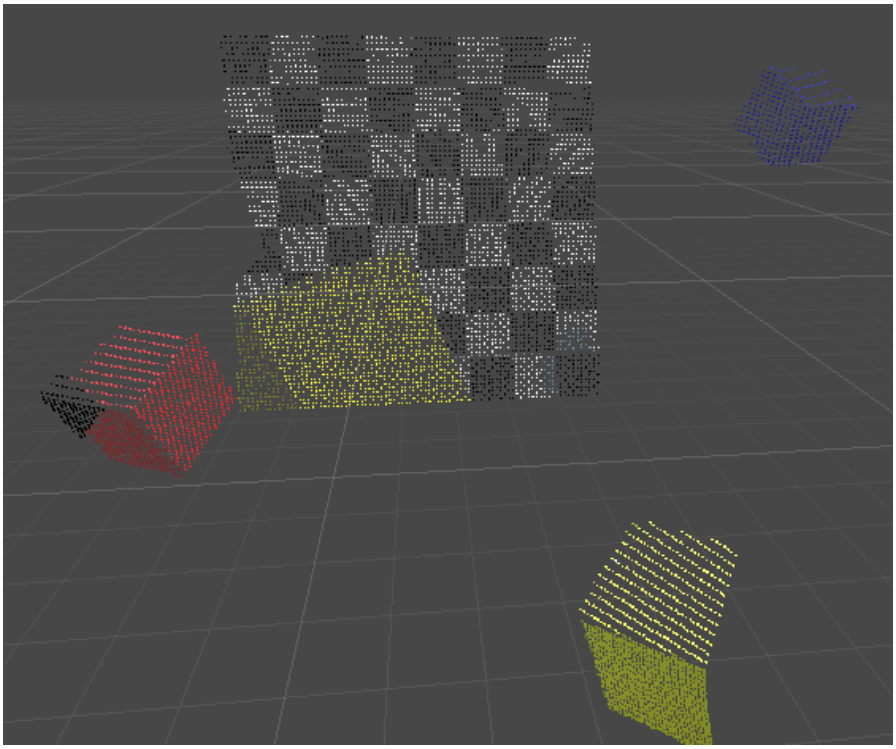

Abbildung 4.20: Ergebnis der Fusion vom Bild von Kamera 2 und dem Lidar

Da in diesem Versuchsaufbau alle Parameter so ideal wie möglich modelliert wurden, sieht man im Ergebnis der Fusion von Kamera 1 und dem Lidar in Abbildung 4.19, dass das Ergebnis praktisch perfekt ist. Nahezu alle Bildpixel wurden auf die korrekten Punkte der Punktwolke abgebildet.

Im Fusionsergebnis von Kamera 2 in Abbildung 4.20 sieht man jedoch ein Problem, welches auftritt, wenn ein Objekt, welches von dem Lidar nicht detektiert wird, ein anderes Objekt im Kamerabild verdeckt.

Die Fläche des gelben Würfels, die in Kamera 2 zu sehen ist wird hierbei vom Lidar nicht detektiert. Es werden hierbei die 3D-Punkte des Schachbretts in das Kamerabild abgebildet, aber da das System nicht wissen kan, dass diese Punkte vom gelben Würfel verdeckt werden, werden diese Werte einfach ausgelesen und diesen Punkten zugeordnet. Hierbei handelt es sich also um eine Fehlinterpretation.

Bei der Fusion mit den Echtweltgeräten ist dieses Problem nicht aufgetreten, da alle Geräte sehr nah beieinander waren und somit alle Geräte die Szene aus mehr oder weni-

ger derselben Perspektive wahrgenommen haben, wodurch Verdeckungen ausgeschlossen wurden.

Sind im Aufbau die Geräte jedoch weit auseinander muss also auf derartige Verdeckungen geachtet werden.

Ein weiterer Aspekt, der bei einem solchen Aufbau zu beachten ist, dass manche Punkte der Punktwolke gar nicht im Kamerabild zu sehen sind, da sie außerhalb des Blickwinkels liegen. In Abbildung 4.20 sieht man, dass die obere linke Ecke des roten Würfels schwarz markiert ist. Das liegt daran, dass erkannt wurde, dass diese Punkte nicht im Kamerabild von Kamera 2 zu sehen waren und somit als ”nicht zugeordnet” markiert wurden, was in der Visualisierung durch einen schwarzen Punkt signalisiert wird.

5 Bewertung und Ausblick

Der Kalibrierungsprozess enthielt in der Durchführung dieser Arbeit offensichtlich Fehler und Abweichungen, wodurch das Resultat der Fusion beeinflusst wurde, dennoch wurde das Kamerabild erfolgreich mit der 3D-Punktwolke fusioniert.

Da in dieser Arbeit insbensondere mit zwei realen Kameras gearbeitet wurde, die je verschiedene Spektren wahrnehmen, wurde gezeigt, dass sich diese Methode auch dazu eignet, multispektrale Eigenschaften einer Oberfläche zu extrahieren.

Auch die Simulation, die in dieser Arbeit umgesetzt wurde ermöglicht es, die realen Sensoren hinreichend zu simulieren, um ein ähnliches Fusionsergebnis zu erzielen.

5.1 Relevante Arbeiten und mögliche Erweiterungen

Der Prozess der Kalibrierung der Kameras und des Lidars sowie die anschließende Fusion wurde in (Ding u. a. (2020)) ähnlich durchgeführt, wie in dieser Arbeit. (Subedi u. a. (2020)) setzt den Prozess der Fusion ähnlich durch, jedoch wurde hier für die extrinsische Kalibrierung ein speziell vorgerfertigtes Objekt benutzt, welches aus einer Platte mit vier kreisförmigen Löchern besteht. Hierfür wurde ein spezieller Bilderkennungsalgorithmus und ein Algorithmus, der dieses Objekt in der 3D-Punktwolke detektieren kann, entwickelt. Dadurch ist es möglich, die 3D-2D-Korrespondenzen automatisch zu berechnen. Diese Art der extrinsischen Kalibrierung ist besser, als die Kalibrierung mit 3D-2D Korrespondenzen, wie sie in dieser Arbeit umgesetzt wurde, da zum einen viel mehr Korrespondenzen abgelesen werden können, was zu einem stabileren Ergebnis für die extrinsischen Parameter führt. Außerdem kann die Kalibrierung in Echtzeit durchgeführt werden. Damit so eine extrinsische Kalibrierung wie in (Subedi u. a. (2020)) durchgeführt werden kann, muss jedoch so ein spezielles Kalibrierungsobjekt vorhanden sein, mit einem Algorithmus, der speziell dieses Objekt erkennen kann, sowohl im Kamerabild als auch in der 3D-Punktwolke. Der Vorteil des Kalibrators in dieser Arbeit ist dabei, dass

Korrespondenzen manuell ausgewählt werden, wobei es genügt, wenn ein beliebiges Objekt in der Kamera und in der Punktwolke ablesbar ist. Da in dieser Arbeit aber sowieso die extrinsischen Parameter direkt abgemessen wurden hat dies keinen Einfluss auf die endgültige Fusion.

(Kim und Park (2019)) benutzt im Aufbau einen 360 Grad Lidar und Kameras, die konzentrisch aufgebaut sind und in einem Ring nach außen ausgerichtet sind. Die Kameras werden auch hier wie in dieser Arbeit basierend auf (Zhang (2000)) kalibriert. Die 3D-2D Korrespondenzen wurden basierend auf einem Schachbrett manuell ausgewählt, womit die Kameras mit dem Lidar kalibriert wurden.

(Munoz-Banon u. a. (2020)) benutzt einen alternativen Ansatz für die Kalibrierung der extrinsischen Parameter, der nicht auf einem spziellen Zielobjekt beruht. Stattdessen wird die Kalibrierung durchgeführt, indem das Lidar-Kamerapaar in der Szene bewegt wird, sodass Änderungen, die jeweils im Kamerabild als auch in der Punktwolke auftreten, detektiert werden, aus denen die 3D-2D-Korrespondenzen abgeleitet werden. Der Vorteil ist hier, dass diese Methode in Echtzeit und in einer beliebigen 3D-Szene angewendet werden kann. Außerdem benutzt (Shi u. a. (2019)) benutzt einen ähnlichen Ansatz um die Kalbirerung aufgrund der Odometrie der Szene zu berechnen.

Wie bereits erwähnt stellt die Fusion, die in dieser Arbeit umgesetzt wurde keinen vollständigen Sensordatenverarbeitungsprozess dar, wie er in (Winner u. a. (2015)) beschrieben wird (siehe Kapitel 1.2). Sie stellt jedoch eine Grundlage dar, auf der höhere Algorithmen Objekthypothesen, wie zum Beispiel 3D-Bounding Boxen für die Erkennung von Fahrzeugen oder ähnliches, machen können. (Balemans u. a. (2019)) setzt genau dies um und versucht aufgrund des Ergebnisses der Fusion Objekte wie Autos in der Szene sowohl in 2D als auch in 3D zu erkennen.

Auch eine Vorbearbeitung der Punktwolke bevor sie mit dem Kamerabild fusioniert wird ist möglich. In diesem Versuch wurden die ”rohen” Daten des Blickfeld Lidars für die Fusion benutzt, mit all ihren Unsicherheiten und Messfehlern. Ein Aufbau, der mehrere Lidars benutzt, könnte diese Unsicherheiten eliminieren, indem beispielsweise eine homogene Sensordatenfusion, wie sie in (Siciliano und Khatib (2016) Kapitel 35) beschrieben wird, umgesetzt wird. Das Ergebnis wäre nach wie vor eine Punktwolke in einem einzelnen zugehörigen Koordinatensystem. Eine solche Punktwolke könnte jedoch eine Szene besser und vollständiger wiederspiegeln, da die Messungen nicht nur aus einer Perspektive stammen. Bei einem solchen Konzept kann es jedoch dazu kommen, dass einzelne 3D-Punkte von Oberflächen stammen, die sich gegenseitig verdecken. Dass Verdeckung

ein Problem darstellen kann, wurde bereits in Kapitel 4.4 besprochen (siehe Abbildung 4.20), jedoch kann diese Art von Fehlinterpretation von Überdeckung vermieden werden. Ein möglicher Ansatz könnte hierfür sein, aus der rohen Punktwolke mögliche Oberflächen mit Normalenvektor abzuleiten und dann Punkte, die auf einer Oberfläche liegen, die von einer anderen Oberfläche im Bild verdeckt werden, aus der Menge der Punkte, die in das Kamerabild abgebildet werden sollen, zu entfernen. Ein geeigneter Algorithmus für eine solche Filterung könnte das Konzept des "Occlusion Cullings" sein, welches in (Akenine-Möller u. a. (2018) Kapitel 19.7) erläutert wird.

Dass der in dieser Arbeit umgesetzte Fusionsprozess auch für die Extraktion von Oberflächeneigenschaften in verschiedenen Spektralbereichen geeignet ist, wurde in dieser Arbeit gezeigt. Jedoch wurde dieses Konzept nicht in der Simulation umgesetzt, da nur eine RGB-Kamera simuliert wird. Die Simulation von einem Infrarotsensor und einer gewöhnlichen RGB Kamera wurde in (Carbone u. a. (2020)) behandelt. Hierbei lag der Fokus jedoch darauf, Unkraut in einem Beet durch ein neuronales Netz zu erkennen, um Schäden bei der Ernte im Agrarsektor zu minimieren. Dennoch wird hier gezeigt, dass es in Unity durchaus möglich ist, Infrarotsensoren neben RGB-Kameras auf photorealistischem Niveau zu simulieren, was auch für die Simulation einer Fusion einer Punktwolke mit einem Kamerabild benutzt werden kann.

6 Schlusswort

In dieser Arbeit wurde die Fusion einer 3D-Punktwolke mit mehreren Kamerabildern erfolgreich umgesetzt. Außerdem wurde gezeigt, dass es möglich ist, durch den Einsatz von Kameras mit verschiedenen Spektralbereichen Oberflächeneigenschaften aus verschiedenen Spektren zu extrahieren. Zwar war das Ergebnis durch Kalibrierungsfehler der intrinsischen und extrinsischen Parameter leicht fehlerhaft, dennoch bildet das in dieser Arbeit umgesetzte Konzept eine stabile Grundlage für eine mögliche weiterführende Datenverarbeitung, wie beispielsweise die Erkennung von 3D-Objekten oder ähnliches.

Literaturverzeichnis

[OpenCVDocs] OpenCV (Veranst.): *OpenCV Online Documentation: Camera Calibration and 3D Reconstruction*. https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html. – Letzter Zugriff: 15-11-2022

[Akenine-Möller u. a. 2018] Akenine-Möller, Tomas ; Haines, Eric ; Hoffman, Naty ; Pesce, Angelo ; Iwanicki, Michał ; Hillaire, Sébastien: *Real-Time Rendering 4th Edition*. Boca Raton, FL, USA : A K Peters/CRC Press, 2018. – 1200 S. – ISBN 978-1-13862-700-0

[Balemans u. a. 2019] Balemans, Dieter ; Vanneste, Simon ; Hoog, Jens de ; Mercelis, Siegfried ; Hellinckx, Peter: LiDAR and Camera Sensor Fusion for 2D and 3D Object Detection. In: *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*. Springer International Publishing, oct 2019, S. 798–807. – URL https://doi.org/10.1007%2F978-3-030-33509-0_75

[Bouain u. a. 2020] Bouain, Mokhtar ; Berdjag, Denis ; Fakhfakh, Nizar ; Ben Atitallah, Rabie: A Comprehensive Approach for Camera/LIDAR Frame Alignment. In: Gusikhin, Oleg (Hrsg.) ; Madani, Kurosh (Hrsg.): *Informatics in Control, Automation and Robotics*. Cham : Springer International Publishing, 2020, S. 761–785. – ISBN 978-3-030-11292-9

[Carbone u. a. 2020] Carbone, Carlos ; Potena, Ciro ; Nardi, Daniele: Simulation of near Infrared Sensor in Unity for Plant-weed Segmentation Classification, 01 2020, S. 81–90

[Ding u. a. 2020] Ding, Yuqi ; Liu, Jiaming ; Ye, Jinwei ; Xiang, Weidong ; Wu, Hsiao-Chun ; Busch, Costas: 3D LiDAR and Color Camera Data Fusion. In: *2020 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, 2020, S. 1–4

[Förstner und Wrobel 2016] Förstner, W. ; Wrobel, B.P.: *Photogrammetric Computer Vision: Geometry, Orientation and Reconstruction*. Springer International Publishing, 2016 (Geometry and Computing). – ISBN 9783319115511

[Kaehler und Bradski 2016] Kaehler, Adrian ; Bradski, Gary: *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. 1st. O'Reilly Media, Inc., 2016. – ISBN 1491937998

[Kim und Park 2019] Kim, Eung-Su ; Park, Soon-Yong: Extrinsic calibration of a camera-LIDAR multi sensor system using a planar chessboard. In: *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*, 2019, S. 89–91

[Levenberg 1944] Levenberg, Kenneth: A method for the solution of certain non-linear problems in least squares. In: *Quarterly of Applied Mathematics* (1944), Nr. 2, S. 164–168. – ISSN 0033569X, 15524485

[Munoz-Banon u. a. 2020] Munoz-Banon, Miguel A. ; Candelas, Francisco A. ; Torres, Fernando: Targetless Camera-LiDAR Calibration in Unstructured Environments. In: *IEEE Access* 8 (2020), S. 143692–143705

[OpenCV] OpenCV: *Open Source Computer Vision Library*. – Siehe https://opencv.org/. Letzter Zugriff: 15-11-2022

[OpenGL] OpenGL: *Open Graphics Library*. – Siehe https://www.opengl.org/. Letzter Zugriff: 15-11-2022

[ROS] ROS: *Robot Operating System (ROS)*. – Siehe https://www.ros.org/. Letzter Zugriff: 15-11-2022

[Shi u. a. 2019] Shi, Chenghao ; Huang, Kaihong ; Yu, Qinghua ; Xiao, Junhao ; Lu, Huimin ; Xie, Chenggang: Extrinsic Calibration and Odometry for Camera-LiDAR Systems. In: *IEEE Access* 7 (2019), S. 120106–120116

[Siciliano und Khatib 2016] Siciliano, Bruno (Hrsg.) ; Khatib, Oussama (Hrsg.): *Springer Handbook of Robotics*. Springer International Publishing, 2016. – URL https://doi.org/10.1007%2F978-3-319-32552-1

[Subedi u. a. 2020] Subedi, Dipendra ; Jha, Ajit ; Tyapin, Ilya ; Hovland, Geir: Camera-LiDAR Data Fusion for Autonomous Mooring Operation. In: *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2020, S. 1176–1181

[Unity] UNITY: *Unity 3D Engine*. – Siehe https://unity.com/. Letzter Zugriff: 15-11-2022

[Winner u. a. 2015] WINNER, Hermann (Hrsg.) ; HAKULI, Stephan (Hrsg.) ; LOTZ, Felix (Hrsg.) ; SINGER, Christina (Hrsg.): *Handbuch Fahrerassistenzsysteme*. Springer Fachmedien Wiesbaden, 2015. – URL https://doi.org/10.1007%2F978-3-658-05734-3

[Zhang 2000] ZHANG, Z.: A flexible new technique for camera calibration. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (2000), Nr. 11, S. 1330–1334

Erklärung zur selbstständigen Bearbeitung

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

Ort	Datum	Unterschrift im Original