

Bachelorarbeit

*Entwicklung eines
Lasertriangulationssensor zur
Oberflächen-Rekonstruktion mit ROS2*

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

vorgelegt dem

Fachbereich Mathematik, Naturwissenschaften und Informatik
der Technischen Hochschule Mittelhessen.

Tristan Elias Wolfram

xx.xx.202x, Gießen

Referentin: Prof. Dr.-Ing. Seyed Eghbal Ghobadi
Korreferent: Moritz Schauer, M.Sc.

Kurzfassung

Titel

Titel auf deutsch

Zusammenfassung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Abstract

Titel

Title in english

Summary

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Inhaltsverzeichnis

1 Einleitung	1
1.1 Aufgabe / Motivation	1
1.2 Stand der Technik	2
1.2.1 Methodik	2
1.2.2 Geräte	3
1.3 Kriterien	3
1.4 Lösungsansatz	4
2 Grundlagen	6
2.1 Bibliotheken	6
2.2 Der Grundlegende Aufbau	6
2.3 OpenCv Pinhole Camera Model	7
2.4 Mathematische Grundlage	9
2.4.1 Homogene Koordinaten	9
2.4.2 Koordinaten-Transformationen	11
2.5 Bildverarbeitung	13
2.5.1 Das Erkennen der Laserlinie	13
2.5.2 Umwandlung zu einem Grauwert-Bild	15
2.5.3 Gauß-Filter	16
2.5.4 Das Erstellen von Subpixeln	17
3 Hauptteil - Projektbeschreibung und Projektergebnisse	20
3.1 Kalibrierung	20
3.1.1 Intrinsische Kalibrierung	20
3.1.2 Extrinsische Kalibrierung	24
3.2 Das Erzeugen einer Punktewolke	30
3.2.1 Die 3D-Koordinaten	30
3.2.2 Das Finden der Farbinformationen	31
3.3 Aufbau / Hardware	33
3.3.1 Hardware-Aufbau des Lasertriangulationssensors	33
3.3.2 Der Versuchsaufbau	34
3.4 Aufbau / Software	36
3.4.1 Python (Bibliothek)	36
3.4.2 ROS2	39
3.5 Qualitative Ergebnisse	43
3.6 Evaluation	43

3.6.1	Testen von Genauigkeit	43
3.6.2	Fehler	43
3.6.3	Probleme und Schwierigkeiten	43
4	Fazit	44
5	Ausblick	46
5.1	Erweiterung	46
5.2	Anpassungen	48
A	Anhang	49
A.1	Anhang A	49
A.2	Anhang B	50
A.3	Anhang C	51
A.4	Anhang D	52
Abbildungsverzeichnis		53
Tabellenverzeichnis		54
Quellcodeverzeichnis		55
Abkürzungsverzeichnis		56

Danksagung

An dieser Stelle kann man Danke sagen. :-)

DANKE

1. Einleitung

Die Möglichkeit eine Oberfläche in ihrer Beschaffenheit 3D zu erfassen, birgt viele Anwendungsfälle. Sei es, um ein 3D-Abbild von einem Raum digital zu erhalten oder in der Industrie Bauteile zu vermessen und auf Fehler zu prüfen. Dabei sind viele verschiedene Verfahren über die Zeit entwickelt und optimiert wurden. 3D Kameras oder auch RGB-D Kameras finden heute vielfältig ihren Einsatz. Diese Arbeit beschäftigt sich mit der Dokumentation und Evaluation eines im Rahmen meiner Praktikumsphase an der Technischen Hochschule Mittelhessen entwickelten Lasertriangulationssensor. Ziel dieser Arbeit ist es dabei, die Funktionalität und Architektur dieses Sensors zu erklären. Zusätzlich soll der Sensor evaluiert und auf bestimmte Aspekte mit dem aktuellen Stand der Technik im Bereich der 3D-Kameras bzw. RGB-D Kameras verglichen werden. Sinn und Zweck eines Lasertriangulationssensor ist es eine Oberfläche zu rekonstruieren und dabei nicht nur die Tiefen-Informationen, sondern auch die Farbinformationen gemeinsam aufzunehmen. Als Verfahren wird, wie im Namen genannt, die Lasertriangulation verwendet.

1.1. Aufgabe / Motivation

Die grundsätzliche Aufgabe und Motivation entstand durch ein Projekt in Zusammenarbeit mit der Firma RINNTECH – „Technik zur Prüfung von Bäumen“. Die Firma bezeichnet sich selbst wie folgt: „Als Anwender und Entwickler verfügen wir über jahrelange Erfahrung und zahlreiche Patente auf dem Gebiet der Baum- und Holzanalyse. Für diese Anforderungen können wir Ihnen daher ausgereifte Technik und umfassenden Service anbieten“ [Vgl.: RIN]. Beispielanwendungen, für die Geräte und Software bereitgestellt werden sind zum Beispiel Bäume kontrollieren, Jahrringe analysieren, Holzkonstruktionen kontrollieren und Holzqualität und Zuwachs im Wald kontrollieren [Vgl.: RIN]. Das zugrundeliegende Projekt wurde als Forschungs- und Entwicklungsprojekt zusammen mit dem Institut für Technik und Informatik (ITI) an der THM gestartet. Es trägt den Titel

„Entwicklung einer Messmethodik zur Ermöglichung einer schnellen Bestimmung von Holzart und -herkunft anhand von Jahrring- und Farbanalyse. Entwicklung der Messwerterfassung und -auswertung der neuen Messmethodik“

und beschäftigt sich konkret mit der Jahrringanalyse. Durch diese soll Holzart und Herkunft bestimmt werden. RINNTECH verkauft das Produkt „LINTAP“ für diesen Zweck [Vgl.: LINTAP]. Das schon existierende Produkt soll in dem Projekt erweitert, optimiert und automatisiert werden. Die Grundidee besteht darin, dass ein Roboter-Arm mit einer hochauflösenden Kamera über das Objekt fährt und entsprechende Bilder aufnimmt, die für die Jahrringanalyse erforderlich sind. Hier wird eine Kamera eingesetzt, die von

dem Roboterarm nah an das Holz herangeführt werden muss. Die Pfade, die der Roboter dementsprechend abfahren muss, müssen also mit hinreichender Genauigkeit errechnet werden. Dazu ist eine digitale Abbildung des Holzes unverzichtbar. Anhand einer rekonstruierten Oberfläche können die entsprechenden Pfade für den Roboter ohne Probleme errechnet werden. Es muss also initial eine Oberflächen-Rekonstruktion des Objektes stattfinden. An diesem Punkt setzt meine Arbeit an. Es wurde eine Software entwickelt, die mithilfe einer Kamera und einen Linienlaser einen 3D-Scan durchführt. Dabei werden die Tiefeninformationen und auch Farbinformationen aufgenommen und verarbeitet. Man erhält eine Punktwolke der gescannten Oberfläche die entsprechend gefärbt ist. Ein typischer Output für eine 3D-Kamera in der Industrie. Die Aufgabenstellung wurde noch etwas konkretisiert. Als Methode soll eine Lasertriangulation verwendet werden, dazu wurde die Kamera und der Linienlaser bereitgestellt. Zusätzlich soll nur Open-Source-Software verwendet werden und die Anwendung soll über ROS2 (Robot Operating System) laufen.

1.2. Stand der Technik

1.2.1. Methodik

Lasertriangulation ist nicht die einzige Methode eine Oberflächen-Rekonstruktion durchzuführen. Die zentralen Technologien zu diesem Zweck sind Triangulation oder Time-of-Flight [Vgl.: SotA]. Bei der Triangulation gibt es zwei Ansätze. Einen direkten über Structured Light. Dabei wird ein bekanntes Muster mit einem Laser oder Ähnlichem auf die Oberfläche projiziert. Eine Kamera nimmt das Muster als Bild auf, welches sich durch die variierende Höhe und Form der Oberfläche verzerrt. Diese Verzerrung wird als Anhaltspunkt verwendet, um die Unterschiedlichkeiten in der Höhe zu ermitteln. Lasertriangulation und der hier verwendete Lösungsansatz gehören zu dieser Möglichkeit. Einen indirekten Ansatz der Triangulation bietet Stereo-Vision. Dabei werden zwei Kameras verwendet, die zwei aufgenommenen Bildern aus unterschiedlichen Positionen liefern. Ebenfalls wird ein fester Punkt benötigt, der auch mit einem Laser oder Ähnlichem im Bild projiziert werden kann. Über die zwei unterschiedlichen Bilder kann die Position ermittelt werden.

Die Time-of-Flight-Technologie benutzt eine andere Lösungsmöglichkeit. Es wird Licht auf einen Punkt auf der Oberfläche gestrahlt. Dort wird es zurück reflektiert und von einem Sensor registriert. Dieser misst die verstrichene Zeit. Durch die bekannte Geschwindigkeit von Licht, kann über die gebrauchte Zeit der zurückgelegte Weg errechnet werden. Dieser entspricht der Höhe.

1.2.2. Geräte

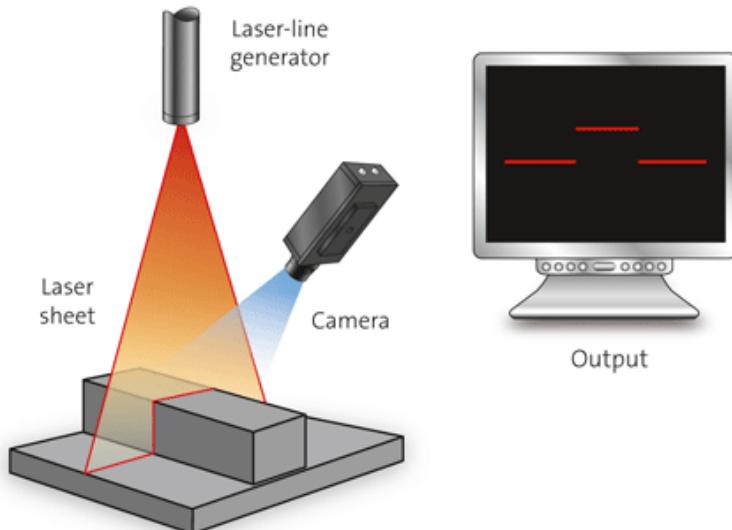
Diese Methoden finden ihre Anwendung auch in der Industrie. Diverse Geräte und Anwendungen zur Oberflächen-Rekonstruktion sind bereits auf dem Markt. Die Rede ist von sogenannten 3D-Kameras bzw. RGB-D Kameras. Dabei steht RGB (Red, Green, Blue) für die Farbinformationen und das D (Depth) für die Tiefeninformation. Angefangen mit der von Microsoft entwickelten „Kinect“ über die „Intel RealSense“ zur „Google Tango“ folgen viele weitere Geräte. Diese sind nicht nur meist für einen geringen Preis verfügbar, sie können auch die entsprechenden Pixelfarben in einer guten Auflösung aufnehmen. Zusätzlich geschieht die Aufnahme in Echtzeit, was bedeutet, dass sich die herausgegebene Punktwolke ändert, sobald sich die aufgenommene Oberfläche ändert bzw. die Kamera bewegt wird [SotA].

1.3. Kriterien

Für das Projekt wurde zu Beginn eine „Intel RealSense d415“ verwendet. Die Industrie-3D-Kamera erfüllt die Anforderungen. Es kann eine 3D-Rekonstruktion des Objektes mit Farbinformationen durchgeführt werden. Die RealSense verwendet Stereo-Vision-Technologie. Sie verfügt über zwei Kameras und einen Projektor für ein vom Menschen nicht sichtbares Infrarot-Muster. Die Kamera berechnet so für jeden Pixel seine 3D-Position in Echtzeit. In dem Forschungsprojekt wurde jedoch auch überlegt, ob man die 3D-Rekonstruktion jenseits einer Industrie-Lösung erhalten kann. Grundsätzlich ist dafür eine einfache Webcam mit einem Linienlaser ausreichend. So kam die Idee einen Open-Source Lasertriangulationssensor zu entwickeln. Dieser soll mit der Intel RealSense und dem generellen Industriestandard von RGB-D Kameras verglichen werden. Dabei ist die Genauigkeit (der Tiefeninformationen), Schnelligkeit des Scans und die Auflösung der Farbinformationen ausschlaggebend. Sowohl die Intel RealSense als auch die Open-Source-Variante sollen am Ende eine Punktwolke liefern. Diese kann direkt verglichen werden. Genauso können die Position der Punkte auf eine Genauigkeit untersucht werden.

1.4. Lösungsansatz

Um den genaueren Erklärungen im Hauptteil folgen zu können, soll einmal oberflächlich der Lösungsansatz der entwickelten Anwendung erläutert werden. Die Grundlegende Idee ist die Lasertriangulation. Dafür ist eine Kamera und ein Projektor für eine Laserlinie notwendig.



© STEMMER IMAGING

Abbildung 1: Lasertriangulation

Die resultierende Punktewolke der 3D-Rekonstruktion muss im Bezug zu einem Koordinatensystem stehen. Als Bezugspunkt wird die Kamera gewählt. Das bedeutet, dass sich die Koordinaten auf die Kamera beziehen und sich der Ursprung des Koordinatensystems an den optischen Sensor der Kamera befindet. Für die Berechnung von einem Pixel zu einem 3D-Punkt wird die Ebene bestimmt, die der Laser projiziert. Ausgehend von dem Startpunkt des Projektors und der projizierten Linie kann das Laserlicht als Ebene begriffen werden. In Abbildung 1 ist diese rot ausgefüllt. Diese Ebene kann aus Kamerasisicht als Ebenengleichung dargestellt und errechnet werden. In der Theorie wird dann, sobald die Ebenengleichung bekannt ist, eine Laserlinie auf die zu scannende Oberfläche projiziert. Die Kamera nimmt ein Bild von dieser Oberfläche auf. Über diverse Bildverarbeitungs-Operationen wird aus dem aufgenommenen Bild die rote Laserlinie herausgearbeitet. In Abbildung 1 wird unter *Output* beispielhaft das Ergebnis dieses Prozesses gezeigt. Die Pixel der Laserlinie werden im nächsten Schritt auf die Ebene gespannt, indem sie in die Ebenengleichung eingesetzt werden. Dadurch wird eine dritte Dimension für die Pixel errechnet. Die 3D-Punkte sind dann, wie gefordert, aus Sicht der Kamera. Diese Methodik

liefert die 3D-Präsentation der Laserlinie, jedoch nicht von dem kompletten Objekt. Der Sensor (bestehend aus der Kamera, den Laser und der Software zusammen) muss zusätzlich über das Objekt bewegt werden. Die Aufgenommenen Linien werden dann zu einer gesamten Punktwolke zusammengefügt. Dabei muss bekannt sein, welche Strecke an Bewegung von dem Sensor zurückgelegt wurde, um die neue Linie in der richtigen Position einzufügen.

2. Grundlagen

Um die Vorgänge und die Funktionsweise des Lasertriangulationssensors genau zu verstehen, sollen zuvor einige Grundlagen für den Lösungsansatz erläutert werden. Angefangen wird mit der mathematischen Grundlage, um die internen Berechnungen nachzuvollziehen. Danach werden die verwendeten selbst erstellten Algorithmen besprochen. Da dabei gewisse Bibliotheken mit Python zum Einsatz gekommen sind, sollen diese im ersten Schritt kurz erwähnt werden.

2.1. Bibliotheken

Der Lasertriangulationssensor wurde vor Allem mit OpenCv und ROS2 entwickelt. OpenCv bietet die perfekte Unterstützung für die benötigte Bildverarbeitung. Diverse Funktionen, um Bilder zu bearbeiten, zur Kamerakalibrierung und zum Errechnen der Ebenengleichung sind in OpenCv implementiert. ROS2 kümmert sich um die Automatisierung und den generellen Ablauf eines Scavorgangs. In der Forschung und Entwicklungs-Projekt mit RINNTECH wird zusätzlich auch ROS2 als übergeordnetes System genutzt. Deshalb war ROS2 auch eine Anforderung an das Projekt. Die vorher benutzte RGB-D-Kamera Intel RealSense ist ebenfalls in der Lage über ROS2 angesprochen zu werden. Da der OpenSource-Lasertriangulationssensor diese ersetzen soll, ist die Verwendung von ROS2 ein logischer Schritt. Erwähnenswert ist ebenfalls die junge Bibliothek Open3D. Sie wird benötigt mit Punktewolken zu arbeiten.

2.2. Der Grundlegende Aufbau

Der grundlegende Aufbau orientiert sich an den Lösungsansatz. Notwendig sind dafür nur ein Linienlaser und eine Kamera. Zuerst wurde für einen Prototyp zum Testen eine Webcam verwenden, später eine Industriekamera. Ausschlaggebend zum Funktionieren des theoretischen Lösungsansatz ist, dass die Kamera einen Linienversatz aufnehmen kann. Um das zu erreichen werden Kamera und Laser in einem gewissen Winkel zueinander gesetzt. Durch die Perspektive der Kamera entsteht der Linienversatz. Dabei ist egal, ob die Kamera von oben auf das Objekt zeigt und der Laser schräg sitzt oder andersrum. Die Lasertriangulationssensoren aus der Industrie weisen zumeist den Aufbau aus Abbildung 1 auf.

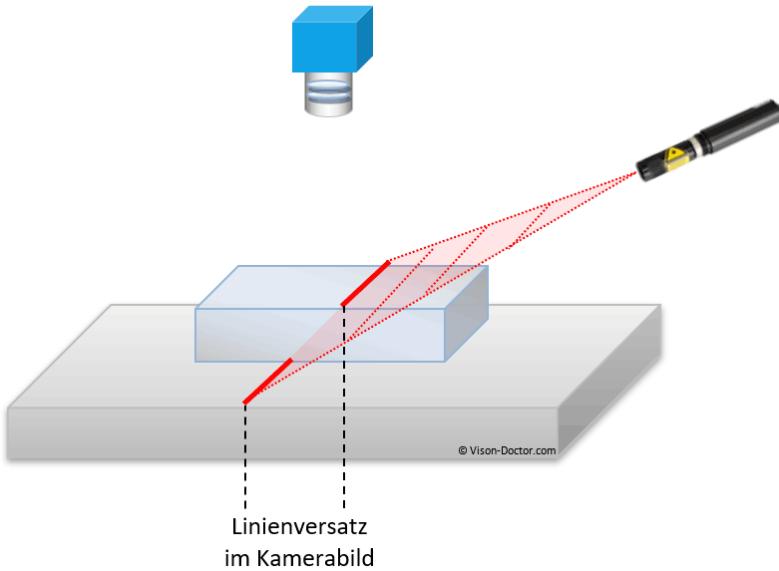
Prinzip der Laser-Triangulation

Abbildung 2: Positionen bei der Lasertriangulation

Der erste Schritt in der Erarbeitung des Sensors war es, die Möglichkeit zu entwickeln, eine aufgenommene Laserlinie in eine korrekte Punktewolke umzusetzen. Danach muss es ermöglicht werden den Sensor über bzw. das Objekt unter dem Sensor bewegen zu können. Für die folgenden Erklärungen zur mathematischen Grundlage, Bildverarbeitung und Kalibrierung ist nur wichtig zu wissen, dass die Kamera und Laser in einem Winkel zueinander über dem Objekt angebracht sind (Abbildung 2). Ebenfalls ist wichtig, dass Kamera und Laser fest angebracht sind und sich zueinander nicht bewegen. Nur der ganze Sensor ist bewegbar, dabei bleiben dann aber Kamera und Laser zueinander in der gleichen Position.

2.3. OpenCv Pinhole Camera Model

Der Anfang aller Berechnungen des Lasertriangulationssensors ist immer ein Bild. Die Kamera als optischer Sensor ist die einzige Informations-Quelle. Ein Bild besteht aus Pixeln. Diese sind 2-Dimensional. Ziel ist es die dritte Dimension zu finden. Für die gesuchten 3D-Punkte des Objektes ist dafür eine Ebenen-Gleichung für die Laser-Ebene notwendig. Diese ist nicht von Anfang an bekannt und es gilt diese herauszufinden. Es wird trotzdem nach einem Verfahren gesucht, welches unabhängig von einem Laser oder einer Ebene, sondern nur mithilfe der Kamera, 3D-Informationen liefern kann. Genau diese Informationen sind zugänglich mit den Pinhole Camera Model von OpenCv. In OpenCv wird eine Kamera genauso, wie eine Lochkamera begriffen.

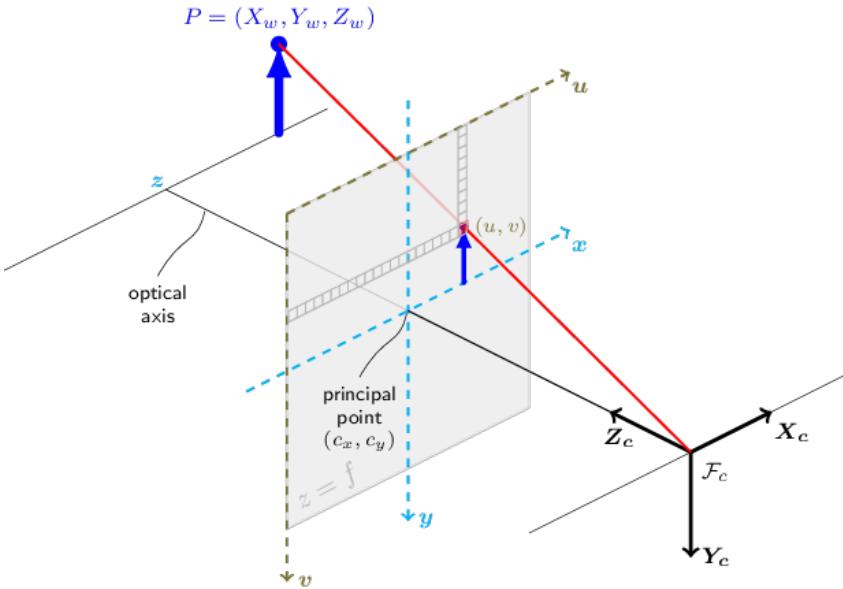


Abbildung 3: Pinhole Camera Model
(tesaugdsakif sd)

In Abbildung 3 ist dieses Modell dargestellt. Dabei gilt als optisches Zentrum, welches bei einer Lochkamera das Lochblende ist. Daran orientiert sich das Kamera-Koordinatensystem. Bei einer Lochkamera ist es üblich, dass eine Szene aus der Echten Welt aufgenommen wird. Diese wird über die Öffnung auf dem Kopf gespiegelt auf einem Schirm abgebildet.

Zum Vergleich ist in Abb. 3 das Modell einer Lochkamera zu sehen. Wenn man die beiden Darstellungen vergleicht, fällt auf, dass der „Schirm“, auf dem das Bild als Reflektion dargestellt wird bei der Lochkamera hinter der Lochblende ist und bei dem Modell von OpenCv davor. Die physikalisch korrekte Darstellung ist die der Lochkamera. Jedoch ist OpenCv nicht auf die richtige physikalische Darstellung angewiesen. Mathematisch macht es keinen Unterschied, ob die Bild-Ebene vor oder hinter dem optischen Zentrum ist. So kann ein Punkt in der Szene als Vektor begriffen werden, der durch die Bild-Ebene einen Pixel definiert und zum optischen Zentrum führt. Zu sehen in Abb.2 als die rote Linie. In unserem Fall kennen wir das Bild und die genaue Position eines Pixels mit u und v . Das Ziel ist es 3D-Koordinate zu erhalten.

2.4. Mathematische Grundlage

Grundlegend für die Umrechnung des Pixels in der Bild-Ebene ist die folgende Formel:

$$s \ p_{pix} = A [R|t] p_w \quad (1)$$

Sie beschreibt die Projektion eines 3D-Punktes in eine Szene zu einem Punkt in der Bild-Ebene. Hierbei ist p_{pix} der Pixel im Bild. p_w ist die Welt-Koordinate, welche gesucht wird. A ist die Kamera-Matrix. $[R|t]$ ist eine Rotation und Translation und beschreibt eine Transformation vom Kamerakoordinatensystem zum Weltkoordinatensystem. Die genaue Entstehung der Formel beschreibt OpenCv in [OpenCv-CamCalib]. Kamera-Matrix, Rotation und Translation sind hierbei neu. Die genaue Bedeutung wird in dem Kapitel 3.1 Kalibrierung genannt. Zum Verstehen der Formel ist hier nur wichtig, dass diese durch eine Kamerakalibrierung herausgefunden werden können. Die Variablen sind also bekannt. Kamera-Matrix und Rotation sind beide jeweils 3x3 Matrizen. Die Translation wird durch einen Vektor (3x1) beschrieben.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (2)$$

In der Formel (2) wird dies noch einmal genauer gezeigt. Bekannt sind also p , A und $[R|t]$. Unbekannte Variablen sind s der Scale-Factor und p_w die Weltkoordinate.

2.4.1. Homogene Koordinaten

In der Formel (2) sind die Rotation (R), die Translation (t) als homogenen Matrix und der Punkt im Weltkoordinatensystem (p_w) als homogener Vektor dargestellt. Der Vorteil vom Verwenden einer homogenen Matrix ist, dass beliebig viele Transformationen im dreidimensionalen Raum in dieser zusammengefasst werden können. Die Matrix kann dann auf einen Punkt im dreidimensionalen Raum, dargestellt als homogener Vektor, angewandt werden. In diesem Fall sind die Transformationen eine Rotation und eine Translation von dem Punkt im Kamerakoordinatensystem zu dem Weltkoordinatensystem. Das Zusammenführen von Rotation und Translation ergibt Sinn. Die Transformation von dem einen Koordinatensystem in das andere ist nur abgeschlossen, wenn sowohl die Rotation als auch die Translation auf den Zielvektor angewandt wurden. Die Formel (1) soll aber im Folgendem nach dem Punkt im Weltkoordinatensystem umgestellt werden (siehe (5)).

Für diese Umstellung wird die homogene Matrix wieder aufgeteilt. Damit dieser Vorgang nachvollziehbar ist, wird kurz die Entstehung der homogenen Matrix für dieses Beispiel erläutert.

Die grundsätzliche Transformation (Rotation + Translation), die angewandt werden soll ist:

$$\begin{aligned} v' &= R v + t \\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} v + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \end{aligned} \quad (3)$$

Dabei ist v ein Beispielvektor.

Eine homogene Matrix ist eine 4x4-Matrix. Die unterste Zeile ist immer $(0, 0, 0, 1)$. Beide Transformationen können in ihr eingebunden werden. Die Matrix wird dann mit den homogenen Vektor verrechnet:

$$\begin{aligned} v'_{homogen} &= \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 1 \end{bmatrix} \\ &= [R|t] v_{homogen} \end{aligned} \quad (4)$$

2.4.2. Koordinaten-Transformationen

Um den 3D-Punkt im Weltkoordinatensystem anhand eines Pixels zu errechnen, wird die grundlegende Formel umgestellt. Die folgende Abbildung zeigt nochmal das Pinhole Camera Model und verdeutlicht dabei die angewandten Transformationen.

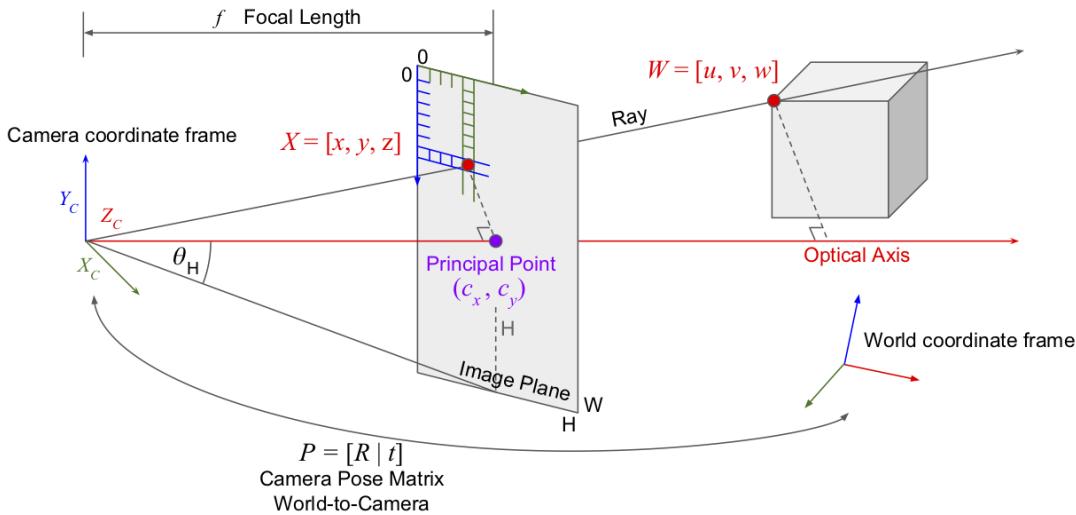


Abbildung 4: Transformationen im Pinhole Camera Model
<https://www.oreilly.com/library/view/mastering-opencv-4/9781789533576/848e4e77-32ec-499d-9945-cb0352e28236.xhtml>

Das Ziel ist es den 3D-Punkt (In Abbild 4 W) im Weltkoordinatensystem zu errechnen. Die Rotation (R) und Translation (t) werden für die Umrechnung in das Kamerakoordinatensystem benötigt. Das sind die sogenannten extrinsischen Parameter. In Abb.: (4) wird diese Transformation unter P der **Camera Pose Matrix** dargestellt.

Die Kamera-Matrix beschreibt die Transformation zur Bild-Ebene bzw. den Pixelkoordinatensystem. Die Matrix enthält die sogenannten intrinsischen Parameter. Das ist die in Abb.: (4) gezeigte **Focal Length** f , welche den Abstand vom Kamera-Koordinatensystem zur Bild-Ebene beschreibt. Hinzu kommt der **Principal Point**. Dieser befindet sich normalerweise in dem Zentrum der Bild-Ebene. Beides zusammen ergibt die gezeigte Matrix aus (2). Letztendlich wird damit die Umrechnung vom Kamerakoordinatensystem (in Abb.: (4) **Camera coordinate frame**) zur Bild-Ebene (in Abb.: (4) **Image Plane**) dargestellt [OpenCv-CamCalib].

Wir können einen Pixel im Bild auswählen und mithilfe dieser Parameter den entsprechenden Punkt im Weltkoordinatensystem errechnen. Dazu muss die Formel (1) nach dem Punkt im Weltkoordinatensystem umgestellt werden.

$$\begin{aligned}
 [A] [R|t] p_w &= s p_{pix} \\
 [R|t] p_w &= s [A]^{-1} p_{pix} \\
 [R] p_w &= s [A]^{-1} p_{pix} - t \\
 p_w &= s [R]^{-1} [A]^{-1} p_{pix} - [R]^{-1} t \\
 p_w &= s \vec{a} - \vec{b} \\
 \text{wobei : } \vec{a} &= [R]^{-1} [A]^{-1} p_{pix} \\
 \vec{b} &= [R]^{-1} t
 \end{aligned} \tag{5}$$

Diese Gleichung ist die Grundlage der Errechnung von 3D-Informationen. \vec{a} und \vec{b} dienen zur Vereinfachung. Wenn $[R]$, $[A]^{-1}$ und p_{pix} miteinander verrechnet werden entsteht ein Vektor (\vec{a}). Genauso entsteht aus $[R]^{-1}$ und t der Vektor (\vec{b}).

Der neue Ausgangspunkt ist die errechnete Weltkoordinate. Nach Aufgabenstellung sollen errechnete Koordinaten immer aus Sicht der Kamera dargestellt werden. Dazu wird die folgende Transformation benötigt.

Weltkoordinate zu Kamerakoordinate:

$$p_{cam} = [R] p_{welt} + t \tag{6}$$

Kamerakoordinate zur Weltkoordinate:

$$p_{welt} = [R]^{-1} (p_{cam} - t) \tag{7}$$

Anzumerken ist noch, das s der Scale-Factor immer noch eine Unbekannte ist. Es scheint also, als ob die Gleichung (5) noch nicht lösbar sei. Sobald konkrete Werte ausgerechnet werden sollen, muss s bekannt sein. Das passiert zum ersten Mal beim Erstellen der Ebenengleichung für die Laser-Ebene bei der extrinsischen Kalibrierung. Dabei wird auch darauf eingegangen, wie s errechnet werden kann.

2.5. Bildverarbeitung

Bekannt sind nun gewisse Grundlagen, wie wir mit einem ausgewählten Pixel im Bild umgehen können. Die Rechnungen und Transformationen sollen aber nicht auf zufällige oder sogar alle Pixel im Bild angewandt werden. Sie sollen auf ganz bestimmte ausgewählte Pixel erfolgen. Und zwar ganz genau diese, die zur abgebildeten Laserlinie gehören. Die Laserlinie ist immer unser Ausgangspunkt für die 3D-Informationen. Alle anderen Pixel interessieren uns im Grunde nicht.

Pixel können einfach und eindeutig als Tupel benannt werden. Sie können als Punkt in einem zweidimensionalen Koordinatensystem begriffen und dann mit einem horizontalen und vertikalen Wert genau gekennzeichnet werden. Benötigte wird eine Menge an diesen Tupeln, für die gilt, dass sie Teil der Laserlinie im Bild sind. Mit diversen Methoden der Bildverarbeitung können diese Pixel herausgefunden und abgespeichert werden, um mit ihnen weiterarbeiten zu können. Das folgende Kapitel bildet somit einen Algorithmus ab, dessen Ziel es ist ein Bild entgegen zu nehmen und die Pixel der Laserlinie zurückzugeben.

2.5.1. Das Erkennen der Laserlinie

Der erste Schritt des Algorithmus muss sein, die Laserlinie im Bild zu erkennen. Ein wichtiges Kriterium dabei war die Genauigkeit der ausgewählten Pixel. Jeder Pixel, der vom Algorithmus markiert wird, aber nicht zur eigentlichen Laserlinie gehört, wird in einem Fehler in der am Ende erstellten Punktwolke enden. Es wird also ein Punkt im Raum gezeigt, der nicht zur eingescannten Oberfläche passt. Dieser hängt dann beispielsweise in der Luft bzw. ist an einer Stelle im Koordinatensystem, wo sich eigentlich nichts befindet. Dieses sogenannte Rauschen soll möglichst gering sein.

Im Zuge einer umfangreichen Recherche zum Thema Lasertriangulation und OpenSource-Produzierten Laserlinienscannern wurde ein Paper von Bajpai und Perelman [baj-per], die auch einen Laserlinienscanner entwickelt haben, als Grundlage gewählt. Nicht nur bei dem Finden der Laserlinie, auch in diversen anderen Schritten der Errechnung von 3D-Punkten aus den Laserlinien-Pixeln ist dieses Paper eine Grundlage und Hilfestellung. Gemäß der dort verwendeten Methodik kam die Idee, zwei Bilder zu machen. In einem ist der Laser angeschaltet, in dem anderen nicht. Wenn diese Bilder voneinander abgezogen werden, kommen im Grunde genau die Veränderungen hervor. Da sich nichts anderes im Bild verändern sollte, außer das Erscheinen der Laserlinie, wird genau diese perfekt aufgezeigt. Voraussetzung dafür ist, dass die Umgebung der zu scannenden Fläche gleich bleibt. Einwirkungen wären zum Beispiel Änderungen bei den Lichtverhältnissen, bzw. bei der Beleuchtung oder auch eine Bewegung von Objekten zwischen der Aufnahme der zwei Bilder. Solche Einwirkungen würden in ungewolltem Rauschen enden oder sogar

die Laserlinie falsch positionieren. Weitere Informationen hierzu befinden sich ebenfalls im Kapitel 3.6.3 Probleme und Schwierigkeiten. Diese Anforderungen werden für den zu entwickelnden Laserlinien-Scanner akzeptiert. Ebenfalls sind es Anforderungen, die nicht direkt von der Software beeinflusst werden können. Sie sollten somit beim Aufbau des Scanners genauer beachtet werden und auf die Bildverarbeitung keinen Einfluss mehr haben dürfen. Das proben dieser Methode führte auch mit dem Testen über das Binärbild zu sehr guten Ergebnissen. Zusätzlich ist diese Methode sehr einfach über OpenCv umzusetzen.

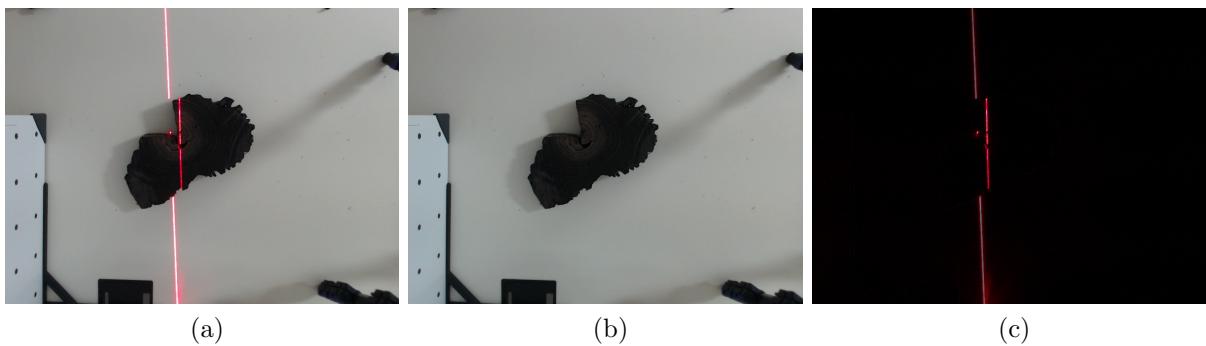


Abbildung 5: Subtraktion der Bilder

Festzuhalten ist damit, dass die Kamera zwei Bilder aufnehmen muss, eins Bild mit Laserlinie und ein Bild ohne. Das Bild ohne Laserlinie (5b) wird dann von dem mit Laserlinie (5a) abgezogen. Der Algorithmus zum Finden der Laserlinien-Pixel arbeitet dann mit der Differenz (5c).

2.5.2. Umwandlung zu einem Grauwert-Bild

Für einen Menschen ist die Laserlinie jetzt schon auf einem Blick gut erkennbar. Ein Algorithmus braucht allerdings spezifische Informationen, um die Pixel auszuwählen. Auch die Pixel, die nicht eindeutig zur Laserlinie gehören sind nicht komplett schwarz mit einem RGB-Wert von (R=0, G=0, B=0). Einen kleinen für den Menschen zumeist nicht sichtbaren Unterschied in zwei nacheinander aufgenommenen Bildern wird es immer geben.

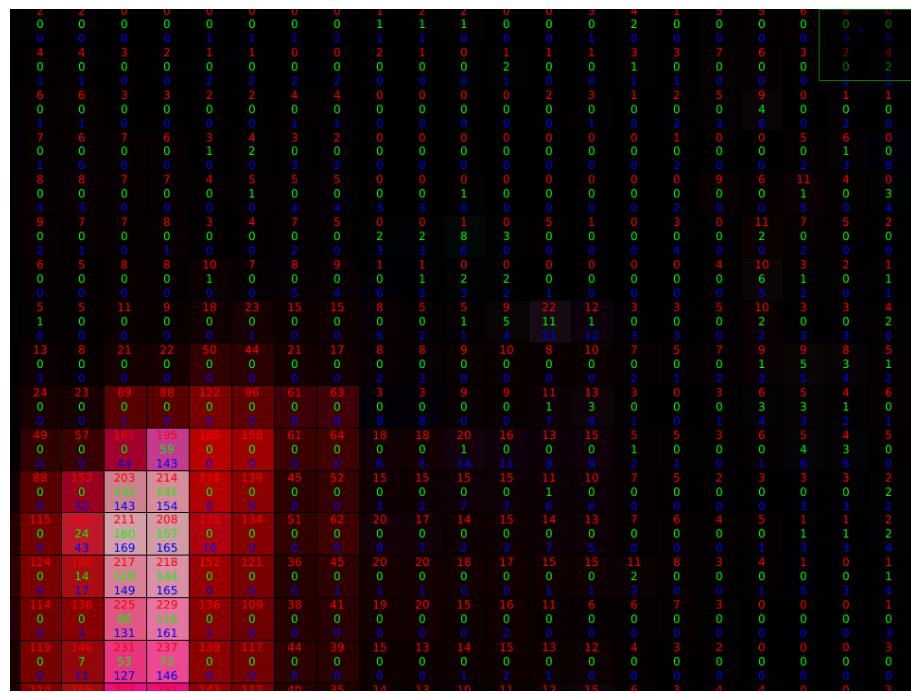


Abbildung 6: Pixel-Werte in einem Ausschnitt aus 5c am Rand der Laserlinie

Die Abbildung zeigt gut die Pixel der Laserlinie unten links. Die verschiedenen Farbbe-
reiche sind stark vertreten, vor allem der Rote. Die Pixel, die kein Teil der Laserlinie sind,
besitzen nur sehr schwache Werte in den drei Farbbereichen. Nach einer Regelung muss
hier immer noch genau festgelegt werden, welche Pixel für die Laserlinie ausgewählt wer-
den. Die Differenz von den zwei Bildern macht dies allerdings einfacher. Gleichbleibende
Stellen zwischen den beiden Bildern wie zum Beispiel sehr helle Stellen oder auch andere
rote Stellen sind in dem Differenz-Bild nicht mehr erkennbar. Ohne diesen Schritt wären
solche Stellen womöglich nicht genau von der Laserlinie zu unterscheiden.

Als zweiter Schritt wird das Bild zu einem Grauwert-Bild konvertiert. Vorteil davon ist, dass jetzt nicht mehr der RGB-Wert mit drei eigenen Werten ausschlaggebend ist, sondern nur noch die Intensität, bzw. der Grauwert eines Pixels. Die jeweilige Intensität für einen Pixel berechnet OpenCv nach der folgenden Formel [Vgl: OpencvDoc]:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \quad (8)$$

Damit hält OpenCv sich an die ITU-R BT 601 [vgl.: CCIR 601] und dem darin festgelegten Farbmodell YC_bC_r . Dabei handelt es sich um eine Konvention, wie digitale Video-Signale zu kodieren sind.

Die Formel (8) zeigt ebenfalls, dass aus den drei Farbwerten nun ein einzelner Grauwert errechnet wird. Allgemein kann dabei festhalten werden, dass je höher die einzelnen Farbinformationen waren, um so höher wird auch der Grauwert bzw. die Intensität sein. Das betrifft die Pixel der Laserlinie. Die anderen Pixel besitzen sehr geringe Farbwerte und werden deshalb auch eine geringe Intensität aufweisen.

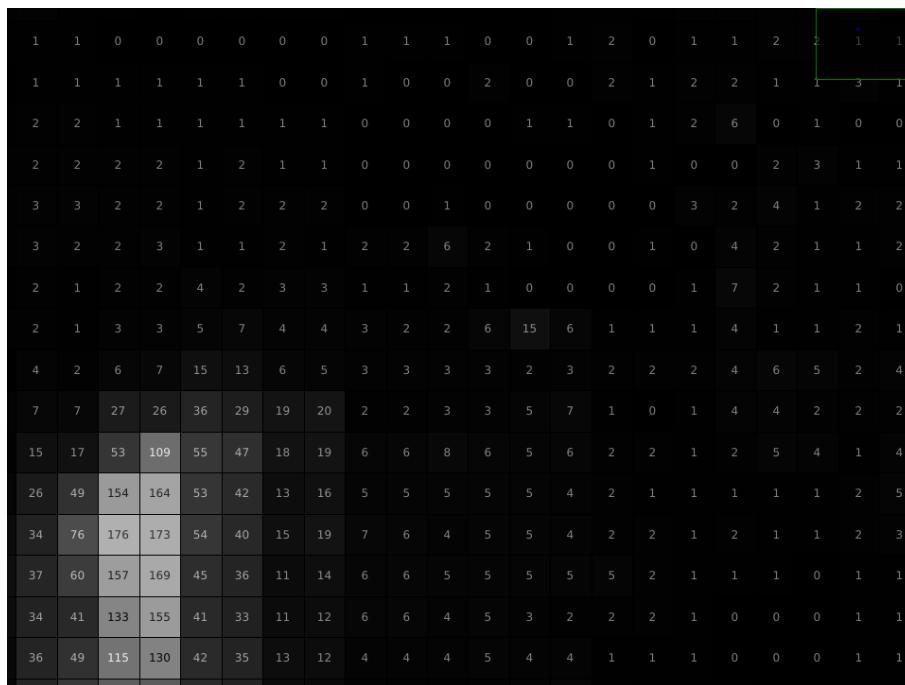


Abbildung 7: Der selbe Ausschnitt wie in Abbildung 6, nur als Grauwertbild konvertiert.

Die Abbildungen 6 und 7 zeigen auch, dass eine Laserlinie bis zu 4 oder 5 Pixel breit sein kann. Für die Weiterverarbeitung soll sie allerdings nur eine Breite von einem Pixel haben. Dazu wird in jeder Pixel-Zeile in einem Bild nach der höchsten Intensität gesucht. Das ist jedoch nur sinnvoll, wenn die Laserlinie im Bild von oben nach unten geht. Diese Anforderung kann schon im Aufbau des Sensors beachtet werden. Im Kapitel 2.2 wurde beschrieben, dass Kamera und Laser zueinander in der gleichen Position bleiben. Demnach kann der Laser entsprechend angebracht werden, dass er im Kamerabild entsprechend aufgenommen wird.

2.5.3. Gauß-Filter

Die einfachste Methode wäre also jede Zeile nach der höchsten Intensität zu suchen und den betreffenden Pixel auszuwählen. Dieser Pixel befindet sich je nach Aufnahme jedoch

nicht genau in der Mitte der Laserlinie. Die erarbeiteten Linien können so sehr „wellig“ werden, da Pixel untereinander zum teil stark in ihrer Position in der Zeile abweichen. Den Pixel mit der höchsten Intensität als Ausgangspunkt zu nehmen ist gut, jedoch sollen die umliegenden Pixel einen Einfluss machen können, um die Genauigkeit der Laserlinie zu erhöhen. Dazu wurden zwei Vorgänge entwickelt.

Die erste Methode ist es, einen Gauß-Filter über das Grauwert-Bild zu legen. Ein Gauß-Filter ist ein Weichzeichner, der umgangssprachlich ein Bild verwischt bzw. unscharf macht. Der Gauß-Filter wird auf jeden Pixel im Bild angewandt. Dabei ändert er den Wert des ausgewählten Pixels in Abhängigkeit der umliegenden Pixel. In diesem Projekt wird er dabei in einer Reichweite von einem 5x5-Feld definiert. Das bedeutet, das ausgehend von dem ausgewählten Pixel ein 5x5-Feld betrachtet wird. Der Gauß-Filter ist eine Standard-Bildverarbeitungsoperation, die vor allem mögliches Rauschen im Bild verringert. Wenn ein einzelner Pixel ein hohen Wert aufweist, jedoch alle anderen um diesen herum im Vergleich niedriger sind, wird der Pixel-Wert nach der Filter-Operation niedriger ausfallen. So werden demnach vereinzelte Pixel dunkler gemacht.

2.5.4. Das Erstellen von Subpixeln

Nachdem der Filter verwendet wurde, kommt die Operation, in der jede Zeile des Bildes nach dem Laserlinien-Pixel abgesucht wird. Hier wird zuerst der Pixel mit der höchsten Intensität gesucht. Dabei kann immer noch mehr Genauigkeit erzielt werden, wenn man die Pixel links und recht neben dem Ausgewählten mit beachtet. So fließt nicht nur die Intensität als Auswahlkriterium ein, sondern auch die benachbarten Pixel der Laserlinie in dieser Zeile. Da die erschlossenen Pixel in 3D-Koordinaten umgewandelt werden und somit vom Bild und der Pixel-Darstellung getrennt werden, ist es nicht mehr notwendig nur ganze Zahlen zur zu verwenden. Die Werte können also auch Nachkommastellen haben und sogenannte Subpixel ergeben. Für dieses Errechnen des passenden Subpixel wurde ein Algorithmus erstellt, der wie folgt funktioniert:

1. Threshold

Zuerst wird ein passender Threshold gesucht um eine Grenze festzulegen, ab welchen Wert überhaupt ein Pixel gefunden werden soll. Wenn die maximale Intensität in einer Zeile unter dem gewählten Threshold liegt, wird diese übersprungen. Damit wird auch ungewolltes Rauschen entfernt, da eine Intensität, die so niedrig ist, nicht zur Laserlinie gehören kann. Ein im Vorhinein festgelegter Threshold ist dabei nicht ausreichend, da dieser nicht an das aktuelle Bild angepasst wäre. Bilder können unterschiedlich ausfallen und nicht wie gewollt auf den Threshold reagieren. Der Threshold soll genau auf das

aktuelle Bild angepasst sein. Die Otsu-Methode ist ein Schwellenwertverfahren, welches genau den gewollten Threshold liefern kann [otsu]. Dabei wird von einem Grauwertbild ein Histogramm erstellt. Dieses wird analysiert und ein passender individueller Threshold ausgegeben. Die höchste Intensität der aktuellen Zeile wird auf diesen geprüft. Wenn sie zu niedrig ist, ist auch die ganze Zeile zu dunkel und wird nicht als Teil der Laserlinie angesehen.

2. Einbeziehen der benachbarten Pixel

Der Ausgangspunkt ist jetzt der Pixel mit der höchsten Intensität und dieser liegt über dem Threshold. Die benachbarten Pixel sollen jetzt in einen endgültigen Wert mit einbezogen werden. Um das zu erreichen wird eine Parabel genutzt. Es gilt eine quadratische Funktion in der Form $ax^2 + bx + c = I_x$, wobei I die Intensität und x die Position in der jeweiligen Zeile ist. Bei einem Bild, das beispielsweise 1920 Pixel breit ist, ist das ein Wert zwischen 0 und 1919. Die quadratische Funktion wird auf den ausgewählten Pixel und seine Nachbarn angewandt. Somit gilt:

$$\begin{aligned}
 ax^2 + bx + c &= I_x \\
 a(x+1)^2 + b(x+1) + c &= I_{x+1} \\
 a(x-1)^2 + b(x-1) + c &= I_{x-1} \\
 \text{daraus folgt:} \\
 \underbrace{\begin{pmatrix} x^2 & x & 1 \\ (x+1)^2 & (x+1) & 1 \\ (x-1)^2 & (x-1) & 1 \end{pmatrix}}_X \underbrace{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}_{\vec{abc}} &= \begin{pmatrix} I_x \\ I_{x+1} \\ I_{x-1} \end{pmatrix} \tag{9}
 \end{aligned}$$

Das Ziel ist es den Vektor \vec{abc} herauszufinden. Da die Intensitäten und die x-Werte bekannt sind, ist \vec{abc} die einzige Unbekannte und kann errechnet werden. Der nächste Schritt ist es die Nullstelle der Parabel herauszufinden. Sie befindet sich dort, wo die tatsächliche Intensität am höchsten ist. Dieser x-Wert befindet sich dann in den meisten Fällen zwischen zwei Pixeln. Dafür gilt:

$$\begin{aligned}
 2a \cdot x + b &= 0 \\
 x &= \frac{-b}{2a} \tag{10}
 \end{aligned}$$

b und a sind durch das Errechnen von \vec{abc} bekannt und ein Wert für x kann gefunden werden.

In der Rechnung fällt auf, dass x für jede Zeile individuell ausgerechnet wird. Dabei ändert sich in (9) die Werte für x und die Intensitäten. Hier kann man die Berechnung vereinfachen. wählt man für $x = 0$, entsteht die folgende Matrix:

$$X = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \end{pmatrix} \quad (11)$$

Wenn \vec{abc} mit dieser Matrix und den individuellen Intensitäten berechnet wird, errechnet man immer die Abweichung zu Mitte. Dabei ist die Mitte 0 und das entspricht den ausgewählten Pixel mit der höchsten Intensität. Man erhält eine Zahl zwischen -1 und 1. Diese wird mit der x-Position des Ausgangspixel in der Zeile verrechnet. Der Algorithmus wurde somit vereinfacht, da sich in jeder Rechnung nur der Intensitäten-Vektor ändert und immer die gleiche Matrix X gewählt werden kann.

Mit dem Finden des x-Wert ist die Bearbeitung einer Zeile fertig. Es gibt nun eine eindeutige 2D-Koordinate für einen Punkt der Laserlinie. Der Algorithmus geht über jede Zeile (y) und findet einen x-Wert. Dabei fügt er den gefundene Subpixel (x, y) in ein Array ein. Die Menge dieser Subpixel ist die Abbildung der Laserlinie in einen 2-dimensionalen Raum. Diese wurde als Output gefordert.

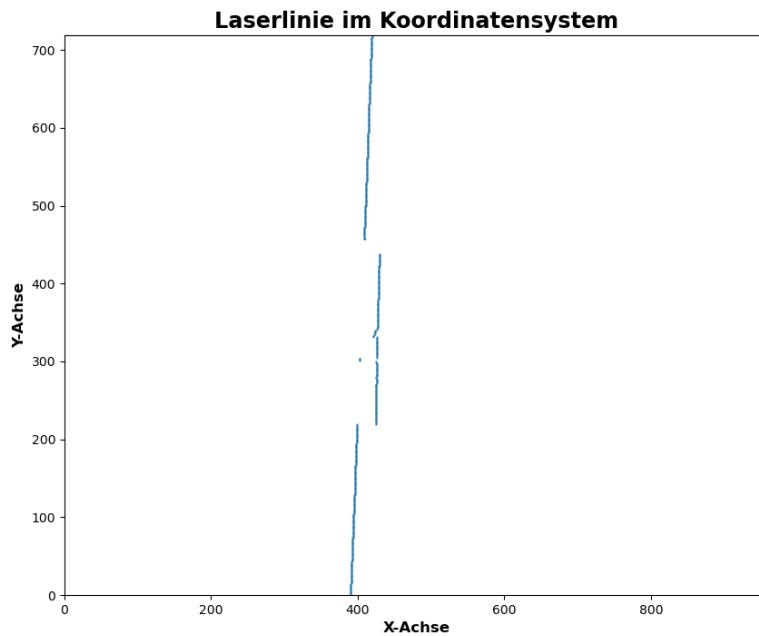


Abbildung 8: Die Subpixel können nun in einem Koordinatensystem dargestellt werden.

3. Hauptteil - Projektbeschreibung und Projektergebnisse

3.1. Kalibrierung

Die Kamerakalibrierung ist ausschlaggebend, um die Formel (2) benutzen zu können. Durch sie wird die Kamera-, Rotationsmatrix und der Translationsvektor gefunden. Dabei unterscheidet man zwischen intrinsische und extrinsischer Kalibrierung, bei denen auch ein unterschiedliches Kalibrierverfahren angewandt wird.

3.1.1. Intrinsische Kalibrierung

Die intrinsische Kalibrierung beschäftigt sich mit dem *Inneren* der Kamera. Ergebnis der Kalibrierung ist die Kameramatrix A . Die Kamerakalibrierung ist in OpenCv implementiert und wurde in diesem Projekt genutzt [OpenCv-CamCalib]. Die Implementierung richtet sich nach der Kamerakalibrierung nach Zhang [Zhang] und nach Bouguet [Bouguet].

Der grundlegende Vorgang ist es, ein bekanntes Muster in verschiedenen Positionen mit der Kamera aufzunehmen. So ein Muster ist beispielsweise ein Schachbrett, welches auch in dieser Arbeit für das Kalibrieren benutzt wurde. Die Maßen von den Kacheln sind dementsprechend dann bekannt und der Druck muss genau sein. Dabei muss es sich nicht um ein herkömmliches Schachbrett handeln. Die Größe und die Menge an Kachel kann zum kalibrieren angegeben werden. Man muss OpenCv also sagen, nach welchem Schachbrett der Algorithmus suchen muss.

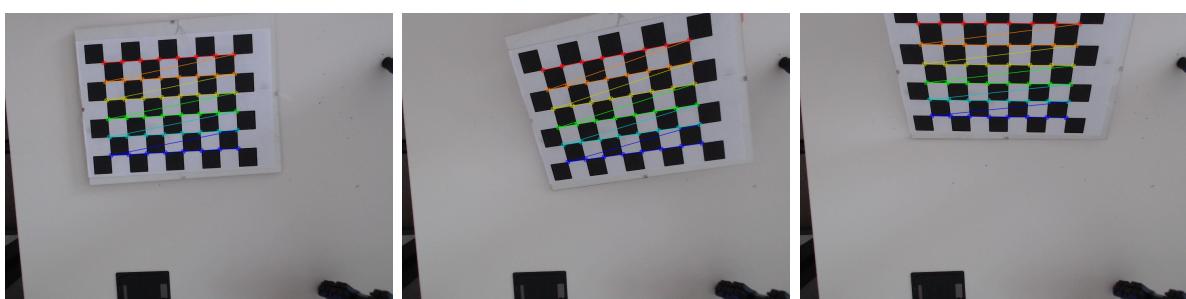


Abbildung 9: Schachbrett-Kalibrierung mit dem Schachbrett aus verschieden Positionen

Die benutzte Methode heißt **calibrateCamera()**. Sie nimmt beliebig viele Bilder entgegen. In der eigenen Implementierung wurde gemäß [OpenCv-Cam] vorgegangen. Dabei handelt es sich um die offizielle Einführung zur Kamerakalibrierung von OpenCv. In dem Beispiel in Abb.: (9) wurden der Einführung folgend auch die Ecken des definierten

Schachbrettes eingezeichnet. Damit kann man im Grunde sichbar machen, dass der Algorithmus das Schachbrett richtig erkannt hat. Gemäß [OpenCv-Cam] werden mindestens 10 Bilder benötigt, um gute Ergebnisse zu liefern. Daran wurde sich ebenfalls gehalten. Die Methode gibt die folgenden Werte zurück:

Kameramatrix

Die **Kameramatrix** wird errechnet und ausgegeben. Dabei handelt es sich um die vollwertige Matrix A aus den Formeln (1) und (2).

Distortion – Koeffizienten

Die Distortion-Koeffizienten werden gemäß [OpenCv-Cam] geliefert. Distortion bedeutet Verzerrung und bezieht sich auf das aufgenommene Bild. Dies Auswirkungen sind, dass aufgenommene gerade Linien in der Szene im Bild nicht mehr gerade dargestellt werden. Sie biegen sich bzw. sind verzerrt. Es gibt zwei verschiedene Verzerrungen, beschrieben in [Dist] und von OpenCv selbst in [OpenCv-CamCalib].

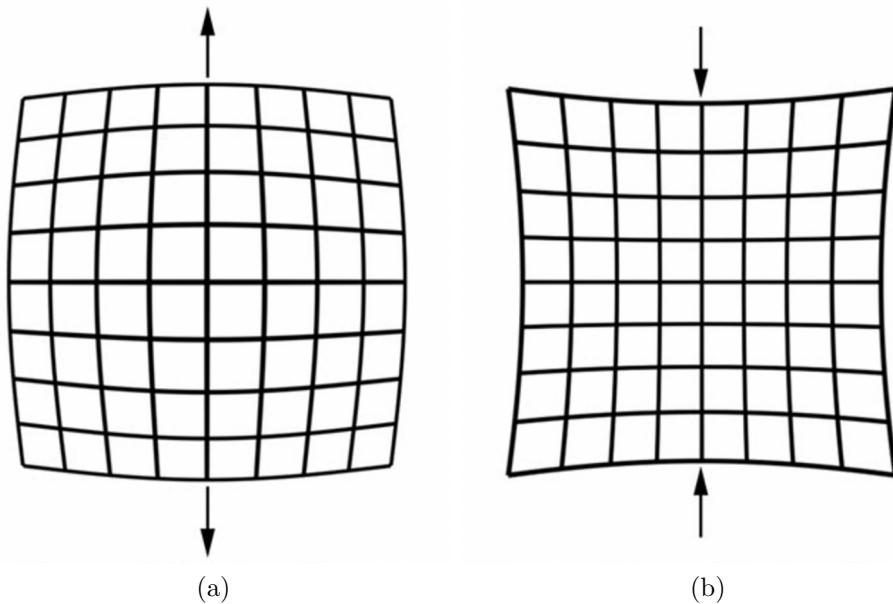


Abbildung 10: Verzerrung bei Bildern; 10a zeigt die Tonnen-Verzerrung (engl.: Barrel-Distortion) und 10b die Kissen-Verzerrung (engl.: Pincushion-Distortion) (<https://www.learningwithexperts.com/photography/blog/look-out-for-lens-distortion>)

Um dieser Verzerrung entgegenzuwirken gibt es in OpenCv die Methode **undistort()**. Diese bringt ein Bild wieder in den normalen Zustand und benutzt dabei die gefundenen Koeffizienten [OpenCv-Cam]. Das benutzen dieser Methode ist unumgänglich. Die Laserlinie ist bei einem ebenen Untergrund gerade. Wenn diese allerdings gebogen ist, führt das beim Aufspannen auf die Ebene zu Höhenunterschieden zwischen den Punkten. Die

3.1. Kalibrierung

Verzerrung ist nicht Teil des Pinhole Camera Models und somit auch kein Teil der Formel (1). Bevor mit aufgenommenen Bildern gearbeitet wird, werden die jedoch mit der Methode **undistort()** entzerrt.

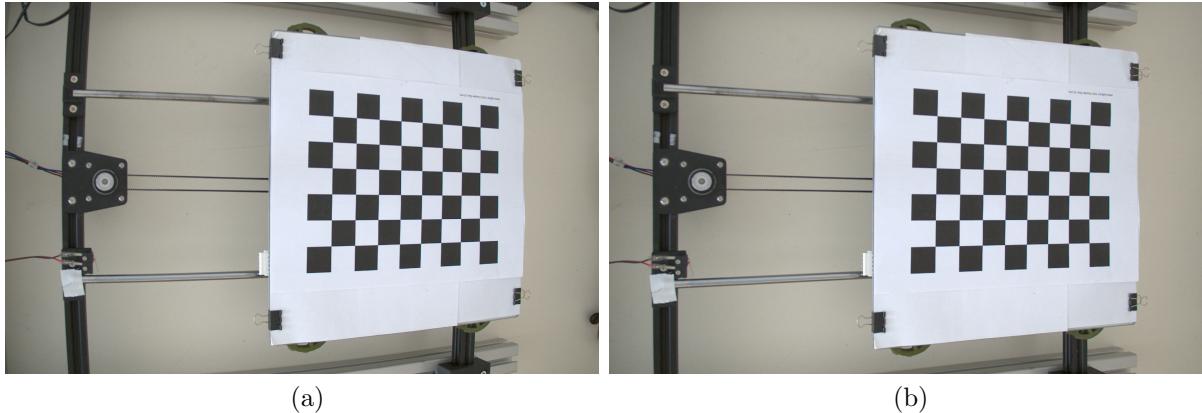


Abbildung 11: Hier ein direktes Beispiel aus dem Aufbau. 11a zeigt das Bild vor dem Anwenden von **undistort()** und 11b danach.

Ebenso ist erkennbar, dass es sich bei den verwendeten Kameramodell um eine *Barrel-Distortion* (siehe Abbildung 10a) handelt.

Rotationen und Translationen

Die Methode gibt ebenfalls eine Rotationsmatrix und einen Translationsvektor für jedes übergebene Schachbrett zurück. Das sind extrinsische Parameter. In jedes Schachbrett wird ein Koordinatensystem gelegt. Die Rotation und Translation beschreiben dann die Transformation von der Kamera zu diesem Koordinatensystem. Es werden also auch schon extrinsische Parameter geliefert. Diese sind allerdings für den Lasertriangulationssensor selbst nicht ausschlaggebend. Damit dieser mit dem Laser kalibriert werden kann, also eine Ebenen-Gleichung für den Laser gefunden wird, muss der Laser mit auf den Bildern abgebildet sein. Dann kann man die Linie in Bezug auf das gefundene Weltkoordinatensystem weiterverwenden. Für die extrinsische Kalibrierung wurde eine extra Methode entwickelt, welche im nächsten Kapitel beschrieben wird. Die Rotation und Translation zu den verschiedenen Positionen des Schachbrettes sind nicht von Interesse.

Die intrinsische Kalibrierung ist nur dazu da, die Kameramatrix zu finden und die Distortion-Koeffizienten zu bestimmen, damit die verwendeten Bilder nicht verzerrt sind. Die Kameramatrix ist explizit für die Kamera gültig, mit welcher das Schachbrettmuster aufgenommen wurde. Sie kann also bestimmt werden und ist dann für alle folgenden Rechnungen gültig und unverändert. Genauso verhält es sich mit den Distortion-Koeffizienten. Das bedeutet, dann der Lasertriangulationssensor initial einmal mit dieser Methode kalibriert

werden muss, um die intrinsischen Parameter zu bestimmen. Danach werden die Kamera-matrix und die Distortion-Koeffizienten abgespeichert und die extrinsische Kalibrierung kann beginnen.

3.1.2. Extrinsische Kalibrierung

Die extrinsische Kalibrierung beschäftigt sich allgemein damit die Kamera zu den eingebauten Linienlaser zu kalibrieren. Genauer gesagt geht es nicht konkret um den Laser, sondern die Laserlinie. Ziel dabei ist es, eine Ebenengleichung zu erhalten, die sich im Kamerakoordinatensystem befindet den Laser repräsentiert. Um sich das besser vorzustellen, kann nochmal Abb.: (1) eingesehen werden. Hier ist die symbolische Ebene des Lasers rot markiert. Die dazugehörige Gleichung beschreibt dann die Position des Laserlinienstrahls. Grundlegend richtet sich die Methode, um die Ebenen-Gleichung zu finden erneut nach Bajpai und Perelman [baj-per].

Bekannt ist, dass durch eine Kamerakalibrierung die Rotation und Translation herausgefunden werden können. Durch die intrinsische Kalibrierung ist die Kameramatrix bekannt. Dabei wurde ein Schachbrett aus verschiedenen Positionen aufgenommen. Ziel ist es nun ein Schachbrett mit einer Laserlinie aufzunehmen. Die allgemeine Transformationsformel (2) kann gelöst werden. Mit dem im Kapitel 2.5 vorgestellten Verfahren werden die Pixel, die die Laserlinie abbilden gefunden. Nur diese werden in die Formel (2) eingesetzt. Dabei können jetzt die 3D-Punkte errechnet werden, welche sich in dem Weltkoordinatensystem befinden. Das Weltkoordinatensystem liegt dabei auf dem Schachbrett und ist durch die Rotationsmatrix und den Translationsvektor bestimmt. Aus einer Linie kann jedoch keine Ebene abgeleitet werden. Was ist aber, wenn sich im selben Bild ein zweites Schachbrett befindet, welches nicht eben, sondern in einem gewissen Winkel zu dem ersten befindet. Das zweite Schachbrett hat dabei dann ein eigenes Weltkoordinatensystem. Da jedoch alle Transformationen bekannt sind, ist es möglich die Laserlinien-Punkte des einen Koordinatensystems in das andere zu transformieren. Betrachtet werden nun zwei Laserlinien, die in einem Winkel zueinander in einem einheitlichen Koordinatensystem stehen. Mit dieser Ausgangslage ist es möglich eine Ebene in die Punkte zu legen. Dabei gibt es diverse Möglichkeiten eine Ebene an Punkte zu fitten und alle liefern eine Ebenengleichung. Rein logisch sind sogar nur mindestens drei Punkte notwendig, um in einem dreidimensionalen Raum eine Ebene zu beschreiben. In diesem Fall sind es weitaus mehr Punkte, die zum Finden der Ebene berücksichtigt werden können.

In [baj-per] wird dann bei jedem Durchgang, bei dem eine Laserlinie in eine Punktewolke übersetzt wird, eine aktuelle neue Ebenengleichung erstellt und die Oberflächenpunkte errechnet. Die Ausgangslage ist hier jedoch etwas anders. Die Idee ist es nach der intrinsischen Kalibrierung eine einmalige extrinsische Kalibrierung vorzunehmen. Es wurde schon erwähnt, dass sich Kamera und Laser in ihrer Position zueinander nicht verändern. Damit bleibt auch die Ebenen-Gleichung in Bezug zur Kamera immer gleich. Die einmal gefundene Ebenengleichung ist also allgemeingültig für den folgenden Scann. Dieser Vorgang der extrinsischen Kalibrierung soll hier nochmal genau beschrieben werden.

Zwei Schachbretter in einem Bild

Um die extrinsische Kalibrierung auszuführen soll also ein Bild von zwei Schachbrettern gemacht werden. Dabei ist es nötig, dass sie in einem Winkel zueinander stehen. Das hat den Grund, dass die verschiedenen Laserlinien einen Winkel bilden müssen, um darin eine Ebene zu finden. Der Winkel ist dabei grundsätzlich egal. Jedoch sollte kein zu hoher oder zu niedriger Winkel gewählt werden, damit dieser nicht zu eng bzw. zu flach ausfällt. Zu diesem Zweck wurde eine eigene Unterlage designt.



Abbildung 12: Unterlage zum Kalibrieren

In diesem Fall wurde ein 90° -Winkel benutzt. Das Kalibrier-Brett ist über ein 3D-Drucker gedruckt wurden. Diese Bilder sind Beispielbilder. Der entwickelte Lasertriangulationssensor liefert während dem Kalibriervorgang Output-Bilder, die verschiedenen Arbeitsschritte untermalen.

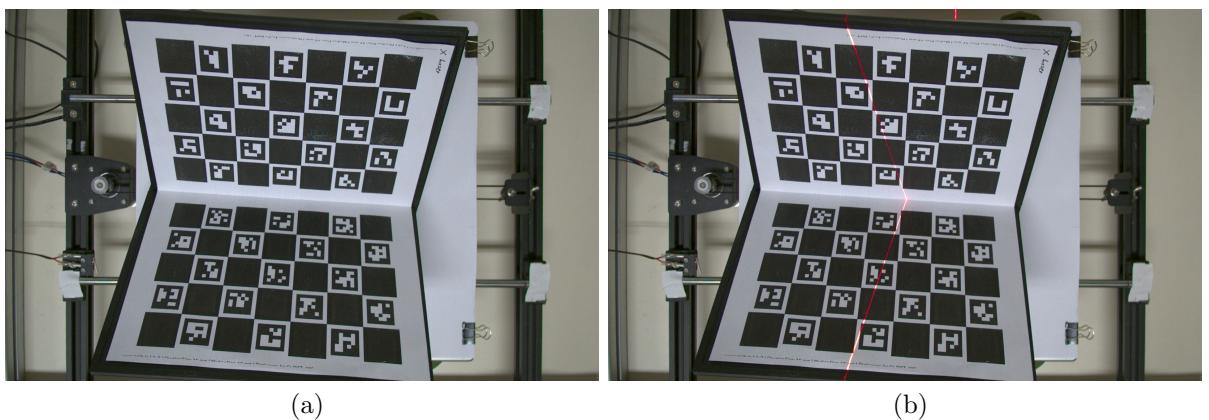


Abbildung 13: Ausgangsbilder der extrinsischen Kalibrierung

Gemäß Kapitel 2.5 wird ein Bild-Paar aufgenommen. In einem ist die Laserlinie sichtbar (Abbildung 13b) und in dem anderen nicht (Abbildung 13a).

3.1. Kalibrierung

Unterscheidung der Schachbretter

In den Vorherigen Abbildungen fällt auf, dass es sich bei den verwendeten Kalibriermuster nicht um ein herkömmliches Schachbrett handelt. Das verwendete Muster ist ein ChArUco-Board. Das Problem bei einem herkömmlichen Schachbrett ist, dass der Algorithmus für die Kamera-Kalibrierung in einem einzigen Bild nicht zwischen zwei Schachbrettern unterscheiden kann. Ein ChArUco-Board besitzt in den normal weißen Flächen eines Schachbretts sogenannte ArUco-Marker. Diese Marker können von der Kamera erkannt und eindeutig unterschieden werden. Das finden und unterscheiden der ChArUco-Board ist in OpenCv implementiert. Dabei definiert man, ähnlich zu dem Schachbrett aus der intrinsischen Kalibrierung 3.1.1, im Vorhinein die Parameter des Boards. Dazu gehören nicht nur die Kachel, sondern auch eine gewisse ArUco-Bibliothek, welche die Marker beschreibt. Der Algorithmus kann nun in einem Bild die beiden Boards unterscheiden und platziert ein Koordinatensystem darauf. Vorausgesetzt wird dabei, das die Kameramatrix und die Distortion-Koeffizienten bereits bekannt sind. Da die intrinsische Kalibrierung initial durchgeführt wird, ist diese Voraussetzung erfüllt.



Abbildung 14: Weltkoordinatensysteme im ChArUco-Board

Bekannt sind jetzt die Rotation und Translation von dem Kamerakoordinatensystem zu den zwei Weltkoordinatensystemen, die aus den ChArUco-Boards folgen.

Finden der richtigen Laserlinie

Der nächste Schritt muss es sein, die Laserlinie im Bild herauszuarbeiten. Wir wollen die 2D-Punkte der Laserlinie wissen und sie in die 3D-Repräsentation in dem Weltkoordinatensystem umwandeln. Kapitel 2.5 beschreibt die Methodik. Das Problem ist, dass nur der Teil der Laserlinie benutzt werden darf, der sich auch tatsächlich auf den jeweiligen Schachbrett befindet. Das bedeutet, dass nur ein gewisser Ausschnitt aus dem Bild wichtig ist. Und zwar genau das zu untersuchende ChArUco-Board.

Das ChArUco-Board selbst ist dabei die Lösung des Problems. OpenCv erkennt die so-

wohl die ArUco-Marker als auch die Ecken der Kachel. Dabei besitzen alle eine eindeutige ID. Über die Ecken kann also ein Bereich abgesteckt werden.

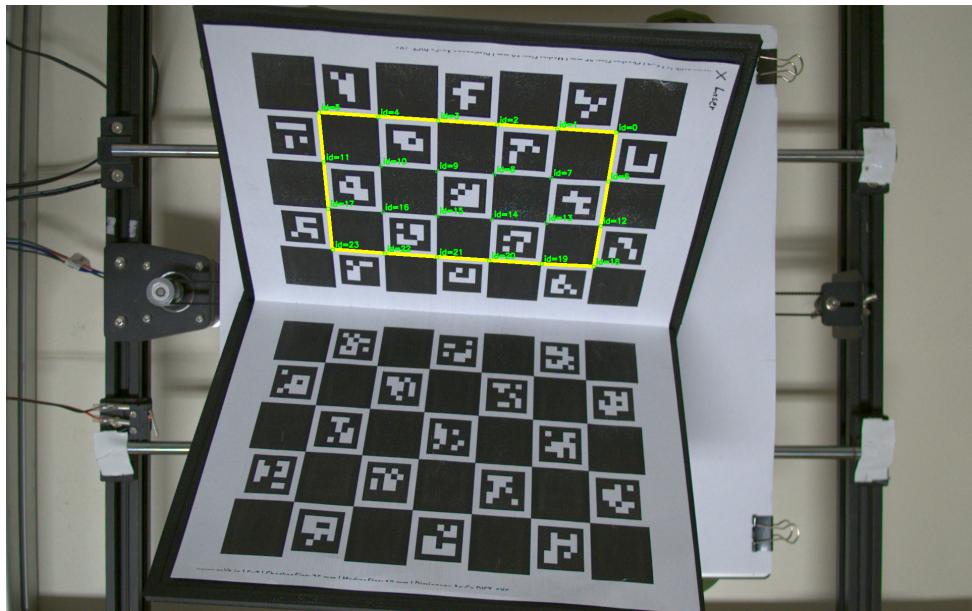


Abbildung 15: Point of Interest im Bild

Um den wichtigen Bereich abzustecken sind die Eckpunkte des Bereiches mit den IDs 0, 5, 18 und 23 ausschlaggebend. Wegen Fehleranfälligkeit wurde jedoch davon abgesehen, genau diese Ecken anzusprechen. Falls in einem Bild eine der äußereren Ecken nicht erkannt wurde, könnten man so den Bereich nicht abstecken. Deswegen wird eine Konvexe Hülle benutzt. Damit werden immer die äußersten Punkte berücksichtigt. Fall gewisse Ecken wegen einer Unschärfe im Bild nicht erkannt werden, führt das nun nicht zu Fehlern. In Abb. (15) ist die abgesteckte Fläche in Gelb eingezeichnet. Falls keine Ecke bzw. keine ID gefunden wurden, ist kein ChArUco-Board erkennbar und die Kalibrierung bricht ab.

Mit den bekannten Bereich kann eine Maske entwickelt werden, die auf das Bild angewandt werden kann. Die Maske besteht dabei aus einem schwarzen Bild, in dem der ausschlaggebende Bereich aus weißen Pixeln besteht. Wenn man das Ziel-Bild bitweise mit der Maske verundet, bleibt nur der markierte Bereich im bestehen. Alle andern Pixel sind schwarz. Ausgehend von den Ursprungsbildern (Abb. 13) werden für die verschiedenen ChArUco-Boards eine Maske entwickelt und angewandt. Das Ergebnis ist die folgende Abbildung.

Die Achsen des jeweiligen Koordinatensystems sind dabei nur zu Übersicht eingezeichnet und nicht Teil der Bildverarbeitung. Verdeutlicht werden soll jedoch, dass das Weltkoordinatensystem bekannt ist.

Die erstellte Maske kann auch auf das Ursprungsbild ohne Laser angewandt werden. Die

3.1. Kalibrierung

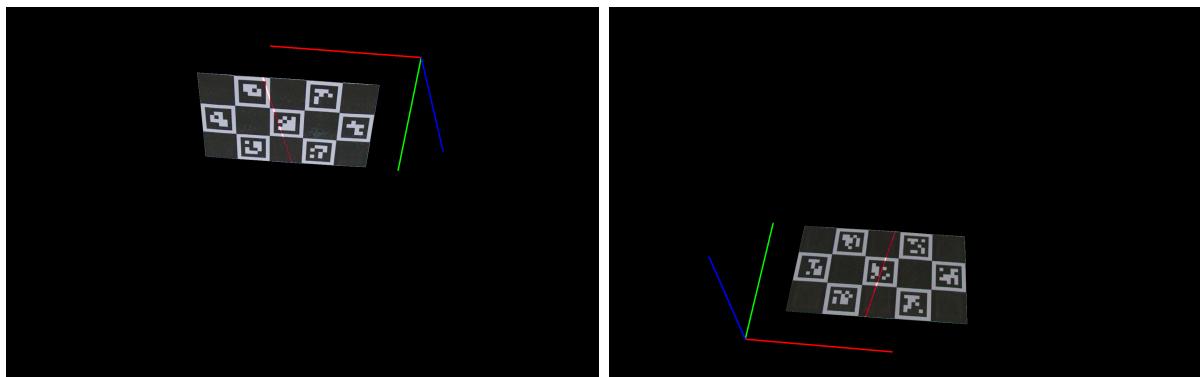


Abbildung 16: Ausgeschnitten Laserlinien

beiden ausgeschnittenen Bilder werden an den Algorithmus zum Finden der Laserlinie (Kapitel 2.5) weitergegeben. Die nächste Abbildung zeigt das Ergebnis des Subtrahierens der Bilder. Zur Übersicht sind wieder die Koordinatenachsen eingetragen. Tatsächlich sind an diesem Punkt als Ergebnis der Bildverarbeitung schon die Subpixel der Laserlinie bekannt. Subpixel sind schwierig in einem Bild darzustellen, deshalb wurde sich für das Zeigen des Zwischenschritts entschieden.

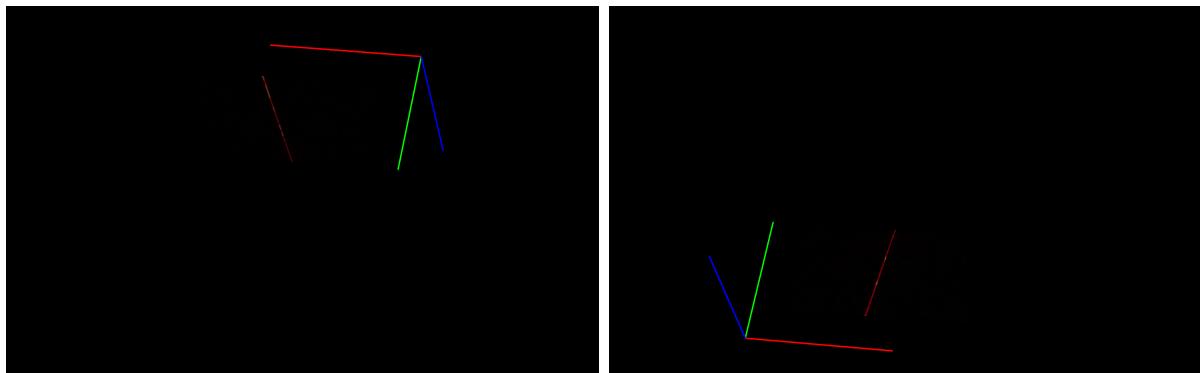


Abbildung 17: Herausgearbeitete Laserlinien

Der Scale – Factor

An diesem Punkt kommt es zum Einsatz der Formel für die 3D-Weltkoordinaten (1). Wie in den Kapitel 2.4.2 angemerkt, ist der Scale-Factor noch eine Unbekannte, die gefunden werden muss. Ohne sie wäre die Gleichung noch nicht lösbar. Dieses Problem kann wie folgt umgangen werden. Das Weltkoordinatensystem wird auf das ChArUco-Board gelegt. Die Laserlinie liegt ebenfalls genau auf dem Board. Das bedeutet, dass die Z-Koordinate der Laserlinien-Punkte 0 ist. Zu sehen ist das auch in Abb. 17. Die Laserlinie liegt in der XY-Ebene. Die Z-Koordinaten der Weltkoordinatenpunkte müssen 0 sein. Mit dem Ausgangspunkt von Gleichung (5) folgt daraus diese Umstellung:

$$p_w = s \vec{a} - \vec{b}$$

das entspricht :

$$\begin{aligned} x &= s a_x - b_x \\ y &= s a_y - b_y \\ z &= s a_z - b_z \end{aligned} \tag{12}$$

mit $z = 0$:

$$\begin{aligned} 0 &= s a_z - b_z \\ s &= \frac{b_z}{a_z} \end{aligned}$$

Damit ist die Basistransformationsgleichung lösbar und die 3D-Punkte können errechnet werden. Damit nun eine Ebene in die Punkte gelegt werden kann, müssen alle Punkte in ein einheitliches Koordinatensystem gelegt werden. Durch die bekannten Transformationen ist dies jedoch kein Problem. Zuerst werden mit (6) und der eigene Rotation und Translation die Weltpunkte der unteren Laserlinie in das Kamerakoordinatensystem transformiert. Danach wird (7) genutzt, um mit der Rotation und Translation der oberen Laserlinie die Punkte in das obere Koordinatensystem zu bringen. Die folgende Abbildung gilt als Symbolbild.

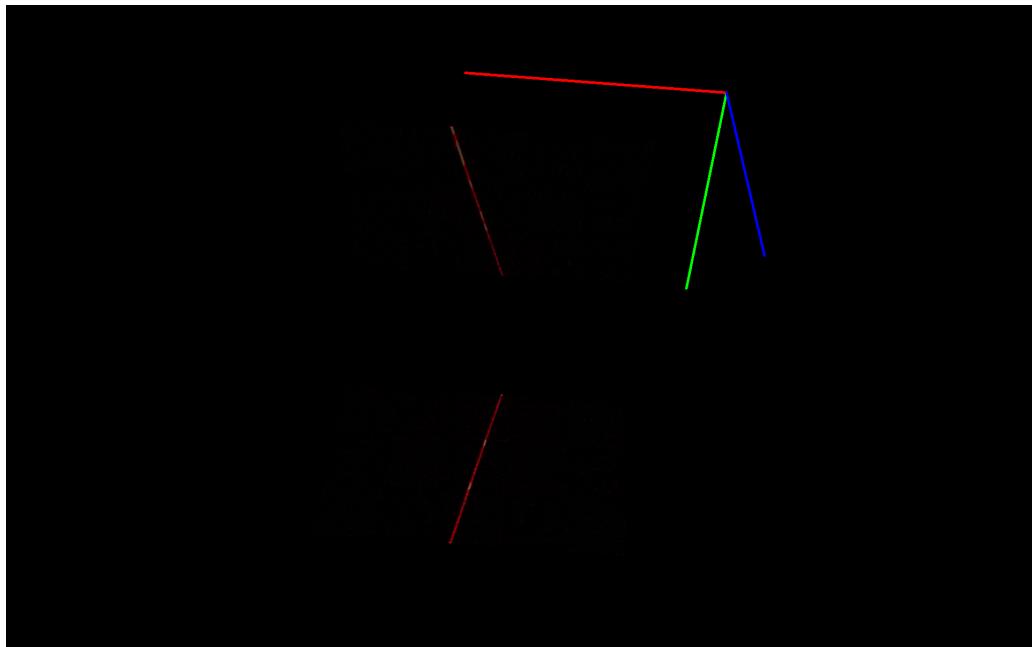


Abbildung 18: Laserlinien in einem Koordinatensystem

Das Bestimmen der Ebenengleichung

Mit dem letzten Schritt wird die gesuchte Ebenengleichung gefunden. Alle Vorbereitungen

sind abgeschlossen. Die richtigen Punkte der zwei nicht parallelen Laserlinien sind in einem einheitlichen Koordinatensystem. Im letzten Schritt muss in diese Punkte eine Ebene gefittet werden. Für einen Ebenen-Fit gibt es diverse Möglichkeiten. Hier wurde sich für die Singular Value Decomposition (SVD) entschieden [svd-t]. Sie ist über die Numpy-Bibliothek in Python implementiert und nutzbar. Wenn man über Numpy den SVD-Algorithmus anwendet, gibt dieser, wie in [svd-t] beschrieben drei Matrizen zurück. Diese heißen U, S und V. Nach [svd-fit] ist die dritte Zeile von U der Normalen-Vektor (\vec{n}) der passenden Ebene. Ziel ist es eine Ebenengleichung als Koordinatenform zu erhalten.

$$Ax + By + Cz + D = 0 \quad (13)$$

$$\vec{n} = \begin{pmatrix} A \\ B \\ C \end{pmatrix}$$

Um D zu erhalten kann ein zufälliger Punkt aus den Laserlinien-Punkten ausgewählt werden. Dieser wird dann für x, y und z eingesetzt und D kann errechnet werden. Damit ist das finden der Laser-Ebene abgeschlossen.

3.2. Das Erzeugen einer Punktewolke

Mit Beendigung von diesem Schritt ist die komplette Kalibrierung des Lasertriangulationssensors beendet. Bekannt ist nun die Laserebene. Mit Aufnahme der Laserlinie von einer Oberfläche kann nun eine Punktewolke erzeugt werden.

3.2.1. Die 3D-Koordinaten

Das Erzeugen der Punktewolke fängt ähnlich an, wie die extrinsische Kalibrierung. Die Grundlage sind wieder zwei Bilder. Aus diesen zwei Bildern werden die Laserlinien-Subpixel nach Kapitel 2.5.4 errechnet. Diese Pixel werden mithilfe der Ebenengleichung in die tatsächlichen 3D-Punkte aus Sicht des Kamerakoordinatensystem umgewandelt. Grundlage sind wieder die Transformationen aus Kapitel 2.4.2. Bei der extrinsischen Kalibrierung konnte der Scale-Faktor (s) nur mit der Annahme errechnet werden, dass in der X-Y-Ebene mit einer Z-Koordinate gleich 0 liegen. Diese Annahme gilt hier nicht mehr. Sicher ist jedoch jetzt, dass die Punkte in der gefundenen Ebenen-Gleichung liegen. Damit gibt es eine vierte geltende Gleichung und alle Unbekannten können errechnet werden. Die folgende Umstellung erläutert die geltenden Gleichungen. Die Ausgangsbasis ist erneut:

$$p_w = s \vec{a} - \vec{b}$$

das entspricht :

$$\begin{aligned} x &= s a_x - b_x \\ y &= s a_y - b_y \\ z &= s a_z - b_z \end{aligned} \tag{14}$$

für die Ebenengleichung gilt :

$$0 = Ax + By + Cz + D$$

Der ausschlaggebende Wert ist der Scale-Faktor. Dieser kann über ein einsetzen der Koordinaten in die Ebenengleichung bestimmt werden:

$$\begin{aligned} 0 &= A(s a_x - b_x) + B(s a_y - b_y) + C(s a_z - b_z) + D \\ 0 &= s A a_x - A b_x + s B a_y - B b_y + s C a_z - C b_z + D \\ 0 &= s A a_x + s B a_y + s C a_z - A b_x - B b_y - C b_z + D \\ 0 &= s(A a_x + B a_y + C a_z) - (A b_x + B b_y + C b_z) + D \\ \vec{n} &= \begin{pmatrix} A \\ B \\ C \end{pmatrix} \\ 0 &= s(\vec{n} \cdot \vec{a}) - (\vec{n} \cdot \vec{b}) + D \\ s &= \frac{\vec{n} \cdot \vec{b} - D}{\vec{n} \cdot \vec{a}} \end{aligned} \tag{15}$$

Der Scale-Faktor ist damit bekannt und die Koordinaten x, y und z können bestimmt werden. Wichtig hierbei ist, dass die Ebenengleichung in einem Weltkoordinatensystem liegt. Dementsprechend müssen die errechneten 3D-Punkte mit der bekannten Transformation (6) von dem Weltkoordinatensystem zum Kamerakoordinatensystem transformiert werden.

3.2.2. Das Finden der Farbinformationen

Zusätzlich gefordert zu den reinen 3D-Informationen waren auch Farbinformationen. Diese können ebenfalls über das grundlegende Bild-Paar gefunden werden. Das Bild, welches ohne das Einschalten des Lasers aufgenommen wird, beinhaltet die gesuchten Farbinformationen. Gesucht wird nur nach der Farbe an der richtigen Stelle. Die Pixel-Werte der Laserlinie sind bekannt. In dem Ursprungsbild muss an genau der Stelle, an der sich ein

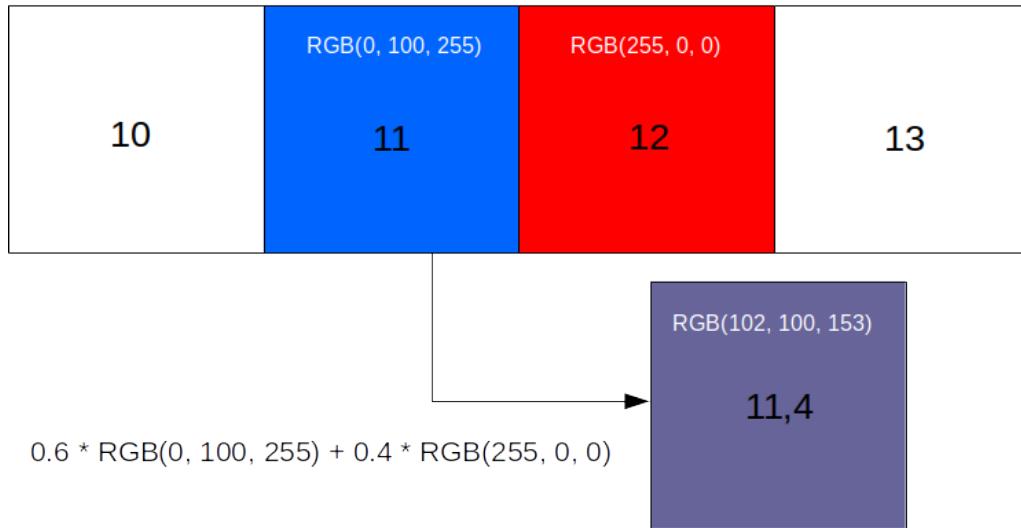


Abbildung 19: Gewichtete Pixel für den x-Wert 11,4

Laserpixel befindet die Farbe ausgelesen werden und für den jeweiligen Punkt abgespeichert werden. Problem hierbei ist, dass die Laserpixel Subpixel sind und sich demnach zwischen zwei Pixeln befinden. Für die endgültige Farbe werden dann die zwei umrandenden Pixel gewichtet zusammengerechnet. Die Abbildung 19 zeigt den Vorgang an einem simplen Beispiel für ein Subpixel in einer beliebigen Reihe (y-Wert) mit einem beispielhaften x-Wert von 11,4. Der nähere Pixel (11) fließt mit höheren Gewicht in die resultierende Farbe ein, als der Pixel weiter weg (12). Die y-Werte der Pixel können keinen ungeraden Wert beinhalten, da der Algorithmus zum Finden der Laser-Pixel (Kapitel 2.5.4) jede Pixel-Zeile im Bild durchläuft und dort einen neuen x-Wert errechnet.

Dieser komplette Vorgang transformiert genau eine aufgenommene Laserlinie in eine Punktewolke. Um eine ganze Oberfläche zu rekonstruieren muss sich die Position der des Sensors um eine bekannte Länge und Richtung verschieben. Die neu Punktewolke, die aus der neu aufgenommenen Laserlinie resultiert, muss dann entsprechend zu einer Gesamt-Punktewolke ergänzt werden. Um die neuen Punkte korrekt einzufügen, ist es zwingend erforderlich, dass die Bewegung genau bekannt ist.

3.3. Aufbau / Hardware

Für den Lasertriangulationssensor wurde verschiedene Hardware und Bauteile benutzt. Diese sollen im folgendem beschrieben werden. Dabei gibt es zwei verschiedene Abschnitte zu betrachten. Zum einen der Lasertriangulationsensor an sich, der nur in der Lage ist eine Laserlinie in eine Punktewolke umzusetzen. Zum anderen wurde ein Versuchsaufbau angefertigt, um ein Objekt unter dem Sensor entlang zu bewegen. Dabei soll der Sensor regelmäßig eine Punktewolke erstellen. Die zusammengefügte Punktewolke ist dann die Aufnahme der Oberfläche.

3.3.1. Hardware-Aufbau des Lasertriangulationssensors

Die Hardware des Lasertriangulationssensor besteht grundsätzlich aus einer Kamera und einem Laser.



Abbildung 20: Der Lasertriangulationssensor

Für die Kamera wurde die Basler a2A1920-160ucPRO verwendet. Dabei handelt es sich um eine Industriekamera mit einer Auflösung von 1920 x 1200 Pixeln. Sie ist eine Farbkamera und kann bis zu 164 Bilder pro Sekunde aufnehmen. Die Spezifikationen sind gemäß [basler-cam-spez] angeben. Wichtig ist, dass die Kamera über USB 3.0 angesprochen werden kann. So kann sie über Software auf einem PC oder ähnlichem angesteuert werden. Mit dem Software-Paket Pylon liefert Basler eine eigene Umgebung um die Kamer zu benutzen und einzustellen. Zusätzlich wird mit Pypylon eine Pythonbibliothek gestellt. Diese gilt als Schnittstelle zwischen Kamera und Python. So kann über direkten Python-Code die Kamera eingebunden werden.

Das zweite Bauteil ist der Laser. Dabei handelt es sich um einen herkömmlichen roten Laser. Der Lichtstrahl des Lasers wird durch diffraktive optische Elemente (DOE) zu einer Linie geformt. Wie in Abb. 20 gezeigt werden dann Kamera und Laser an eine Halterung in entsprechender Position befestigt.

3.3. Aufbau / Hardware

Die Kamera hat ein ansteuerbares Output-Signal. Dieses Signal kann genutzt werden, um von der Kamera direkt den Laser anzusprechen. Dabei wird ein MOSFET (Metal Oxide Semiconductor Field-Effect Transistor) verwendet. So kann das Output-Signal angesprochen werden, um über den MOSFET den Strom des Lasers an und aus zu stellen. So kann in der Kamera selbst ein Ablauf definiert werden, in dem sie zuerst ein Bild aufnimmt, dann den Laser anschaltet und ein zweites Bild aufnimmt. Zuletzt muss der Laser wieder ausgeschaltet werden. So werden von der Kamera die geforderten Bildpaare aufgenommen. Mit Pypylon können diese Bilder in Python und mit OpenCv weiterverwendet werden.

3.3.2. Der Versuchsaufbau

Das Ziel des Versuchsaufbau ist es Bewegung in die aufgenommene Szene zu bringen. Dabei soll die zurückgelegte Bewegung zwischen den Aufnahmen der Oberfläche bekannt sein. So kann die neue Punktewolke der Laserlinie passend angefügt werden. Andernfalls würde man immer den Querschnitt der aktuellen Oberfläche an der Position des Lasers beobachten. Die einfachste Möglichkeit das zu realisieren ist, wenn sich die Szene in genau eine definierte Richtung mit genauen definierten Abstand bewegt. Dazu wurde eine Linear-Führung benutzt. Diese Funktioniert über einen Schrittmotor und bewegt eine Fläche mit einer linearen Bewegung. Der Schrittmotor kann dabei genau angesprochen werden und das Objekt um in unserem Fall genau einen Millimeter weiter schieben. Die

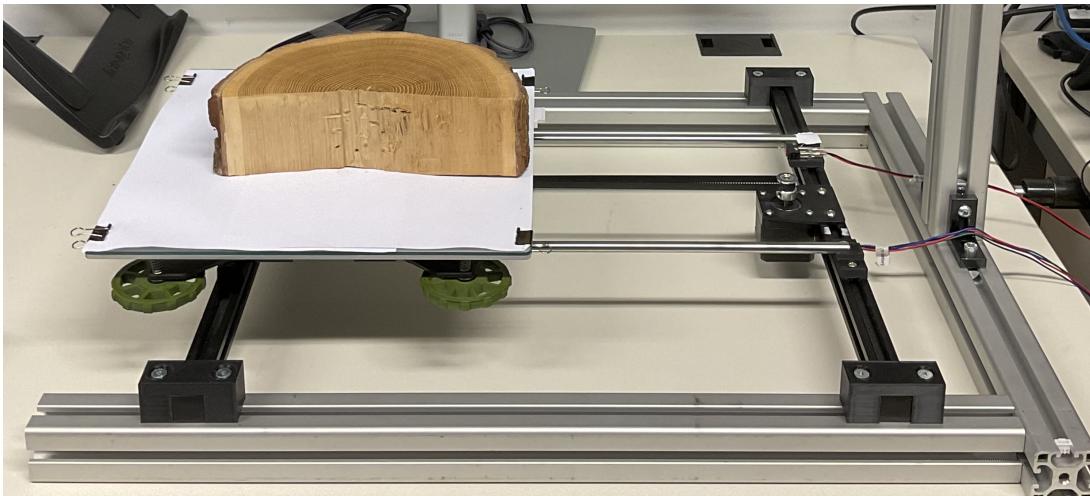


Abbildung 21: Der Lasertriangulationssensor

Linear-Führung stammt aus einem 3D-Drucker. Anzusteuern von der Software des Lasertriangulationssensors ist der Schrittmotor. Dieser wird über eine CNC-Steuerung mit Arduino und GRBL-Software angesprochen. Der Arduino ist der Steuerungs-Rechner. Die CNC-Steuerung übernimmt das Ansteuern des Motors. Dazu gibt es ein geeignete

Erweiterungsplatinen für einen Arduino, sogenannte CNC-Shields. Damit Arduino und CNC-Shield miteinander kommunizieren können wird die Open-Source-Software GRBL verwendet. So können Befehle des Arduino über GRBL und der CNC-Steuerung für den Schrittmotor lesbar sein. Mit diesem Aufbau kann über die Bibliothek *serial* in Python

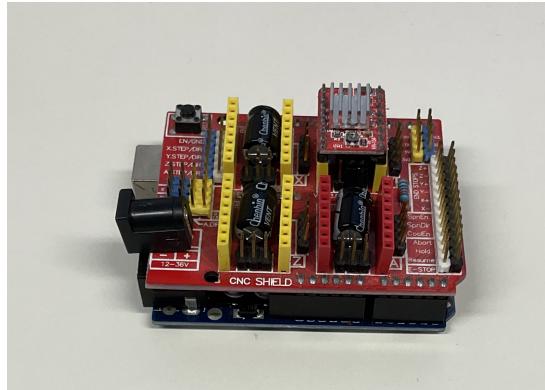


Abbildung 22: Arduino mit CNC-Shield

eine sogenannte „Serial Connection“ aufgebaut werden. So kann direkt über ein USB-Anschluss auch der Motor angesprochen werden. Über die Serial Connection werden dann GRBL-Befehle an den Motor gesendet. Diese sind einfache Befehle wie „Startposition festlegen!“, „1mm bewegen“ oder „Zur Startposition zurückkehren!“.

3.4. Aufbau / Software

Die Software des Lasertriangulationssensor gliedert sich in zwei Bereiche. Zum einen wurde eine Python-Bibliothek entwickelt. Diese wurde Objektorientiert mit diversen Klassen, die den Lasertriangulationssensor repräsentieren entwickelt. Grundlegende Aufgabe der Python-Bibliothek sind die elementaren Berechnungen, um eine Punktewolke aus einem Bild-Paar zu erhalten. Das bedeutet, dass ein Konstrukt erstellt werden kann, welches Kalibriert werden kann und darauf folgend Punktewolken liefert.

Der andere große Bereich ist die Umsetzung in ROS2. ROS2 liefert das Betriebssystem des Scanners. Alle verschiedenen Elemente sollen zu einem Gesamtprodukt zusammengefügt werden. Dazu gehört die Ansteuerung der Kamera und des Lasers. Schwerpunkt ist auch die Kommunikation zwischen einzelnen Elementen. Beispiele sind dabei das Aufnehmen der Bilder für die Kalibrierung oder das Erstellen der Punktewolke und die Weiterleitung an die entsprechenden Methoden aus den Klassen der Python-Bibliothek.

3.4.1. Python (Bibliothek)

Die Python-Bibliothek besteht grundlegend aus vier Klassen. Die Scanner-, Kamera-, Laser-, und LaserLine-Klasse. Dabei ist die Scanner-Klasse die volle Repräsentation des Lasertriangulationssensors, die in ROS2 eingebunden wird.

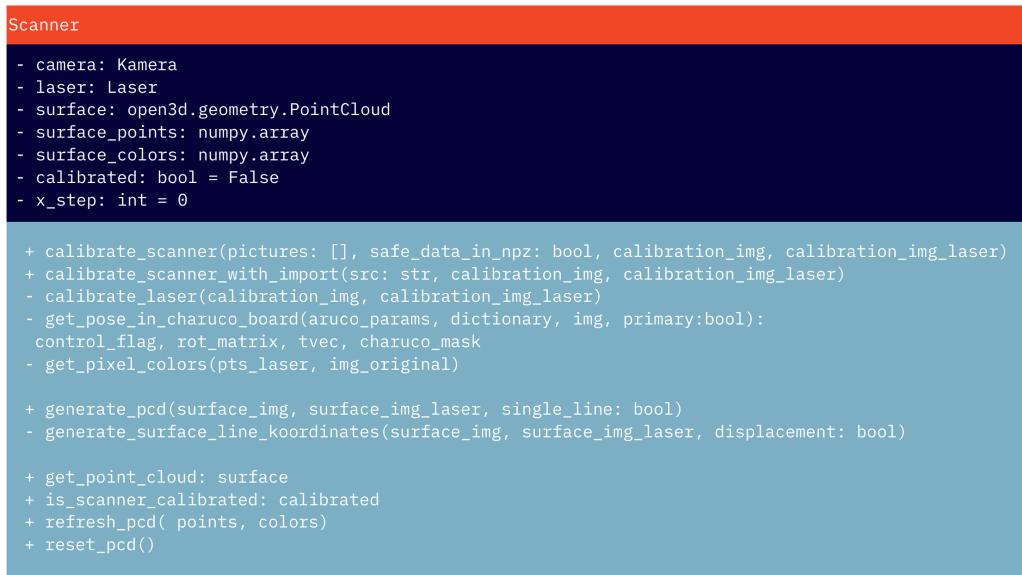


Abbildung 23: Die Scanner-Klasse als UML-Diagramm

Der Scanner beschreibt den Lasertriangulationssensor. Er besitzt eine Kamera und einen Laser. Zusätzlich wird als Variable die Punktewolke der Oberfläche und auch direkt deren Punkte und Farben abgespeichert. Hinzu kommt ein Boolean-Flag um zu kennzeichnen,

ob der Scanner kalibriert ist. Der xStep ist für den Versuchsaufbau wichtig. Über die Liniarführung wird das Objekt in x-Richtung unter dem Sensor durch bewegt. Die jeweiligen Millimeter-Schritte müssen entsprechend einbezogen werden. Dazu wird diese Variable genutzt.

Zusätzlich stellt die Scanner-Klasse die grundlegenden Methoden, die die Funktionalität des Triangulationssensors wieder spiegel, zur Verfügung. Diese umfassen zum einen das komplette kalibrieren des Scanners. Dazu wird eine Liste von Bildern für die intrinsische Kalibrierung und ein Bildpaar des ChArUco-Boards für die extrinsische Kalibrierung benötigt. Hinzu kommt eine abgewandelte Version der Kalibrier-Methode, in der die intrinsische Kalibrierung durch das Importieren der Kameradaten ersetzt wird. Hinzu kommt die Methode zum Erstellen einer Punktewolke. Sie bekommt ein entsprechendes Bildpaar von der Oberfläche übergeben und errechnet die Punktewolke. Wenn der Scanner sich gerade in einem Scavorgang befindet wird die neue Punktewolke entsprechend an die gesamte Punktewolke angehängt. Die Scanner-Klasse besitzt zusätzlich noch Hilfsmethoden die Zugriff von außen auf den Scanner Status und die Punktewolke geben oder die Punktewolke selbst erneuern oder aktualisieren.

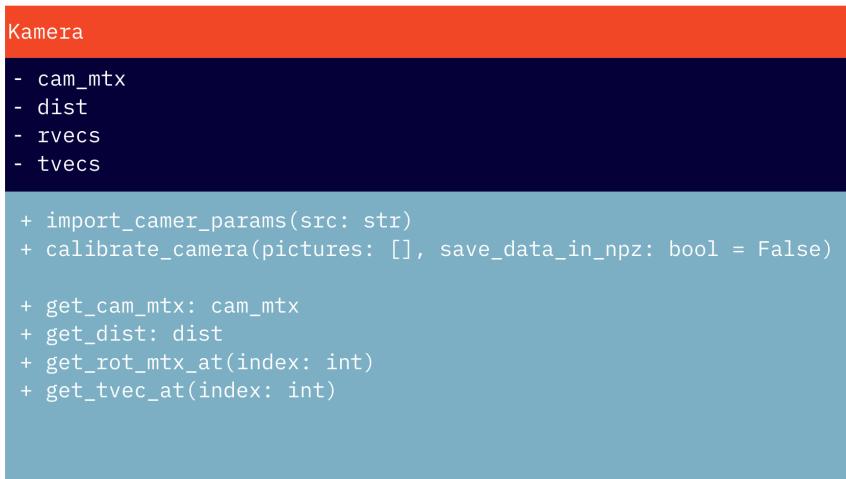


Abbildung 24: Die Kamera-Klasse als UML-Diagramm

Die Kamera selbst besitzt auch eine Repräsentation in Form einer Klasse. Diese Klasse fungiert als Speicher für die Kameradaten. Gespeichert werden die Kamera-Matrix die Distortion-Koeffizienten und die Rotationsmatrizen sowie die Translationsvektoren der Bilder von der intrinsischen Kalibrierung. Also alle Daten, die aus der intrinsische Kalibrierung hervorgehen. Hierbei ist wichtig hervorzuheben, dass die Klasse keine Bilder aufnehmen kann bzw kein Zugriff auf die tatsächliche Kamera im Triangulationssensor hat. Sie ist lediglich für das Ablegen der Daten und die intrinsische Kalibrierung zuständig.

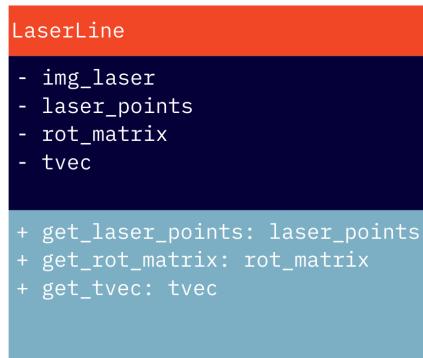


Abbildung 25: Die LaserLine-Klasse als UML-Diagramm

Die Klasse LaserLine beschreibt eine aufgenommene Laserlinie. Information die dafür als Variablen gespeichert werden sind das Differenz-Bild, die errechneten 3D-Punkte der Linie und die entsprechende Rotation und Translation zu dem Weltkoordinatensystem, in dem sich die Laserlinie befindet. Zum initialisieren der LaserLine-Klasse wird das aufgenommene Bildpaar und die Rotation und Translation zu dem jeweiligen Weltkoordinatensystem benötigt. Die Klasse errechnet dann noch im Konstruktor die Bild-Differenz und die Laserlinienpunkte.

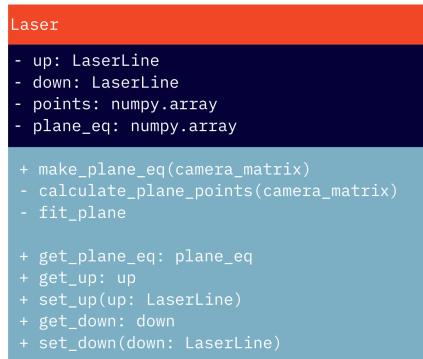


Abbildung 26: Die Laser-Klasse als UML-Diagramm

So wie die Kamera-Klasse die Daten der intrinsischen Kalibrierung beinhaltet, beinhaltet die Laser-Klasse die Daten der extrinsischen Kalibrierung. Dazu benötigt sie zwei Laserlinien der Klasse LaserLine, die zur Unterscheidung als „oben“ und „unten“ definiert sind. Aus diesen kann die Ebenengleichung bestimmt werden. Dazu wird eine eigene Methode bereitgestellt in der erst die Punkte in ein einheitliches Koordinatensystem zusammengefasst und dann eine Ebene an die Punkte gefittet wird. Die vereinheitlichten Punkte und die Ebenengleichung werden von der Klasse gespeichert.

```
Hilfsmethoden außerhalb der Klassen

# Transformationen
world2cam(pts, trans_vec, rot_matrix)
cam2world(pts, trans_vec, rot_matrix)
bild2world(pts, trans_vec, rot_matrix, cam_matrix plane = None)

# Plane-Fitting
plane_fit_with_svd(points)
plane_fit_with_pyransac3d(points)

# Berechnung der Subpixel
get_line_pixel(diff_img_laser)
```

Abbildung 27: Die Hilfsmethoden ohne Zugehörigkeit zu einer Klasse

Zusätzlich zu den vier beschriebenen Klassen gibt es Hilfsmethoden, die nicht konkret zu einer Klasse gehören. Das bedeutet, dass sie nicht nur von einer Klasse benutzt werden, sondern von verschiedenen Klassen an verschiedenen Stellen aufgerufen werden. Diese Methoden umfassen die Transformationen zwischen den Koordinatensystemen, die Methoden für das Plane-Fitting und die Berechnung der Subpixel aus einem Differenz-Bild.

Die gesamte Darstellung des UML-Diagramm befindet sich im Anhang unter Abbildung A.1.

3.4.2. ROS2

Übergeordnet über der Python-Bibliothek befindet sich die ROS2-Implementierung, mit der letztendlich der Triangulationssensor bedient wird. Mit ROS2 erstellt man sogenannte **Nodes**, die verschiedene Möglichkeiten bereitstellen miteinander zu kommunizieren. Dazu gehören zum einen **Services**. Services sind am ehesten mit Methoden einer Klasse zu vergleichen, wobei dann die Nodes eine Klasse darstellen. Eine Node stellt ein Service bereit. Dieser kann aufgerufen werden und man erhält eine Antwort zurück. Der Triangulationssensor selbst ist auch über Nodes umgesetzt. Dabei sind die beiden ausschlaggebenden Nodes die Scanner-Node und die Kamera-Node. Die Scanner-Node enthält die vorgestellte Python-Bibliothek und damit eine Instanz von der Scanner-Klasse. Die Instanzen von der Scanner-Klasse selbst, also der Laser und die Kamera sind nach Initialisierung noch nicht mit Daten gefüllt. Das liegt daran, dass die Kalibrierung noch nicht stattgefunden hat. Dazu gibt es die Kamera-Node, die nun konkrete Bilder liefern kann. Die Kamera-Node hat Zugriff auf die Pypylon-Bibliothek [pypylon] und damit auf die Repräsentation der Basler-Kamera. Die Node beinhaltet eine Funktion, die das gewünschte Bildpaar liefert. Da der Linienlaser selbst über den Output der Kamera gesteuert wird, kann dieser auch direkt in der Kamera-Node angesprochen werden. So stellt die Kamera ein Service bereit,

der ein Bildpaar aufnimmt und dieses an den Scanner weiterleiten kann. Genauso stellt sie einen Service bereit, der mehrere Bilder aufnimmt und diese als Liste weiterleitet. Damit ist auch für die intrinsische Kalibrierung gesorgt.

Diese Kommunikation funktioniert in ROS2 über **Topics**. Topics beschreiben eine Nachricht. Eine Node hat die Möglichkeit eine Nachricht über ein gewähltes Topic zu veröffentlichen (publish). Ebenfalls gibt es die Möglichkeit, dass eine Node Nachrichten über ein gewähltes Topic empfängt (subscribe). Der Service hier nimmt ein Bildpaar auf und veröffentlicht es über das jeweilige Topic. Was die Scanner-Node beim Eintreffen der Bilder macht, ist für den Service selbst nicht mehr von Belangen.

In der Kommunikation muss zusätzlich festgehalten werden, welchen Datentyp die Nachricht bzw. das Topic verwenden soll. In ROS2 wird das über einen **Message-Type** realisiert. Hierbei kann man selbst eine Message erstellen oder man benutzt Vorgefertigte, die von ROS2 bereitgestellt werden. Für ein Bild gibt es einen vorgefertigten Message-Type. Da jedoch zwei Bilder auf einmal geschickt werden sollen, wurde hier ein eigener Typ (img_pair.msg) erstellt. Ähnlich verhält es sich bei dem Schicken der Kalibrierungsbilder für die intrinsische Kalibrierung. Hier wird ein Message-Type erstellt, der eine Liste an Bildern beinhaltet (cam_calib_imgs.msg). Die Kamera nimmt demnach bei einem Aufruf des Services die Bilder auf und veröffentlicht sie über das jeweilige Topic. Der Scanner ist ein Subscriber zu diesem Topic und empfängt sie. Dabei wird in der Node eine Callback-Funktion aufgerufen, die auf dieses Empfangen reagiert. Der Scanner selbst speichert die für die Kalibrierung gesendeten Bild erst einmal nur ab. Zusätzlich hat ein Service in ROS2 die Möglichkeit ein Feedback zurückzugeben. Diese Möglichkeit wird hier genutzt, damit die Kamera zeigen kann, dass die Bilder erfolgreich aufgenommen und veröffentlicht wurden.

Die Scanner-Node stellt einen eigenen Service bereit, der den Scanner Kalibriert. Dazu werden die Methoden, die die Scanner-Klasse bereitstellt genutzt. Überprüft wird hierbei, ob dem Scanner schon die benötigten Bilder bereitstehen. Falls das nicht der Fall ist, wird über die Antwort des Services darauf hingewiesen und der Aufruf des Services schlägt fehl. Andernfalls kann die Node den Scanner Kalibrieren und ein Scan kann gestartet werden. In der ROS2-Implementierung gibt es noch zwei weitere Nodes. Zum einen wird ein Empfänger gebraucht, der entsprechende Ergebnisse des Scanners auch empfangen und benutzen kann. In diesem Fall werden von Scanner Punktewolken veröffentlicht. Das geschieht auch wieder über ein Topic. Dabei ist der Message-Type des Topics ein von ROS2 vorgefertigter Typ, der Punktewolken beinhaltet. Benötigt demnach wieder eine Node, die ein Subscriber des Punktewolken-Topic ist. Dazu wird **Rviz2** benutzt. Rviz2 ist ein von ROS2 bereitgestellten Paket und dient zur Visualisierung. Die Funktionalität von Rviz2 ist sehr weitgehend. Wichtig für den Lasertriangulationssensor ist, dass es Rviz2 möglich

Subscriber von dem Punktewolken-Topic zu sein. Die Empfangenen Punktewolken können hier dargestellt und dynamisch beobachtet werden.

Die vierte Instanz, die benötigt wird, sind sogenannte Clients. Es wurde schon öfters von Services und dem Aufrufen eines Service geredet. In ROS2 wird dafür ebenfalls eine Node benötigt, die den jeweiligen Service aufruft und das entsprechende Feedback erhält. In der Anlage unter A.2 befindet sich eine Grafik, die die Nodes und ihre Kommunikationswege darstellt. Die Client-Node wird dort als eine einzelne Node zusammengefasst. Intern wird für jeden Aufruf eines Services jedoch eine eigene individuelle Node generiert. Da diese allerdings alle den selben Zweck folgen, den bestimmten Service aufrufen und das Feedback ausgeben, wurden sie hier als eine einzelne Node zusammengefasst. Als Nutzer startet man die jeweiligen Clients, um das gewollte System zu starten. Voraussetzung dabei ist, dass die Service-Nodes, hier der Scanner und die Kamera, ebenfalls gestartet und aktiv sind. Diese werden genauso, wie ein Client gestartet, nur dass sie selbst nicht aktiv etwas tun, es sei denn der Service, den sie bereit stellen wird aufgerufen oder ein Topic von dem sie Subscriber sind wird veröffentlicht. Sie warten also auf eine Interaktion. Die Client-Nodes sind dabei die Akteure, die eine Interaktion hervorrufen. Das Starten der Nodes ist in ROS2 über Konsolen-Skripts umgesetzt. Die wichtigen Skripts, die von einer gewissen Aktionskette gefolgt sind, sollen im Folgenden beschrieben werden. In A.2 wurden diese Skripts als Funktionen (Functions) benannt, da sie nicht einfach nur eine Node starten, sondern im ganzen System etwas zum Laufen bringen.

Vorgang bei der Kalibrierung des Lasertriangulationssensors

Die jetzt beschriebenen Vorgänge sind im Anhang unter A.3 in einem Sequenz-Diagramm visualisiert.

Der Scanner braucht für eine Kalibrierung Bilder für die intrinsische und für die extrinsische Kalibrierung. Da das sich die Szene der Bilder ändert, findet zwischen den Kameraaufnahmen keine Automatisierung statt. Das bedeutet, dass manuell zuerst die Bilder-Liste der intrinsischen Kalibrierung und das Bild-Paar der extrinsischen Kalibrierung von der Kamera geschickt werden muss, bevor die Kalibrierung selbst erfolgen kann. Welche Aufnahme zuerst erfolgt, ist dabei nicht wichtig. In A.3 wird zuerst die Funktion **intr_calib_imgs** aufgerufen. Diese startet ein Client, der den Service der Kamera **send_cam_calib_imgs** aufruft. Die Kamera-Node benutzt daraufhin ihre Funktion zum Aufnehmen der Kalibrierbilder. Der Benutzer muss dann für jedes Bild das Schachbrett in einer anderen Position vor die Kamera halten. Die Kamera-Node veröffentlicht die Bild-Liste und schickt ein positives Feedback an den Client zurück. Der Client gibt das Feedback in der Konsole aus und ist damit fertig. Die Node hört somit auf zu existieren. Der Scanner reagiert auf das veröffentlichen der Bilder und speichert sie ab.

Der selbe Vorgang passiert bei der Funktion `extr_calib_imgs`. Der Unterschied besteht darin, dass der Client einen anderen Service (`send_laser_calib_imgs`) aufruft. Der Benutzer muss hier das Kalibrier-Brett mit dem ChArUco-Board vor die Kamera halten.

Der Scanner ist nun im Besitz der benötigten Bilder. Mit der Funktion `calibrate` wird ein Client gestartet, der den Service `calibrate_laser` der Scanner-Node aufruft. Das veranlasst die Scanner-Node dazu, die Funktion `calibrate_scanner()` der internen Scanner-Klasse aufzurufen. Für die Visualisierung und zum Überprüfen erstellt die Scanner-Node eine Punktwolke, die die Laserebene darstellt und veröffentlicht sie über ein Topic. Über Rviz2 kann dann das Ergebnis der Kalibrierung angesehen werden.

Vorgang des Scannablauf

Die jetzt beschriebenen Vorgänge sind im Anhang unter A.4 in einem Sequenz-Diagramm visualisiert.

Nachdem der Scanner kalibriert ist, kann ein Scann-Vorgang, zugeschnitten auf den Versuchsaufbau vorgenommen werden. Hierbei wird kein konkreter Service der Scanner-Node benutzt. Die Scanner-Node weiß viel mehr, dass das gesendete Bild-Paar von der Oberfläche aufgenommen wurde und eine Punktwolke errechnet werden soll. Das verwendete Topic (*ImgPair*) hat einen Message-Type, der zwei Bilder beinhaltet. Dieser kann somit sowohl für die extrinsische Kalibrierung, als auch für ein Bildpaar von der Oberfläche genutzt werden. Hier wurde ein Boolean-Flag eingebaut, der zeigt, ob das Bild-Paar für die Kalibrierung oder zur Oberflächenrekonstruktion benutzt werden soll. Somit weiß die Scanner-Node genau, wie sie weiterzuarbeiten hat. Falls die Scanner-Node ein Bildpaar zur Oberflächenrekonstruktion erhält, aber nicht kalibriert ist, wird der Fehlerfall mit einer entsprechenden Warnung behandelt.

Die Funktion `start_scan` startet einen Client, der zum Anfang der Scanner-Node über das Topic `ScannerStatus` signalisiert, dass sie sich jetzt im laufenden Scann befindet. Der Scann startet damit, dass der Service `send_img_pair_surface` der Kamera aufgerufen wird. Die Kamera nimmt ein Bild-Paar auf und veröffentlicht es. Der Scanner reagiert darauf, in dem er das Bildpaar in eine Punktwolke umsetzt. Diese wird dann ebenfalls veröffentlicht und in Rviz2 dargestellt. Mit dem Senden des Bild-Paares schickt die Kamera ein positives Feedback an den Client zurück. Der Client signalisiert dann der Serial-Connection über die Sprache Grbl, dass sich die Linearführung um einen Millimeter bewegen soll. Danach wiederholt sich der Vorgang mit dem Aufrufen des Services der Kamera. Dabei bestimmt der Client wie oft sich der Vorgang wiederholt. Im Versuchsaufbau bewegt sich die Linear-Führung 290mm, somit wird der Vorgang 290 mal wiederholt. Die Scanner-Node selbst zählt auch mit, wie oft ein Bild-Paar veröffentlicht wurde und rechnet die Verschiebung mit ein. Die aktualisierte Punktwolke wird dabei immer selbst

neu veröffentlicht und in Rviz2 aktualisiert.

Nachdem alle Wiederholungen durchgeführt wurden signalisiert der Client der Scanner-Node, dass der Scann fertig ist. Daraufhin wird der Scanner die finale Punktewolke in einem (.ply)-File speichern und dann die intern gespeicherte Punktewolke auf null setzen. Damit ist er bereit für einen erneuten Scann und behält seine Kalibrierung bei.

3.5. Qualitative Ergebnisse

3.6. Evaluation

3.6.1. Testen von Genauigkeit

3.6.2. Fehler

3.6.3. Probleme und Schwierigkeiten

4. Fazit

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier

ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

5. Ausblick

5.1. Erweiterung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

5.2. Anpassungen

wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

5.2. Anpassungen

A. Anhang

A.1. Anhang A



Abbildung 28: UML-Klassendiagramm des Lasertriangulationssensor

A.2. Anhang B

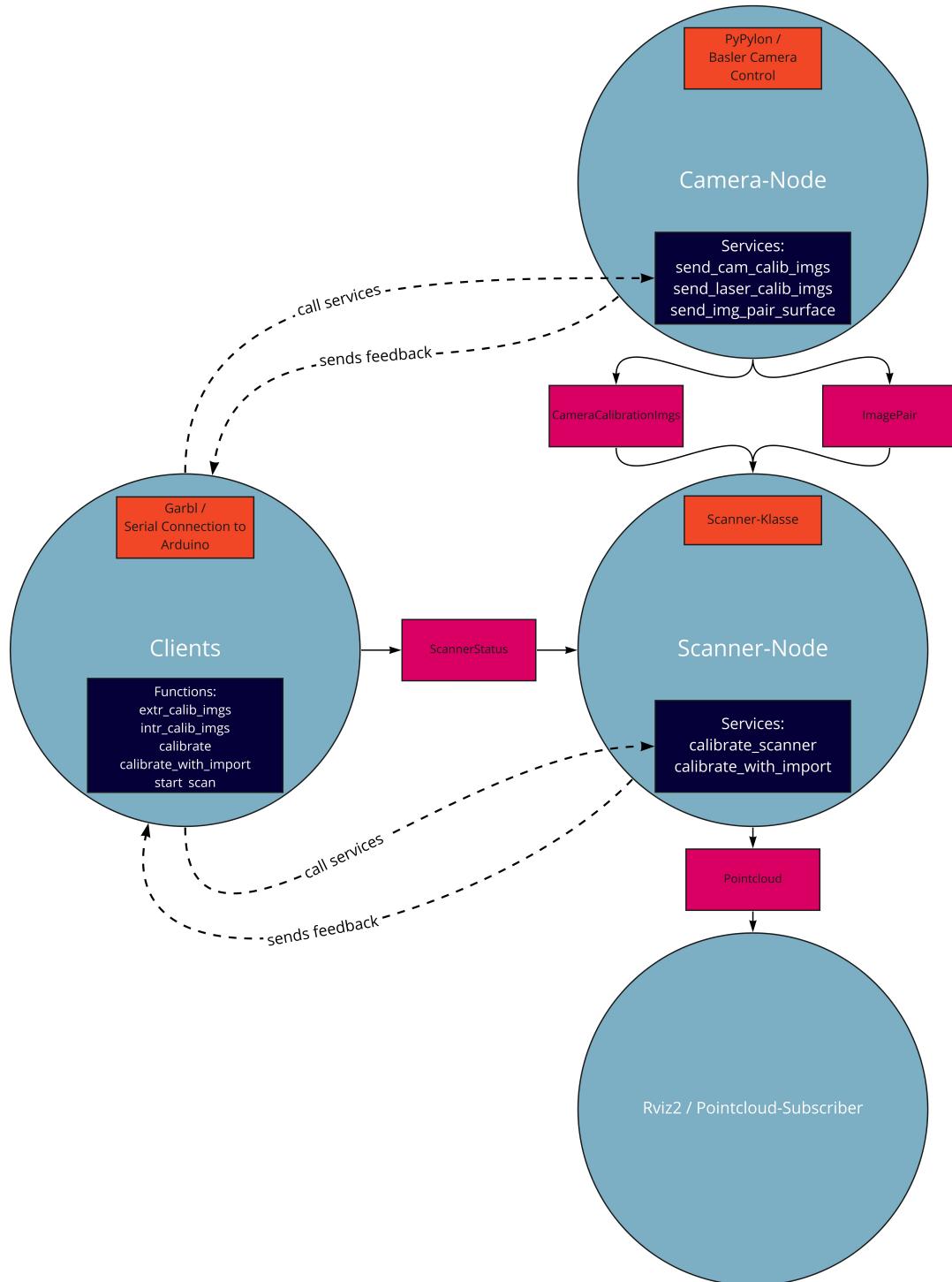


Abbildung 29: ROS2-Implementierung des Lasertriangulationssensor

A.3. Anhang C

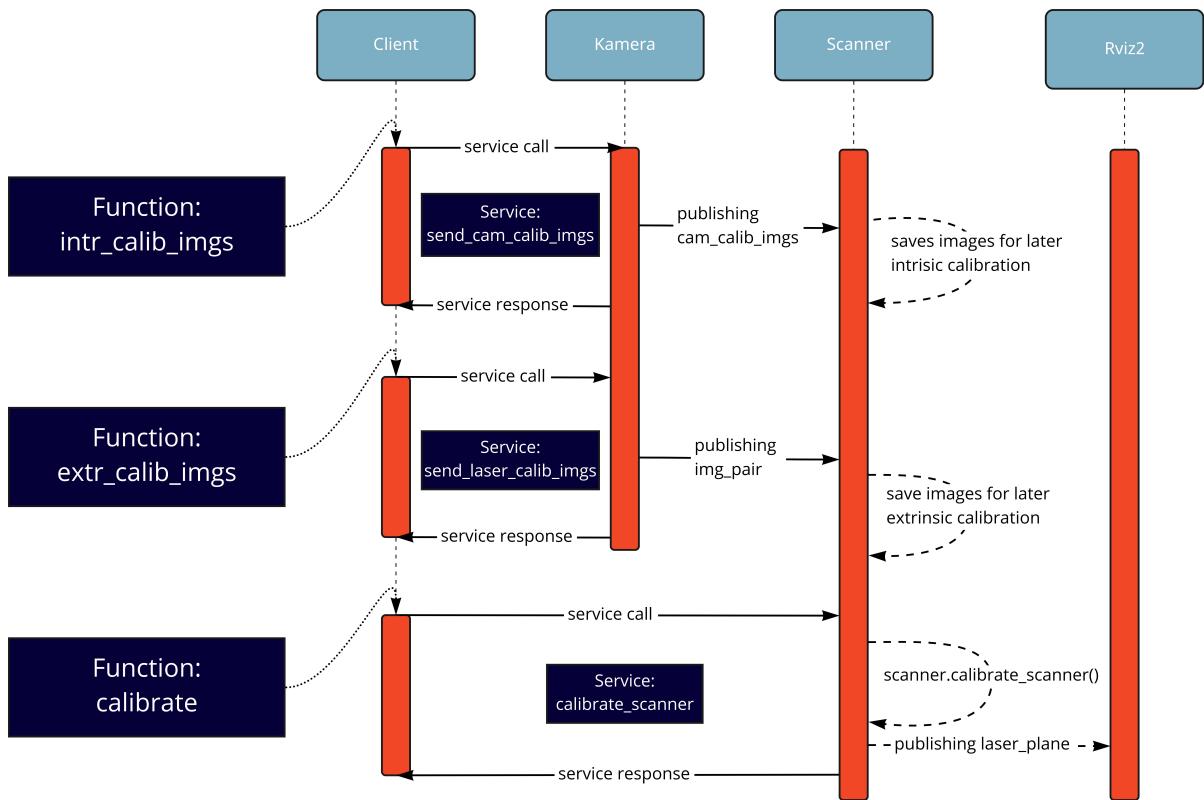


Abbildung 30: Vorgang der Kalibrierung

A.4. Anhang D

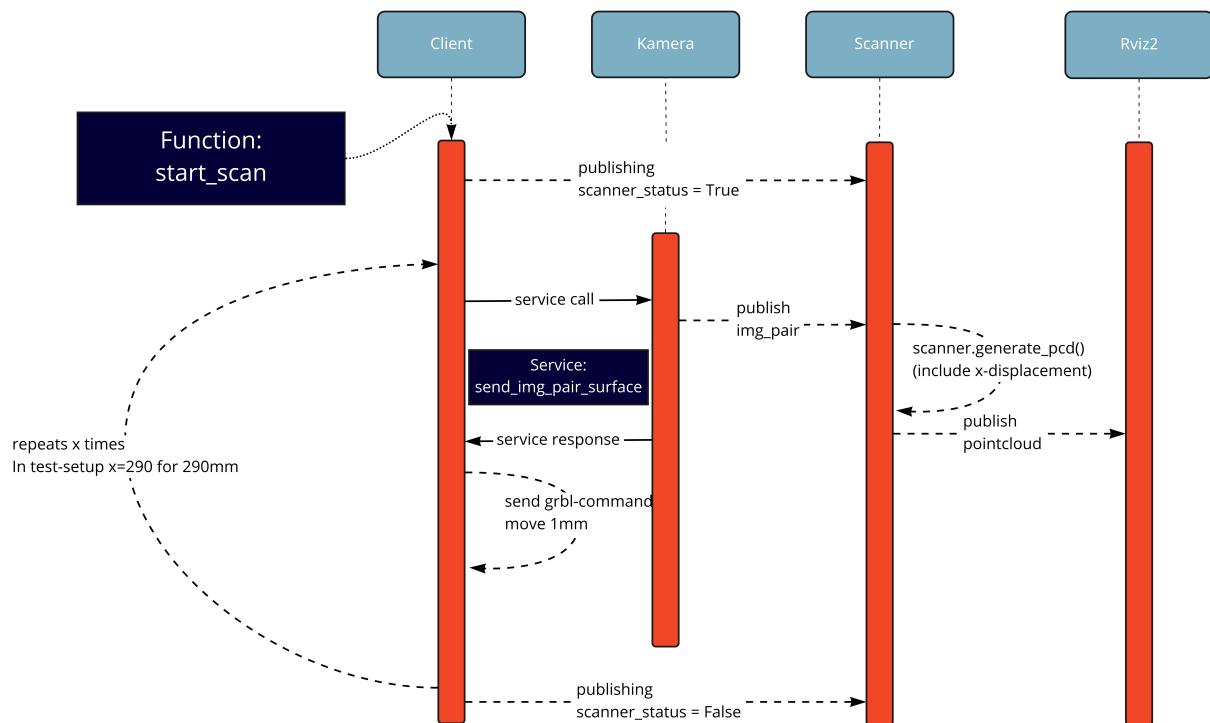


Abbildung 31: Vorgang des Scanablaufs

Abbildungsverzeichnis

Abb. 1	Lasertriangulation	4
Abb. 2	Positionen bei der Lasertriangulation	7
Abb. 3	test	8
Abb. 4	Transformationen	11
Abb. 5	Subtraktion der Bilder	14
Abb. 6	Pixel-Werte in einem Ausschnitt aus 5c am Rand der Laserlinie	15
Abb. 7	Der selbe Ausschnitt wie in Abbildung 6, nur als Grauwertbild konvertiert.	16
Abb. 8	Die Subpixel können nun in einem Koordinatensystem dargestellt werden.	19
Abb. 9	Schachbrett-Kalibrierung	20
Abb. 10	Verzerrung bei Bildern	21
Abb. 11	Beispiel für die Verzerrung in Bildern	22
Abb. 12	Unterlage zum Kalibrieren	25
Abb. 13	Ausgangsbilder der extrinsischen Kalibrierung	25
Abb. 14	Weltkoordinatensysteme im ChArUco-Board	26
Abb. 15	Point of Interest im Bild	27
Abb. 16	Ausgeschnitten Laserlinien	28
Abb. 17	Herausgearbeitete Laserlinien	28
Abb. 18	Laserlinien in einem Koordinatensystem	29
Abb. 19	Gewichtete Pixel für den x-Wert 11,4	32
Abb. 20	Der Lasertriangulationssensor	33
Abb. 21	Der Lasertriangulationssensor	34
Abb. 22	Arduino mit CNC-Shield	35
Abb. 23	Die Scanner-Klasse als UML-Diagramm	36
Abb. 24	Die Kamera-Klasse als UML-Diagramm	37
Abb. 25	Die LaserLine-Klasse als UML-Diagramm	38
Abb. 26	Die Laser-Klasse als UML-Diagramm	38
Abb. 27	Die Hilfsmethoden ohne Zugehörigkeit zu einer Klasse	39
Abb. 28	UML-Klassendiagramm des Lasertriangulationssensor	49
Abb. 29	ROS2-Implementierung des Lasertriangulationssensor	50
Abb. 30	Vorgang der Kalibrierung	51
Abb. 31	Vorgang des Scanablaufs	52

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Gießen, 31.05.2022

Ort, Datum

Unterschrift