Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Vincent Schröder

Entwicklung einer Panoramakamera mit stereoskopischen Eigenschaften zur Kollisionsvermeidung bei mobilen Robotern

Fakultät Technik und Informatik

Faculty of Engineering and Computer Science

Vincent Schröder

Entwicklung einer Panoramakamera mit stereoskopischen Eigenschaften zur Kollisionsvermeidung bei mobilen Robotern

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Mechatronik
an der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Stephan Schulz
Zweitgutachter : Prof. Dr.-Ing. Jörg Dahlkemper

Abgegeben am 19. Januar 2018

Vincent Schröder

Thema der Bachelorthesis

Entwicklung einer Panoramakamera mit stereoskopischen Eigenschaften zur Kollisionsvermeidung bei mobilen Robotern

Stichworte

Digitale Bildverarbeitung, Homographie, Panoramabild, Kamerakalibrierung, Stereoskopie, Disparitätenkarte, 3D-Rekonstruktion, Punktwolke, Octree

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit der Entwicklung eines Kamerasystems zur Kollisionsvermeidung in der mobilen Robotik. Dabei werden die Bilder mehrerer Kameras zusammengeführt, um eine Panoramaansicht der Umgebung zu erhalten und mögliche Hindernisse zu erkennen. Für eine autonome Lösung wird zudem Stereoskopie angewandt. Durch die Kombination von zwei Kameras aus verschiedenen Betrachtungswinkeln können Entfernungen zu Objekten ermittelt werden.

Vincent Schröder

Title of the thesis

Development of a Panoramic Camera with Stereoscopic Features for Collision Avoidance in Mobile Robots

Keywords

Digital image processing, homography, panoramic image, camera calibration, stereoscopy, disparity map, 3d reconstruction, point cloud, octree

Abstract

This thesis deals with the development of a camera system for collision avoidance in mobile robotics. The images of multiple cameras are stitched together to get a panoramic view of the environment and detect possible obstacles. In addition stereoscopy is used for an autonomous solution. Two cameras from different angles can be combined to measure distances to objects.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Eine zunehmende Anforderung in der Robobotik besteht in der autonomen Fortbewegung. Um diese zu gewährleisten ist die Kollisionsvermeidung ein notwendiger Bestandteil eines solchen mobilen Roboters.

Diese Bachelorarbeit befasst sich mit der Entwicklung eines Kamerasystems zur Hinderniserkennung. Ziel ist es, sich in der Umgebung orientieren und frühzeitig Bedrohungen erkennen zu können. Sowohl visuell durch den Bediener am Monitor als autonom durch eine automatische Erkennung von annähernden Objekten. Relevant ist dies zum Beispiel bei Drohnen

Bei der visuellen Erkennung sollen Hindernisse durch eine Panoramasicht wahrgenommen werden. Die autonome Kollisionserkennung die in dieser Arbeit vorgestellt werden soll basiert dabei auf den Prinzipien der Stereoskopie. Durch die Kombination von zwei Kameras aus verschiedenen Betrachtungswinkeln können Entfernungen zu Objekten ermittelt werden.

Das bekannteste Beispiel für Stereoskopie ist das Sehvermögen des Menschen, es beruht auf der versetzten Anordnung der Augen und damit der Betrachtung aus verschiedenen Blickwinkeln. Was das menschliche Gehirn intuitiv abschätzt, muss in der Bildverarbeitung mittels Triangulation berechnet werden.

1.1. Struktur der Arbeit

Insgesamt in acht Kapitel ist die vorliegende Bachelorthesis gegliedert. Der Aufbau ist so strukturiert, dass die Praxis nie weit von der Theorie entfernt ist. Zunächst einmal werden

nach der Einleitung in Kapitel 2 nur die wichtigsten theoretischen Grundlagen erläutert, die als Basis für das Verständnis der nachfolgenden Kapitel dienen.

Im Anschluss an die Grundlagen werden im 3. Kapitel die Anforderungen an das Projekt zusammengefasst und das Konzept vorgestellt. Der experimentelle Aufbau des Systems wird beschrieben und dabei die Konstruktion sowie verwendete Hardware gezeigt.

In den Kapiteln 4, 5 und 6 wird immer tiefer in die Materie vorgedrungen und dabei die Theorie direkt anhand von praktischen Beispielen erklärt. Anschließend wird jeweils die Implementierung beschrieben, woraufhin in jedem Kapitel eine Bewertung der Ergebnisse folgt. Die Intention hinter diesem Aufbau ist eine anschaulichere Darstellungsweise als lediglich die gesamten Grundlagen in einem Kapitel am Anfang zusammenzutragen. So kann der Prozess von den theoretischen Erkenntnissen über die praktische Umsetzung bis hin zu den Messergebnissen verfolgt werden.

Mit der Erstellung eines Panoramabildes aus mehreren Kameras beschäftigt sich das 4. Kapitel, wobei im Wesentlichen der mathematische Algorithmus vorgestellt wird.

In Kapitel 5 wird die Arbeit um die stereoskopische Komponente erweitert. Es wird erklärt, was die Voraussetzungen für eine Stereokamera sind, wie diese erreicht werden und wie aus zweidimensionalen Bildern Informationen über die räumliche Tiefe gewonnen werden.

Dies stellt die Basis für das nachfolgende 6 Kapitel dar, in dem die Erzeugung einer Punktwolke aus einem stereoskopischen Tiefenbild gezeigt wird. Mithilfe dieser Wolke sollen abschließend Objekte, die sich der Stereokamera nähern, erkannt und damit mögliche Kollisionen verhindert werden.

Kapitel 7 fasst die Erkenntnisse der gesamten Arbeit zusammen, betrachtet sie kritisch und gibt einen Ausblick auf Verbesserungsmöglichkeiten.

2. Elementare theoretische Grundlagen

In diesem Kapitel wird die grundlegende Theorie beleuchtet, die vorausgesetzt wird, um die nachfolgenden Kapitel zu verstehen. So wird die Beziehung zwischen den Punkten zweier Bilder durch geometrische Transformationen beschrieben. Diese werden im ersten Abschnitt 2.1 vorgestellt. Man unterscheidet zwischen der affinen Abbildung (2.1.1) und der perspektivischen Projektionen (2.1.2).

Letztere ist Grundlage für die Bildaufnahme in optischen Systemen. In Abschnitt 2.2 wird das allgemeine Kameramodell erklärt, mit dem in der Bildverarbeitung und Computergrafik gearbeitet.

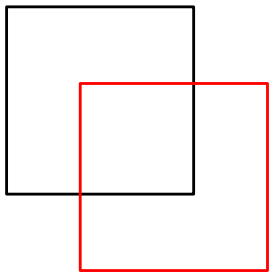
2.1. Geometrische Bildtransformationen

2.1.1. Affine Abbildung

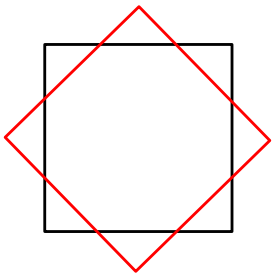
Die affine Abbildung zählt zu den geometrischen Bildtransformationen und ist eine Abbildungstechnik, die im Allgemeinen die Verschiebung, Drehung, Vergrößerung und Verkleinerung von Objekten in einem Koordinatensystem oder zwischen mehreren Koordinatensystemen beschreibt. Die Methode wird verwendet um zum Beispiel Abbildungsfehler zu korrigieren, die durch Weitwinkellinsen entstehen [21].

Dabei setzen sich affine Abbildungen aus einer Translation (siehe Abbildung 2.1a), also der reinen Verschiebung eines Objektes im Raum, bei der die ursprüngliche Form nicht verloren geht, und linearen Abbildungen zusammen. Letztere beinhalten Rotation (2.1b), Skalierung (2.1c) und Scherung (2.1d) eines Objektes in einem Koordinatensystem und

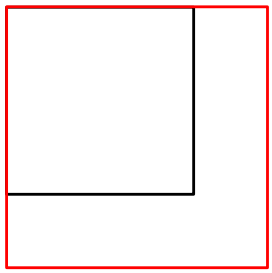
verändern dessen ursprüngliche Form. Bei der affinen Abbildung bleiben Punkte erhalten, Geraden in Geraden über und Parallelen gehen in Parallelen. Aus diesem Grund wird sie auch als Parallelprojektion bezeichnet.



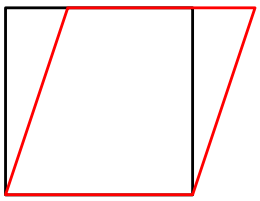
(a) Translation



(b) Rotation



(c) Skalierung



(d) Scherung

Abbildung 2.1.: Zu sehen sind die elementaren affinen Abbildungen eines planaren Oberflächenelements.

Eine affine Abbildung kann im Zweidimensionalen wie folgt durch eine Matrixmultiplikation und eine Vektoraddition ausgedrückt werden, x und y entsprechen dabei den Koordinaten im Ursprungsbild und x' und y' den Koordinaten im transformierten Bild [16]:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{2.1}$$

Für die Translation ergeben sich die Freiheitsgrade t_x und t_y und für die linearen Abbildungen die Freiheitsgrade a_{11}, a_{12}, a_{21} und a_{22}.

Um die Vektoraddition in die Multiplikation einzufügen und damit den Umgang mit den Transformationen zu vereinfachen, werden homogene Koordinaten verwendet. Es wird eine weitere Dimension eingeführt, jedem Punkt in der Gleichung 2.1 wird eine zusätzliche Koordinate zugeordnet, wobei die Umrechnung in 2D-Koordinaten durch Division der x- und y-Komponente durch die zusätzliche Koordinate erfolgt. Deshalb wird als Koordinate in der Regel eine Eins verwendet. Mithilfe der homogenen Koordinaten kann die affine Abbildung anschließend durch nur eine einzige Matrixmultiplikation beschrieben werden [16]:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.2}$$

Eine Transformationsmatrix, die eine reine Translation von einem Objekt beschreibt, sieht wie folgt aus. Die Parameter t_x und t_y stehen für die Verschiebung entlang der x- bzw. y-Achse [21]:

$$\mathbf{T_t} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

In den vier Freiheitsgraden der linearen Abbildungen a_{11}, a_{12}, a_{21} und a_{22} sind Skalierung, Rotation und Scherung enthalten. Diese jeweiligen Transformationen können auch in einzelne Matrizen zerlegt werden.

Eine Matrix, die ein Objekt in einem Koordinatensystem skaliert, enthält die Skalierungsfaktoren s_x und s_y in x- bzw. y-Richtung:

$$\mathbf{T_s} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Für die Rotation eines Objektes im Raum benötigt es folgende Transformationsmatrix. Der

Winkel φ gibt die Drehung in der xy-Ebene an:

$$\mathbf{T_r} = \begin{bmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Die Matrix für eine reine Scherung ergibt sich mit den Scherungsfaktoren sh_x und sh_y entlang der x- bzw. y-Achse an:

$$\mathbf{T_{sh}} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Die einzelnen Transformationsmatrizen ergeben multipliziert miteinander die Gleichung 2.1 mit den Parametern t_x, t_y, a_{11}, a_{12}, a_{21} und a_{22}. Dabei ist jedoch zu beachten, dass das Multiplizieren nicht beliebig erfolgen kann. Matrizenmultiplikation ist nicht kommutativ, die Reihenfolge der Terme ist entscheidend [35].

2.1.2. Perspektivische Projektion

Die affine Abbildung ist eine Parallelprojektion und reicht nur im Fall eines kleines Gesichtsfeldes für die Abbildung in optischen System aus [16]. Für alle anderen Fälle gibt es die perspektivische Projektion. Diese wird folgendermaßen ausgedrückt:

$$\begin{bmatrix} w' \cdot x' \\ w' \cdot y' \\ w' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} w \cdot x \\ w \cdot y \\ w \end{bmatrix} \tag{2.3}$$

Es sind im Vergleich zur affinen Transformation (Gleichung 2.2) zwei Koeffizienten hinzugekommen: a_{31} und a_{32}, die die perspektivische Projektion beschreiben. In der Bildverarbeitung wird die Transformationsmatrix aus acht Koeffizienten auch als Homographie bezeichnet.

Bei der Methode werden Linien in Linien überführt. Die Parallelität der Linien bleibt aber nur erhalten, wenn sie parallel zu Projektionsebene liegen. Die Abbildung von einem Rechteck ergibt ein allgemeines Viereck. Aus diesem Grund wird die Transformation auch Vierpunkt-Abbildung genannt. Im Gegensatz zur affinen Abbildung ist die Transformation nicht linear.

Um die insgesamt acht unbekannten Koeffizienten zu berechnen, sind vier oder mehr korrespondierende Punkte nötig. Dazu wird die Transformation in Gleichung 2.3 zuerst einmal in Standardkoordinaten umgeformt:

$$x' = a_{11} \cdot x + a_{12} \cdot y + a_{13} - a_{31} \cdot x \cdot x' - a_{32} \cdot y \cdot x'$$
$$y' = a_{21} \cdot x + a_{22} \cdot y + a_{23} - a_{31} \cdot x \cdot y' - a_{32} \cdot y \cdot y'$$

Mit dieser Gleichung wird ein lineares Gleichungssystem aufgestellt, das mit der Methode der kleinsten Quadrate gelöst werden kann [16]:

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \\ x'_N \\ y'_N \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1 \cdot x'_1 & -y_1 \cdot x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1 \cdot y'_1 & -y_1 \cdot y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2 \cdot x'_2 & -y_2 \cdot x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2 \cdot y'_2 & -y_2 \cdot y'_2 \\ & & & & \vdots & & & \\ x_N & y_N & 1 & 0 & 0 & 0 & -x_N \cdot x'_N & -y_N \cdot x'_N \\ 0 & 0 & 0 & x_N & y_N & 1 & -x_N \cdot y'_N & -y_N \cdot y'_N \end{bmatrix} \cdot \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix}$$

2.2. Bildaufnahme

Um nachzuvollziehen wie allgemein in der Computergrafik und im Speziellen in OpenCV Bilder aufgenommen werden, ist es notwendig, das Modell der Lochkamera zu kennen. Es handelt sich dabei um die einfachste Art und Weise, Objekte auf einer Fläche abzubilden. Eine Lochkamera besteht aus einem lichtundurchlässigen Gehäuse mit einer kleinen Öffnung auf der Vorderseite, durch das Lichtstrahlen eindringen können (siehe Abbildung 2.2). Ein Objekt, das sich vor diesem Loch befindet, wird umgekehrt und spiegelverkehrt auf die Rückwand des Gehäuses projiziert. Ist diese transparent, kann das Bild auch von außen betrachtet werden. In der Computergrafik wird das Lochkameramodell idealisiert und von einer unendlich kleinen Lochblende ausgegangen, sodass immer nur genau ein Lichtstrahl von einem Punkt des Objektes auf die Rückwand trifft.

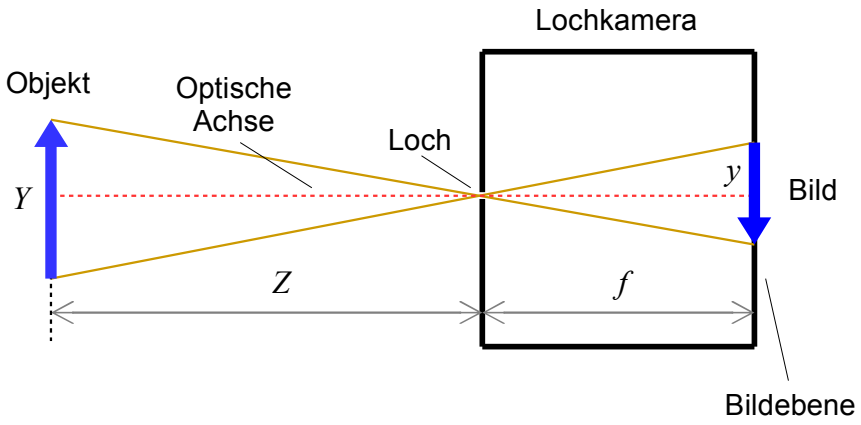


Abbildung 2.2.: Darstellung einer einfachen Lochkamera. Das Objekt wird durch das Eintrittsloch umgekehrt und spiegelverkehrt auf die Bildebene projiziert.

Die Größe des abgebildeten Objektes wird in dem Fall durch die Distanz Z von Objekt Y zum Loch (Abbildung 2.2) und die Strecke f vom Loch zur Abbildung y auf der Rückwand der Kamera, die Bildebene, bestimmt. Letztere entspricht in diesem Fall der Brennweite und wird daher als f bezeichnet. Mit dem Strahlensatz ergibt sich folgende Gleichung für

y:

$$-y = f \cdot \left(\frac{Y}{Z}\right) \tag{2.4}$$

Um die Mathematik einfacher zu gestalten, wird das traditionelle Lochkameramodell in der Computergrafik modifiziert. Die Bildebene wird imaginär entlang der optischen Achse um $2 \cdot f$ verschoben (siehe Abbildung 2.3). Das Minuszeichen in der Gleichung 2.4 fällt dadurch weg, die Maßstäbe bleiben erhalten. Als Folge steht damit die Projektion auf der Bildebene nicht mehr auf dem Kopf und ist nicht mehr spiegelverkehrt.

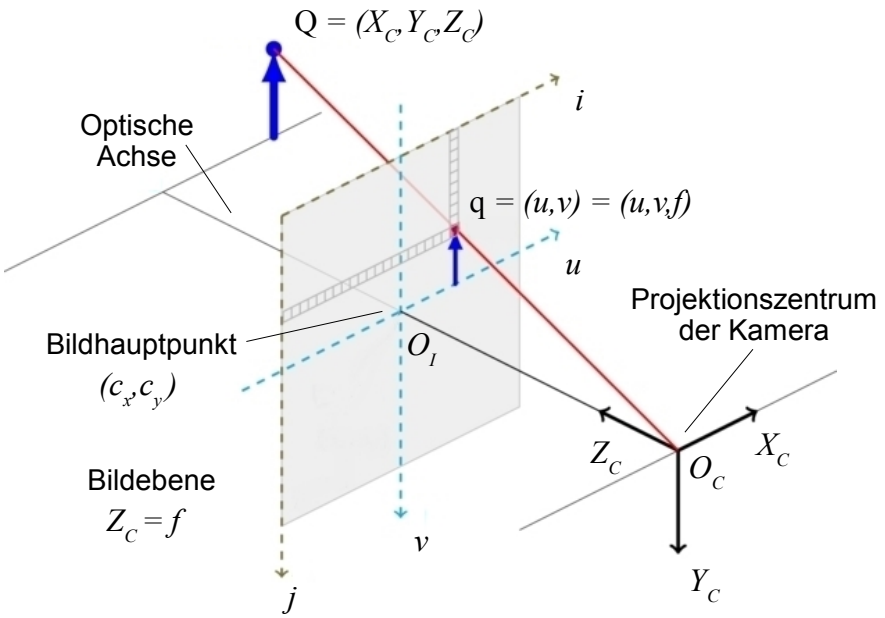


Abbildung 2.3.: Die Grafik zeigt die Abbildung eines Raumpunktes auf der Bildebene, das Modell basiert auf der Lochkamera. (Quelle: in Anlehnung an [27])

Im Weiteren werden nun 3D-Koordinaten betrachtet und gezeigt, wie ein beliebiger dreidimensionaler Punkt $\mathbf{Q}$ im Raum auf der Bildebene als zweidimensionaler Punkt $\mathbf{q}$ abgebildet wird.

Da sich die Bildebene genau in der Brennweite bei $Z_C = f$ befindet, kann der Punkt $\mathbf{q}$ in 3D-Koordinaten als $\mathbf{q} = \begin{bmatrix} u & v & f \end{bmatrix}^T$ beschrieben werden. Wieder mithilfe der Strahlensätze

ergibt sich:

$$\frac{u}{X_C} = \frac{v}{Y_C} = \frac{f}{Z_C}$$

$$u = f \cdot \left(\frac{X_C}{Z_C}\right) \tag{2.5}$$

$$v = f \cdot \left(\frac{Y_C}{Z_C}\right) \tag{2.6}$$

Da insbesondere bei günstigeren Kameras der Mittelpunkt des Sensors meist nicht exakt mit dem Zentrum der Linse übereinstimmt, entspricht der Bildhauptpunkt nicht dem Mittelpunkt der Aufnahme und den Gleichungen 2.5 und 2.6 muss ein Offset addiert werden: c_x, c_y

Auch die Pixel eines Sensors im Low-Cost-Bereich entsprechen nicht idealen Quadraten, sondern viel mehr Rechtecken. Es sind also verschiedene Brennweiten in x- sowie in y-Richtung zu beachten: f_x, f_y

$$u = f_x \cdot \left(\frac{X_C}{Z_C}\right) + c_x \tag{2.7}$$

$$v = f_y \cdot \left(\frac{Y_C}{Z_C}\right) + c_y \tag{2.8}$$

Um die Gleichungen 2.7 und 2.8 in Matrixschreibweise darzustellen, werden homogene Koordinaten verwendet und eine Dimension hinzugefügt. Aus dem Punkt $\mathbf{q} = \begin{bmatrix} u & v \end{bmatrix}^T$ wird $\tilde{\mathbf{q}} = \begin{bmatrix} u & v & 1 \end{bmatrix}^T$ und aus $\mathbf{Q} = \begin{bmatrix} X_C & Y_C & Z_C \end{bmatrix}^T$ wird $\tilde{\mathbf{Q}} = \begin{bmatrix} X_C & Y_C & Z_C & 1 \end{bmatrix}^T$. In Komponentenschreibweise ergibt sich:

$$\tilde{\mathbf{q}} = \mathbf{M} \cdot \tilde{\mathbf{Q}} \tag{2.9}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix}$$

Die Matrix $\mathbf{M}$ kann in eine 3x3-Matrix und eine Einheitsmatrix mit Nullvektor aufgespalten werden $\mathbf{M} = \mathbf{K} \cdot \begin{bmatrix} \mathbf{I} & 0 \end{bmatrix}$:

$$\tilde{\mathbf{q}} = \mathbf{K} \cdot \begin{bmatrix} \mathbf{I} & 0 \end{bmatrix} \cdot \tilde{\mathbf{Q}}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix}$$

Die entstandene 3x3-Matrix enthält die Brennweiten und Offsets des Bildhauptpunktes. Sie fasst damit die Parameter für die sogenannte innere Orientierung der Kamera zusammen und wird deshalb als Kameramatrix $\mathbf{K}$ (Gleichung 2.10) bezeichnet.

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

Bisher wurde angenommen, dass das Koordinatensystem der Kamera und des betrachteten Objektes miteinander übereinstimmen, dies ist in der Regel jedoch nicht der Fall. Die Koordinaten des Objektes werden Weltkoordinaten genannt und müssen in die Kamerakoordinaten überführt werden. Die Transformation besteht aus einer Kombination von Rotation $\mathbf{R}$ und Translation $\mathbf{T}$ und beschreibt die Orientierung der Kamera im Raum:

$$\mathbf{Q} = \mathbf{R} \cdot \mathbf{Q_W} + \mathbf{T}$$

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \cdot \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

In Matrixschreibweise wird die Transformation als $\mathbf{Q} = \mathbf{Q_W} \cdot \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix}$ ausgedrückt. Es ist eine kombinierte Matrix aus Rotationsmatrix und dem Translationsvektor entstanden, die einen Punkt von Weltkoordinaten in Kamerakoordinaten überführt. Bezogen auf die Glei-

chung 2.9 ergibt sich für die Matrix $\mathbf{M}'$:

$$\mathbf{M}' = \mathbf{K} \cdot \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix}$$

In ausführlicher Komponentenschreibweise werden die innere Orientierung, also die Kameramatrix, und die äußere Orientierung der Kamera im Raum wie folgt zusammengefasst:

$$\tilde{\mathbf{q}} = \mathbf{M}' \cdot \tilde{\mathbf{Q}}$$

$$\tilde{\mathbf{q}} = \mathbf{K} \cdot \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix} \cdot \tilde{\mathbf{Q}}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix}$$

Ohne Beschränkung der Allgemeinheit kann nun die Bildebene festgelegt werden, indem die Koordinate Z_C gleich Null gesetzt wird [4]. Damit fällt in der Rotatonsmatrix die dritte Spalte weg.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_C \\ Y_C \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X_C \\ Y_C \\ 1 \end{bmatrix}$$

Die Kameramatrix ergibt zusammen mit der verbliebenen kombinierten Matrix aus Rotation

und Translation die sogenannte Homographie $\mathbf{H}$, eine 3x3-Matrix:

$$\mathbf{H} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{2.11}$$

Der Koeffizient h_{33} ist lediglich ein Skalierungsfaktor und kann auf eins gesetzt werden. Damit ergeben sich acht Unbekannte, die mit mindestens vier korrespondierenden Punkten gelöst werden können. Bei einer Homographie handelt es sich um die Transformationsmatrix einer perspektivischen Projektion (siehe Unterabschnitt 2.1.2).

Abschließend kann die gesamte Transformation, um einen Punkt $\tilde{\mathbf{Q}}'$ im Raum als Punkt $\tilde{\mathbf{q}}$ auf einer Bildebene abzubilden, somit wie folgt zusammengefasst werden:

$$\tilde{\mathbf{q}} = \mathbf{H} \cdot \tilde{\mathbf{Q}}'$$

3. Konzept

In diesem Kapitel werden die Anforderungen an das Projekt definiert und der experimentelle Aufbau vorgestellt. Dabei wird auf die wichtigste Hardware genauer eingegangen.

3.1. Anforderungen

In dieser Arbeit soll eine Lösung entwickelt werden, die in der Lage ist, Objekte in der unmittelbaren Umgebung wahrzunehmen, um eine mögliche Kollision zu verhindern. Dies soll sowohl visuell durch einen Bediener (Panoramabild) als auch computergestützt durch maschinelles Sehen erfolgen (Stereoskopie).

Für die Erkennung von Hindernissen ist vor allem das Sichtfeld der Kamera entscheidend, dieses soll größtmöglich sein. Zum einen bezieht sich dies auf die menschliche Beobachtung, um alle Richtungen im Blick zu haben und so eine sichere Navigation zu ermöglichen. Ebenso wichtig ist das Sichtfeld aber auch für eine autonome Lösung. Für Stereoskopie müssen die Kameras kalibriert werden, wodurch ein Teil des Bildes und damit des Sichtfeldes verloren geht. In der Bildverarbeitung sind zudem eine hohe Auflösung und geringes Bildrauschen von Vorteil.

Da als Anwendungsgebiet mobile Roboter vorgesehen sind und insbesondere der Einsatz an einem UAV vorstellbar ist, sind eine kleine und kompakte Bauweise sowie ein niedriges Gewicht erforderlich. Vor allem in diesem Kontext sind eine schnelle Erkennung der Hindernisse sowie allgemein eine flüssige Darstellung der Ergebnisse wünschenswert.

Für einen ersten experimentellen Aufbau des Systems standen aus dem Bereich Rapid Prototyping sowohl ein 3D-Drucker als auch ein Laser Cutter zur Verfügung. Daher soll der

überlegte Prototyp in Hinsicht auf Geometrie, Größe und Material mit den vorhandenen Mitteln realisierbar sein. Außerdem ist der Anspruch klar auf Low-Cost ausgerichtet.

3.2. Festlegung auf Konstruktion

In einem vorherigen Projekt wurde vom Autor dieser Arbeit bereits eine Panoramakamera entwickelt, konstruiert und durch Rapid Prototyping gefertigt (siehe Abbildung A.1). Die Programmierung, um Bilder zusammenzuführen, blieb jedoch aus. Vier Kameras sind in diesem Fall horizontal in einem Winkel von 90° zueinander verbaut, jede Kamera ist zusätzlich vertikal um 45° geneigt. Es wurden die gleichen Raspberry Pi Kameras wie in dieser Arbeit verwendet. Das Sichtfeld ist dementsprechend horizontal ca. 125° und vertikal ca. 90° (siehe Abbildung A.2).

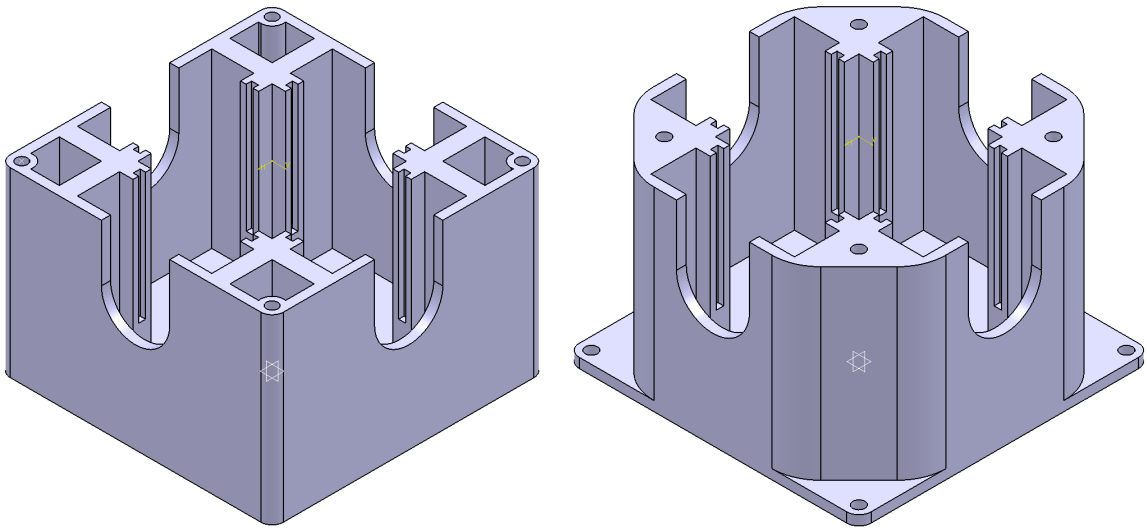
(a) (b)

Abbildung 3.1.: Konstruktion eines im Vorfeld entwickelten Aufbaus

Durch verschiedene Versuche gelang es allerdings nicht, unter diesen Bedingungen die Bilder der Kameras zusammenzuführen. Dies ist im Anhang A detailliert dargestellt. Aus diesem Grund wurden die Kameras in der in dieser Arbeit vorgestellten Konstruktion ohne zusätzlichen Neigungswinkel verbaut.

3.3. Experimenteller Aufbau

Nachdem sich für eine Anordnung der Kameras in einem 90°-Winkel ohne zusätzliche Neigung entschieden wurde, erfolgte in CATIA V5 [9] zunächst die Konstruktion eines Gehäuses für die Panoramaerstellung in Kapitel 4 (siehe Anhang 3.2a). Anschließend wurde es per 3D-Druck mit einem MakerBot Replicator 2x [19] aus ABS gefertigt. Um Stereoskopie zu ermöglichen, kam im weiteren Verlauf ein weiteres Gehäuse für vier Kameras hinzu (Anhang 3.2b).

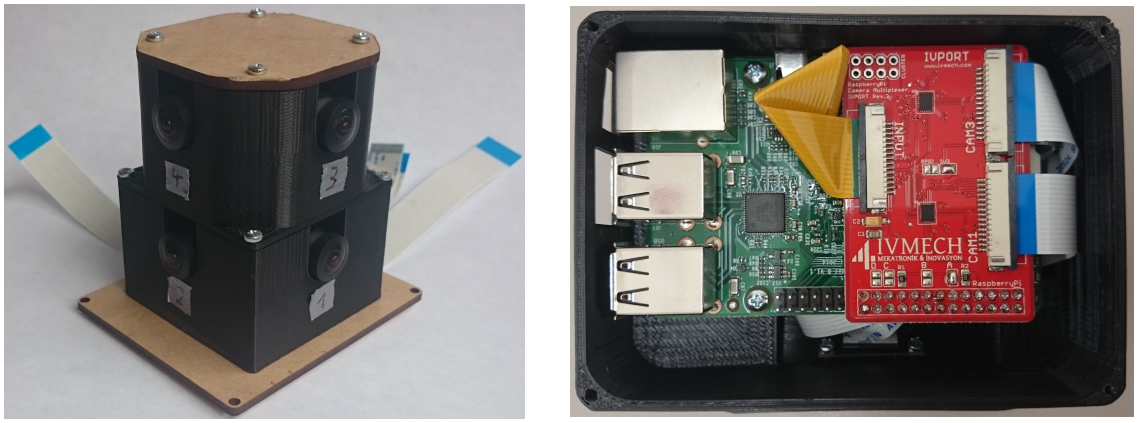


(a) Unteres Gehäuse, das zuerst für die Erstellung des Panoramas dient.

(b) Durch das obere Gehäuse ergibt sich zu jeder Seite eine Stereokamera.

Abbildung 3.2.: Gehäuse für den Einbau von je vier Kameras, die miteinder verschraubt werden können, sodass insgesamt acht Kameras Platz finden.

Beide Gehäuse können mit Schrauben verbunden werden, sodass je immer zwei Kameras direkt übereinander sind (siehe Abbildung 3.3a). Die Deckel wurden per Laser Cutting hergestellt. Als Material kam dabei Kraftplex mit einer Stärke von 3,0 mm zum Einsatz, welches aus 100 % Weichholzfasern besteht und sich dadurch auszeichnet, dass es sowohl stabil und leicht als auch einfach zu verarbeiten ist.



(a) Per 3D-Druck und Laserschneiden gefertigtes Gehäuse

(b) Systemaufbau aus Raspberry Pi (grüne Platine) und Multiplexer-Modul (rote Platine)

Abbildung 3.3.: Hier sind

Die im Gehäuse (Abbildung 3.3a) eingebauten Kameras schicken über Flachbandkabel ihre Bilder an einen Raspberry Pi. Es handelt sich dabei um einen Pi 2 Model B mit einem ARM Cortex-A7 Prozessor (900 MHz) in ARMv7 32-Bit Architektur. Der Grafikprozessor vom Typ Broadcom Dual Core VideoCore IV taktet mit 250 MHz und der Arbeitsspeicher ist 1024 MB groß. Da nur eine Kamera-Schnittstelle (CSI) vorhanden ist, kommt ein Multiplexer zum Einsatz (siehe Abschnitt 3.3.2), der über die GPIO-Pins des Raspberry Pi angebunden ist.

3.3.1. Kamera-Modul

Zum Einsatz kommen Raspberry [31] Pi V1 Kameras mit einem fünf Megapixel Sensor (OmniVision OV5647 [25]). Bilder können mit einer maximalen Auflösung von 2592x1944 Pixeln und Videos mit 1920x1080 (1080p) bei 30 fps bzw. 1280x720 (720p) bei 60 fps aufgenommen werden. Die Kamera verfügt über ein Fixfokus-Objektiv mit einer Brennweite von 3,6 mm und wird mit einem 15-poligen Flachbandkabel über die CSI-Schnittstelle des Raspberry Pi angebunden. Auf die eigentliche Kamera wurde bereits im Auslieferungszustand ein frei justierbares Fisheye-Objektiv gesetzt, wodurch sich ein größerer Sichtwinkel, aber auch eine neue, nicht fixierbare Brennweite ergibt. Es kann per Hand durch Drehen

justiert werden. Um für alle Kameras die gleichen Bedingungen zu schaffen, wurden sie auf ein Schachbrettmuster in einem Meter Entfernung scharf eingestellt.

Vom Hersteller wird die Kamera samt Objektiv mit einem Sichtfeld (Field of View, FOV, siehe Abbildung 3.4) von 160° beworben. Diese Aussage kann missverständlich interpretiert werden, denn gemeint ist damit lediglich das diagonale Sichtfeld (FOV_D), horizontal (FOV_H) ergeben sich 128° und vertikal (FOV_V) 96°. In der Realität sind diese Angaben geringer, es wurde ein Sichtfeld von circa 125° in der Horizontalen und 90° in der Vertikalen gemessen.

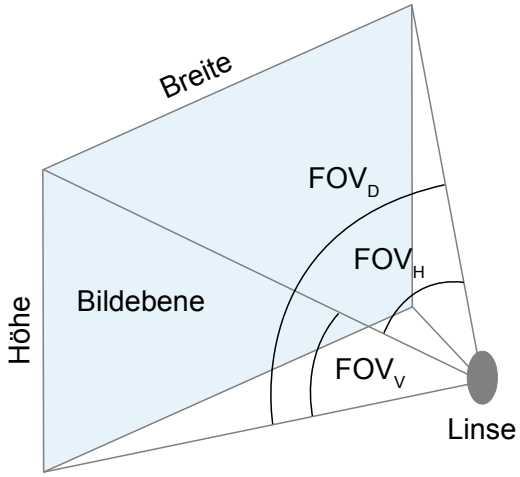
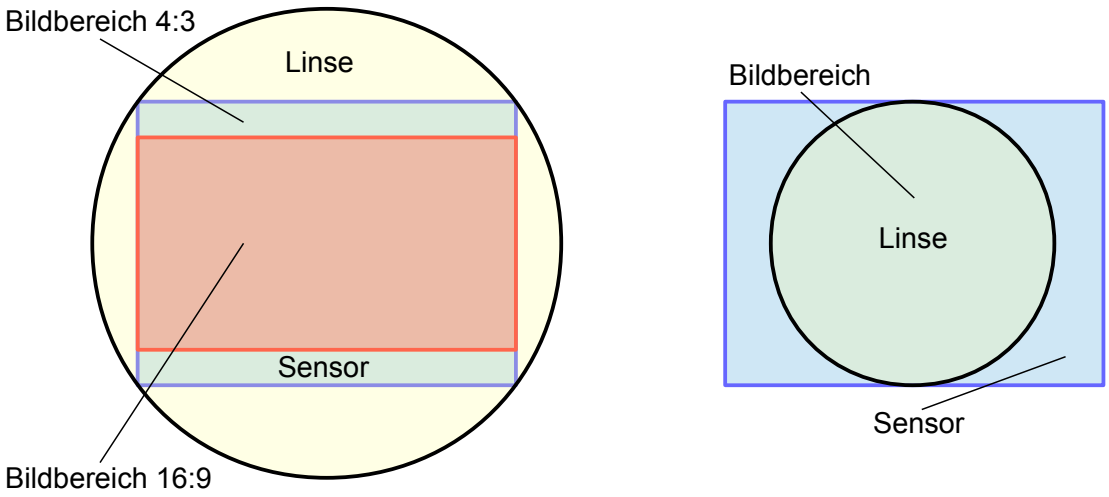


Abbildung 3.4.: Darstellung des Sichtfeldes (Field of View) einer Kamera in der Diagonalen (FOV_D), Horizontalen (FOV_H) und Vertikalen (FOV_V)



(a) Full-Frame Fisheye (b) Circular Fisheye

Abbildung 3.5.: In (a) ist ein Full-frame Fisheye zu sehen, das den kompletten Bildbereich vom Sensor abdeckt (blau). In Rot ist die Ausgabe im Breitbild-Format dargestellt, wobei das Sichtfeld vertikal nicht voll ausgenutzt wird. (b) zeigt zum Vergleich ein Circular Fisheye. Die Linse ist kleiner als der Sensor, das Ausgabebild entsprechend kreisförmig.

Bei der Kamera handelt es sich um ein Full-Frame Fisheye (Abbildung 3.5a), die Linse ist größer als der Kamerasensor. Es wird also im Gegensatz zum Circular Fisheye kein Bereich des Sensors verschwendet. In der Standardeinstellung gibt die Kamera Bilder in einer Auflösung von 1920 x 1080 Pixeln aus, das Seitenverhältnis ist somit $16 : 9$. Der Sensor ist jedoch für eine $4 : 3$-Auflösung ausgelegt und somit wird in der Einstellung das vertikale Sichtfeld nicht voll ausgenutzt (Abbildung 3.5b), es ist um 25 Prozent kleiner.

3.3.2. Multiplexer-Modul

In Abbildung 3.3b ist der Raspberry Pi samt aufgestecktem Multiplexer-Modul zu sehen. Es handelt sich dabei um den IVPort Raspberry Pi Camera Module Multiplexer in der Revision 3 vom Hersteller IVMECH [15]. Das Modul wird über ein 15-poliges Flachbandkabel an die CSI-Schnittstelle des Raspberry Pi angeschlossen, vier CSI-Anschlüsse für folglich vier Kameras befinden sich auf der Platine selbst. Werden mehr Schnittstellen benötigt,

kann einfach ein weiteres Modul auf die Pins gesteckt und wiederum per CSI mit dem ersten Modul verbunden werden. Es ist möglich, insgesamt vier dieser Multiplexer, also 16 Kameras einzusetzen. Die Schaltung erfolgt über die GPIO-Pins des Raspberry Pi; bei vier gewünschten Schnittstellen werden drei Pins belegt, fünf Pins für acht und neun Pins für 16. Die Gatterlaufzeit wird vom Hersteller mit 0,5 ns und die Schaltzeit zwischen den Kanälen mit 50 ns angegeben.

3.3.3. Workstation und Software

Die Python-Skripte wurden in Windows 7 64-Bit programmiert. Dabei wurde python(x,y) [30] verwendet, in dem die OpenCV-Bibliothek in der Version 2.4.11-7 integriert ist. Für die Programmierung in C++ in Verbindung mit der Point Cloud Library 1.8 wurde dagegen auf Linux-Distribution Kubuntu 16.04 64-Bit gearbeitet. Als Workstation diente ein PC mit einem Intel Core i3-4170 Prozessor bei einer Taktung von 3,7 GHz, 8 GB Arbeitsspeicher und 250 GB SSD. Aus Performance-Gründen wurde hauptsächlich auf der Workstation gearbeitet und der Raspberry Pi lediglich zur Bildaufnahme verwendet.

4. Zusammenführen einzelner Bilder zu einem Panoramabild

Dieses Kapitel befasst sich mit der Erzeugung eines Panoramabildes. Es wird das Zusammenführen (engl. Stitching) der einzelnen Kamerabilder zu einem Gesamtbild beschrieben. Zuerst wird der entwickelte Algorithmus in Abschnitt 4.1 dargelegt, wie die einzelnen Bilder per Environment Mapping transformiert werden müssen, um später miteinander zusammengefügt werden zu können. Anschließend in Abschnitt 4.2 wird gezeigt, wie dieser Prozess implementiert wurde. Im letzten Abschnitt 4.3 werden die praktischen Ergebnisse des geschriebenen Programms bewertet.

4.1. Environment Mapping

Environment Mapping ist eine Texture Mapping-Methode, um Bilder in eine Sphäre zu projizieren. Dabei wird die beobachtete Umgebung (engl. Environment) auf dem Inneren einer hohlen Kugel, der Sphäre, abgebildet. Wenn man nun vom Zentrum der Kugel aus die Innenfläche betrachtet, sieht man abhängig von der Richtung die projizierte Umgebung [32]. Um wieder ein zweidimensionales Bild zu erhalten, wird anschließend die Innenfläche der Kugel auf eine Textur, die Bildebene, projiziert.

In OpenCV ist der Ursprung des Koordinatensystems eines jeden Bildes in der oberen linken Ecke festgelegt [3] (vergleiche auch Abbildung 2.3 im Grundlagenkapitel). Von der Pixelkoordinate (0,0) erstreckt sich j horizontal bis zum maximalen Pixelwert, der Bildbreite (siehe Abbildung 4.1). Folglich ist i vertikal vom Ursprung (0,0) bis zum Maximalwert die Höhe des Bildes.

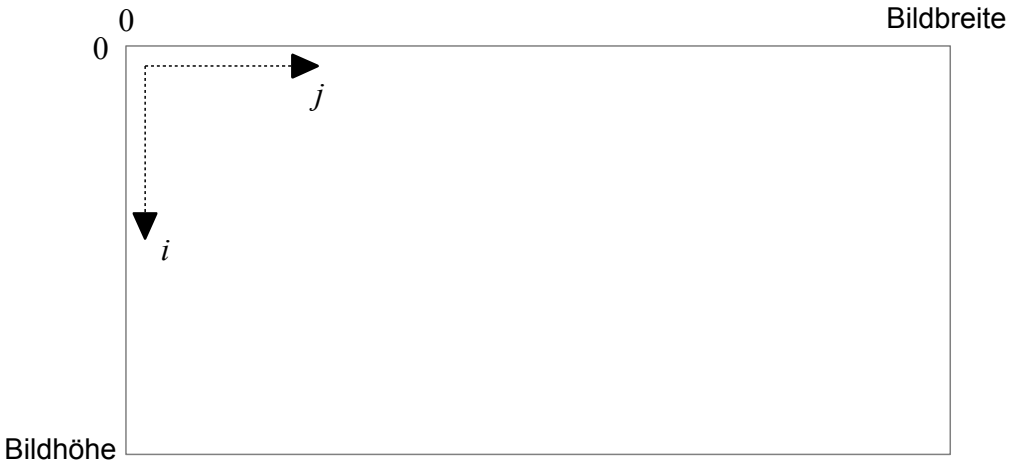


Abbildung 4.1.: Die Abbildung zeigt, wie die Pixelkoordinaten in OpenCV definiert sind. Die Variable j bezeichnet die Koordinaten horizontal und i vertikal.

Um ein Bild von der Fisheye-Kamera in einer Sphäre abzubilden, müssen die Pixelkoordinaten zuerst in kartesische Koordinaten und danach in sphärische Koordinaten (oft auch als Kugelkoordinaten bezeichnet [28]) transformiert werden. Für diese Koordinaten gibt es eine Reihe von Konventionen, die sich in der Definition und Bezeichnung der Winkel unterscheiden. Für die Projektion in die hohle Kugel wird sich an die Bezeichnungen von sphärischen Koordinaten in der Geographie gehalten [20]. Neben den mathematischen Zusammenhängen wird zum besseren Verständnis am Beispiel der Weltkugel die Transformation grafisch gezeigt.

Die geographischen Längengrade werden von -180° bis 180° aufgeteilt und als λ bezeichnet, während die Breitengrade von -90° bis 90° definiert sind und mit φ angegeben werden. Gearbeitet wird in OpenCV jedoch nicht in Grad, sondern in Bogenmaß, was umgerechnet für die Winkel bedeutet:

$$\text{Längengrade: } -\pi \leq \lambda \leq \pi$$
$$\text{Breitengrade: } -\pi/2 \leq \varphi \leq \pi/2$$

Aus diesen Bedingungen ergeben sich für die Winkel λ und φ folgende Funktionen in

Abhängigkeit der Pixelkoordinaten i bzw. j:

$$\lambda = 2 \cdot \pi \cdot (\frac{j}{\text{Bildbreite}} - 0.5) \tag{4.1}$$

$$\varphi = \pi \cdot (\frac{i}{\text{Bildhöhe}} - 0.5) \tag{4.2}$$

Gleichungen 4.1 und 4.2 auf die Pixelkoordinaten i und j angewandt, unter Beachtung von Bildhöhe sowie Bildbreite (siehe 4.1), führt zu den Koordinaten in Abbildung 4.2, die dargestellte Karte wird auch als Rektangularprojektion bezeichnet.

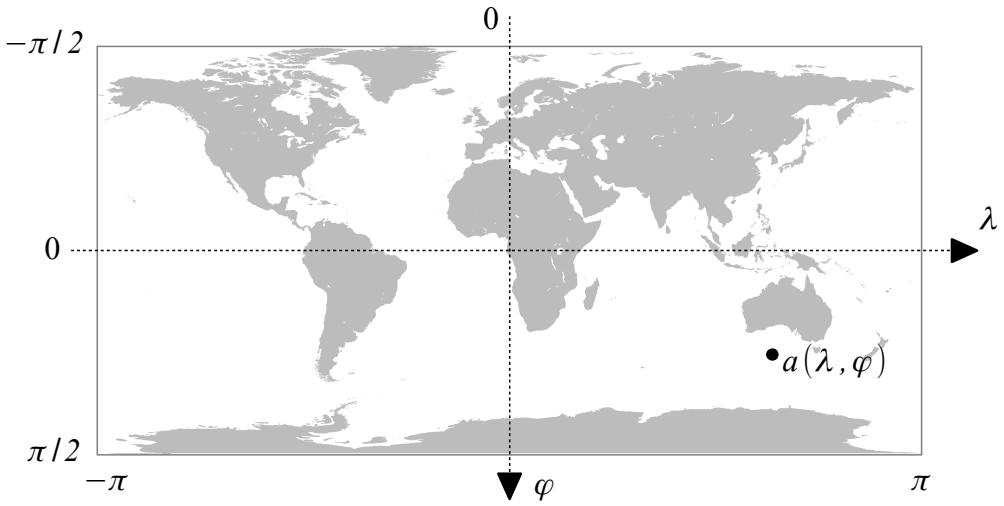


Abbildung 4.2.: Darstellung der geographischen Längen- und Breitengrade. Beispielhaft ist die Welt als Rektangularprojektion, auch Plattkarte genannt, zu sehen (Quelle der Hintergrundgrafik: [8]).

Die geographische Länge λ und Breite φ können nun als kartesische Koordinaten für die Umrechnung in sphärische Koordinaten verwendet werden. Dabei soll ein Zusammenhang des Koordinatensystems in Abbildung 4.2 mit dem in Abbildung 4.3 hergestellt werden. Mathematisch ist diese Transformation in Bourke [2] beschrieben. Im Unterschied zur dortigen Darstellung wurde allerdings das Koordinatensystem gedreht. Die Z-Achse zeigt nun in die Ebene hinein und die XY-Ebene wurde so gedreht, dass X horizontal und Y vertikal verläuft. Damit wurden die Achsen an die in OpenCV übliche Konvention in Abbildung

2.3 angepasst. Um Missverständnisse zu vermeiden, wurden in der gesamten Arbeit die Koordinaten auf diese Weise ausgerichtet,

Die kartesischen Koordinaten des Punktes a, bestehend aus φ und λ (siehe Abbildung 4.2), stehen zu dem Punkt $A(X,Y,Z)$ auf der Kugeloberfläche in sphärischen Koordinaten (Abbildung 4.3) in folgender Beziehung:

$$X = \rho \cdot \cos(\varphi) \cdot \sin(\lambda)$$
$$Y = \rho \cdot \sin(\varphi)$$
$$Z = \rho \cdot \cos(\varphi) \cdot \cos(\lambda)$$

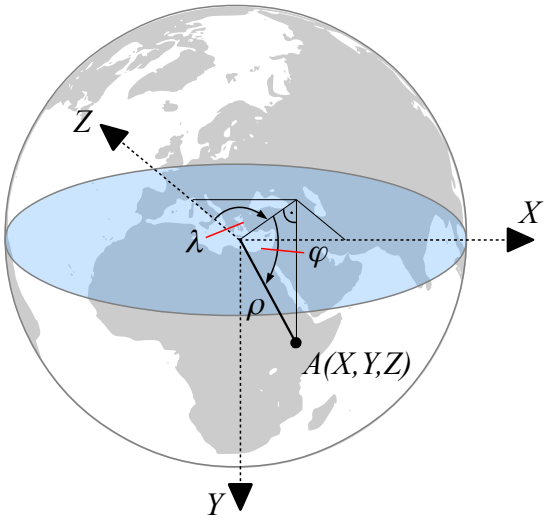


Abbildung 4.3.: Dargestellt sind sphärische Koordinaten wie sie in der Geographie üblicherweise gelten. Die Weltkarte aus 4.2 ist auf der Innenfläche der Sphäre projiziert. Der Punkt A stellt einen Punkt auf der Kugeloberfläche, der Sphäre, in Abhängigkeit von ρ, λ und φ dar (Quelle der Hintergrundgrafik: [12]).

Zur Vereinfachung wird die Sphäre als Einheitssphäre betrachtet [1], sodass sich mit $\rho = 1$

ergibt:

$$X = \cos(\varphi) \cdot \sin(\lambda)$$
$$Y = \sin(\varphi)$$
$$Z = \cos(\varphi) \cdot \cos(\lambda)$$

Die 2D-Koordinaten des Ursprungsbildes sind damit in 3D-Koordinaten umgewandelt, das Bild ist auf der Innenfläche eine hohlen Kugel abgebildet. Die Innenfläche gilt es nun wieder in Abhängigkeit vom Sichtfeld (Field of View) der Fisheye-Kamera auf eine 2D-Bildebene zu projizieren.

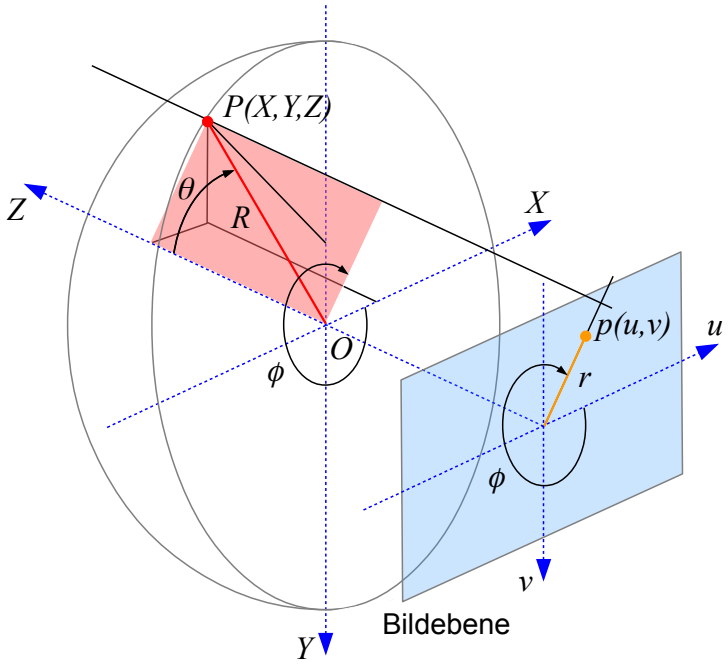


Abbildung 4.4.: Die Abbildung stellt die Rückprojektion der Innenfläche der hohlen Kugel in sphärischen Koordinaten auf eine 2D-Bildebene in UV-Koordinaten dar. Es gelten die für eine Fisheye-Kamera üblichen Konventionen der Winkel.

Für diese Projektion müssen die Winkel neu definiert werden. Es wurde sich nach der Konvention für Fisheye-Koordinaten gerichtet, wie sie in Corke [7] festgelegt ist. Dabei ist

die Z-Achse als die optische Achse der Kamera festgelegt. R ist die Strecke vom Ursprung der Sphäre O zum Raumpunkt P und der Winkel θ befindet zwischen der Z-Achse und R. Der Winkel ϕ ist sowohl in der Sphäre als auch in der projizierten UV-Bildebene vorzufinden und dreht sich um die Z-Achse (siehe Abbildung 4.4).

Wie aus der Abbildung 4.4 abgelesen werden kann, sind für das Texture Mapping von einem dreidimensionalen Punkt P in Fisheye-Koordinaten zu einem zweidimensionalen Punkt p in der Bildebene folgende UV-Koordinaten verantwortlich [7]:

$$u = r \cdot \cos(\phi) \tag{4.3}$$

$$v = r \cdot \sin(\phi) \tag{4.4}$$

Dabei ist r vom Winkel θ und dem Sichtfeld der Kamera (in der Gleichung als FOV bezeichnet) abhängig [2]:

$$r = \frac{2 \cdot \theta}{\text{FOV}}$$

Aus der Abbildung 4.4 können für θ, ϕ und R folgende Zusammenhänge hergeleitet werden:

$$\theta = \arccos(\frac{Z}{R})$$
$$\phi = \arctan(\frac{Y}{X})$$
$$R = \sqrt{X^2 + Y^2 + Z^2}$$

Damit sind alle notwendigen Variablen aus den Gleichungen 4.3 und 4.4, um den Punkt P mit UV-Koordinaten abzubilden.

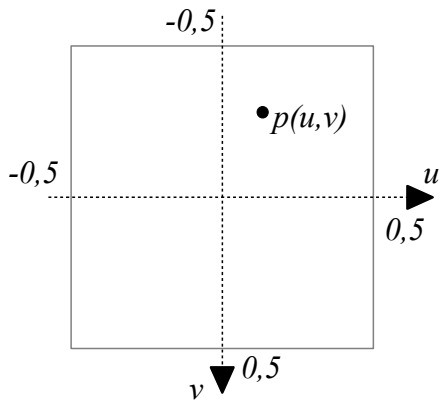


Abbildung 4.5.: Aus der Sphäre auf 2D-Bildebene projizierter Punkt p in UV-Koordinaten.

Bei der Projektion der Kugelinnenfläche aus Richtung der Z-Achse auf die 2D-Bildebene entstehen sowohl für u als auch v Koordinaten im Wertebereich von -0,5 bis 0,5 (siehe Abbildung 4.5).

OpenCV und Texture Mapping im Allgemeinen benötigen jedoch Koordinaten von 0,0 bis 1,0 [22], die Gleichungen 4.3 und 4.4 müssen also noch modifiziert werden (siehe Abbildung 4.5):

$$u' = r \cdot \cos(\phi) + 0.5$$
$$v' = r \cdot \sin(\phi) + 0.5$$

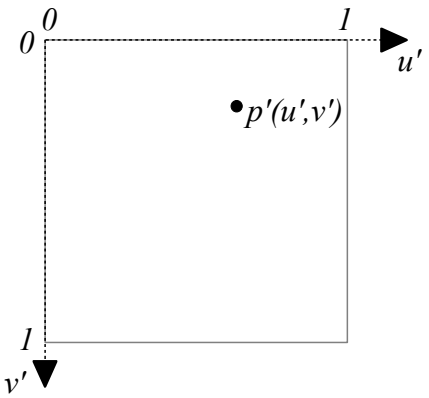


Abbildung 4.6.: Punkt p' in für Texture Mapping üblichen UV-Koordinaten.

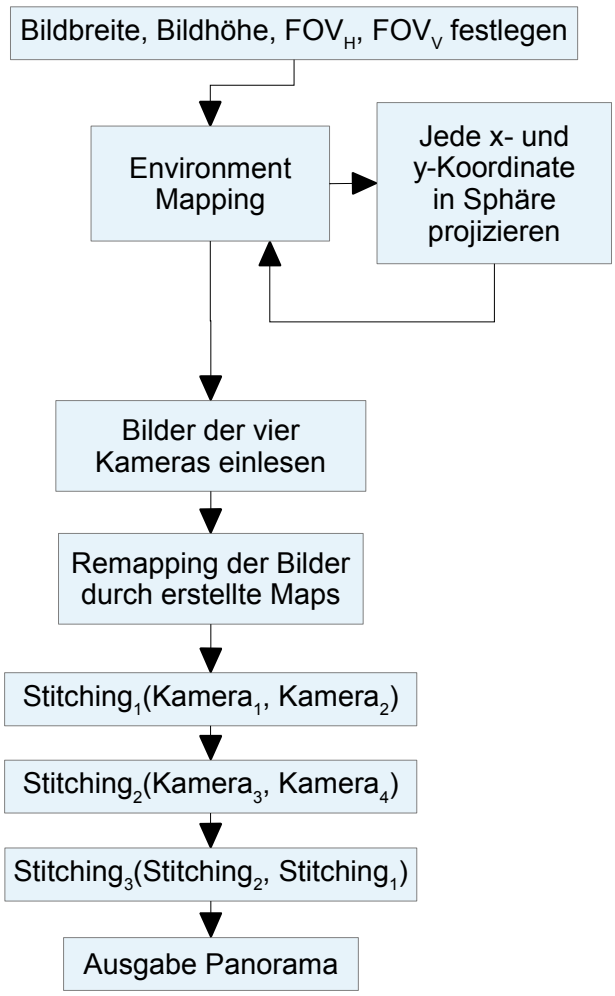
4.2. Programmierung

Für die Umsetzung des Vorhabens, die Bilder von vier Kameras im 90°-Winkel zueinander zusammenzuführen, wurde als Programmiersprache Python in Verbindung mit OpenCV [3] gewählt. OpenCV ist eine weit verbreitete und schnelle Bibliothek, die Algorithmen aus der Bildverarbeitung frei zur Verfügung stellt. Dabei stand zur Auswahl direkt auf dem Raspberry Pi zu arbeiten und Bilder in Echtzeit zu erhalten oder auf einer externen leistungsfähigen Workstation zu programmieren. Im Hinblick auf die geringe Rechenleistung des Raspberry Pi wurde hauptsächlich auf der Workstation gearbeitet, um schneller Ergebnisse zu erhalten. In dem Fall war der Raspberry Pi lediglich für die Aufnahme der Bilder zuständig.

Zu Beginn des Programms werden Breite und Höhe des gewünschtes Ausgangsbildes festgelegt. Zusammen mit den gemessenen horizontalen und vertikalen Sichtfeldern der Kameras werden die benötigten sogenannten Maps für das Texture Mapping erstellt.

Die Remap-Funktion von OpenCV ermöglicht es, Bilder zu beliebigen neuen Bildern zu transformieren. Meistens wird sie zur Korrektur von verzeichneten Aufnahmen verwendet. Verzeichnung beschreibt die nicht maßstabsgetreue Wiedergabe einer Szene und wird in Kapitel 5.1 erklärt.

Für die Funktion *cv2.remap()* werden jeweils eine Map in horizontaler und eine in vertikaler Bildrichtung benötigt. Die Maps enthalten Informationen über die Zielkoordinaten eines jeden Pixels aus dem Ursprungsbild, der Algorithmus und damit die Funktion *projectToSphere()* ist in Abschnitt 4.1 beschrieben. Allerdings müssen die Maps nur einmal am Anfang erstellt werden und können fortan zum Remapping der einzelnen Kamerabilder verwendet.




```
for y in xrange(0, height):
  for x in xrange(0, width):

    U, V = projectToSphere(x, y,
       width, height, FOVh, FOVv)

      if (int(U) < 0 or
             int(U) > width-1
             or int(V) < 0 or
             int(V) > height-1):
             continue

      mapU[y, x] = int(U)
      mapV[y, x] = int(V)
```

```
cv2.imread
```

```
remapped = cv2.remap(image,
  mapx, mapy,
  interpolation=cv2.INTER_LINEAR)
```

```
cv2.imshow
```

Abbildung 4.7.: Das Diagramm zeigt den Ablauf des in Python mit OpenCV geschriebenen Programms. Rechts sind zu den einzelnen Schritten die wichtigsten Funktionen und Auszüge aus dem Quelltext dargestellt. Die Stitching-Funktion ist in Abbildung 4.8 ausführlicher erklärt.

Mit der Stitching-Funktion (siehe Abbildung 4.8) werden nun die Bilder von zwei nebeneinander befindlichen Kameras zusammengeführt, danach die Bilder von den zwei übrig gebliebenen Kameras. Anschließend werden die beiden Ergebnisse wieder der Stitching-Funktion übergeben und ein sphärisches Panoramabild erzeugt.

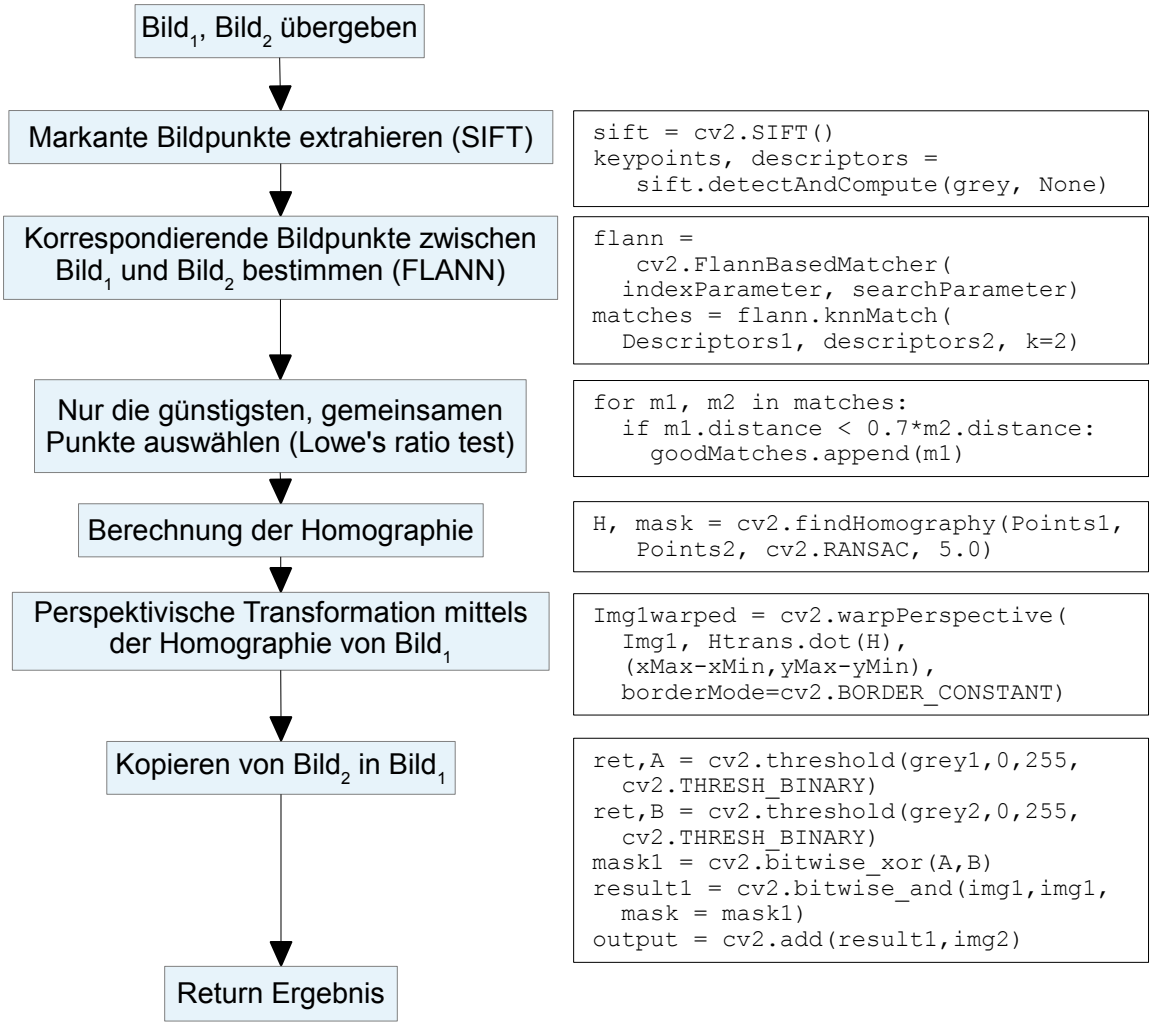


Abbildung 4.8.: Zu sehen ist der detaillierte Ablauf der Stitching-Funktion in Python/OpenCV mit entscheidenden Auszügen aus dem Programm (der Quellcode zur perspektivischen Transformation basiert auf einem Beispiel aus Joshi [17])

Beim Stitching werden mithilfe des SIFT-Algorithmus (Scale-invariant Feature Transform [18]) von beiden eingelesen Bildern Keypoints und Descriptors also markante und reproduzierbare Bildpunkte erkannt und beschrieben.

Mit dem FLANN-Algorithmus (Fast Approximate Nearest Neighbor Search [24]) werden als nächstes die gemeinsamen, korrespondierenden Merkmale zwischen den beiden Bildern

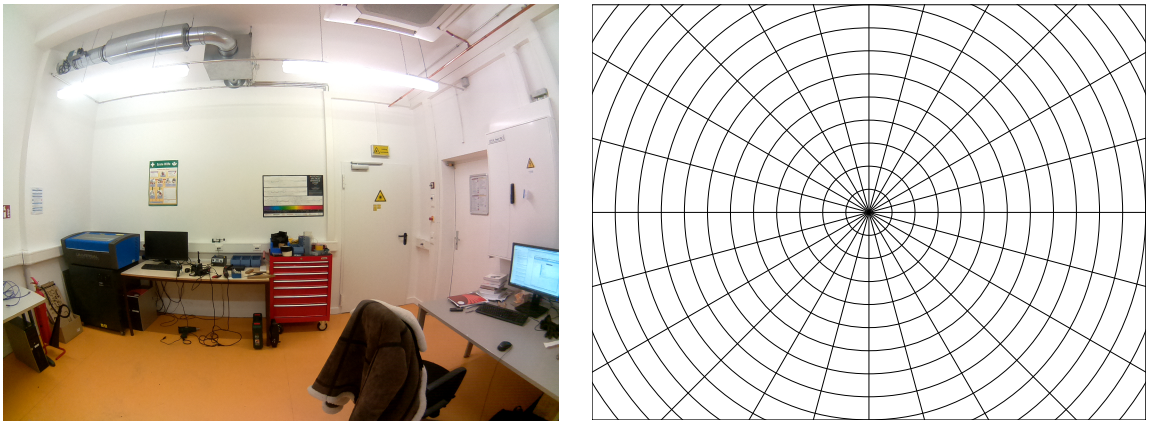
bestimmt. Sind ausreichend Treffer vorhanden, werden mittels Lowe's Ratio Test [18] die optimalen Paare ausgewählt.

Mit den korrespondieren Punkten kann daraufhin ein Gleichungssystem aufgestellt werden, um die sogenannte Homographie zu berechnen. Diese beschreibt eine lineare Bildtransformation in homogenen Koordinaten und wird mit dem RANSAC-Algorithmus (Random Sample Consensus [11]) gelöst. Mit RANSAC werden allgemein Modelle zu Messdaten geschätzt, die viele Ausreißer enthalten. Mit der ermittelten Homographie wird das erste Bild perspektivisch transformiert.

Zum Schluss werden beide Bilder zu einem Gesamtbild zusammengefügt. Dabei werden von beiden Aufnahmen binäre Masken erzeugt und anschließend die Maske vom ersten Bild mit der vom Zweiten XOR-verknüpft. Das Binärergebnis aus dieser Operation wird mit dem erstem Bild UND-verknüpft, sodass nur noch der Teil vom Bild übrig bleibt, der nicht vom zweiten Bild verdeckt werden würde. Zuletzt wird noch das zweite Bild in das Ergebnis hineinkopiert.

4.3. Bewertung und Zusammenfassung

Zur Auswertung wurde der Prototyp in die Mitte eines hell beleuchteten Raumes auf ein Stativ gestellt. Das System wurde dabei über das Netzwerk via TightVNC [36] gesteuert und mit jeder Kamera Bilder geschossen. Um möglichst ähnliche Aufnahmen zu erhalten, wurde ein ISO-Wert von 320 und eine Verschlusszeit von 1/25 s für jede Kamera festgelegt. Außerdem wurde der für künstliche Beleuchtung optimierte Modus „tungsten“ für den automatischen Weißabgleich eingestellt.



(a) Verzeichnetes Originalbild einer Raspberry Pi V1 Kamera mit Fisheye-Obektiv

(b) Schematische Darstellung einer Aufnahme von einer Full-frame Fisheye Kamera

Abbildung 4.9.: Aufnahme einer realen Umgebung im Vergleich mit einer schematischen Darstellung der Fisheye-Kamera

In der Abbildung 4.9 ist links (4.9a) das von der Kamera aufgenommene verzeichnete Bild eines Raumes zu sehen und rechts (4.9b) die schematische Darstellung des Sichtwinkels der Fisheye-Kamera. Jeder Ring entspricht 10° des Sichtwinkels, wodurch sich für das horizontale Field of View 120° und für das Vertikale 90° ergeben. Es handelt sich um ein Full-Frame Fisheye (siehe Abbildung 3.5a).

Nachfolgend in Abbildung 4.10 ist für ein Einzelbild einer Kamera das Texture Mapping dargestellt, also die Projektion in eine Sphäre. Links (4.10a) ist die transformierte Aufnahme der Kamera und rechts (4.10b) das transformierte Bild des schematischen Field of View zu sehen. Mit der *cv2.remap*-Funktion erhält jede Bildkoordinate in x- und y-Richtung des Ursprungsbildes (4.9a) eine neue Zielkoordinate im Ausgangsbild. Genaue Informationen über die Transformationen sind in den Maps enthalten, die mittels des in Abschnitt 4.1 beschriebenen Algorithmus berechnet wurden.

(a) In Sphäre überführtes Bild der Raspberry Pi Kamera

(b) Schematisches Remapping des Fisheye-Gitters in die Sphäre

Abbildung 4.10.: Die Remap-Funktion ordnet jedem Bildpunkt aus den Originalbildern neue Zielkoordinaten zu.

Wenn die Originalbilder der Kameras transformiert wurden, gilt es gemeinsame Punkte zwischen den Aufnahmen der verschiedenen Kameras zu finden, um die Bilder miteinander zu überlagern. Wie in Abschnitt 4.2 beschrieben, werden zuerst durch das SIFT-Verfahren markante und reproduzierbare Bildpunkte gesucht und anschließend mit FLANN-Algorithmus korrespondierende Merkmale zwischen den beiden Bildern erkannt. Durch den Lowe's Ratio Test werden die optimalen Bildpunktpaare bestimmt. Diese in den jeweiligen Bildern miteinander korrespondierenden Punkte sind in der Abbildung 4.11 durch grüne Linien dargestellt.

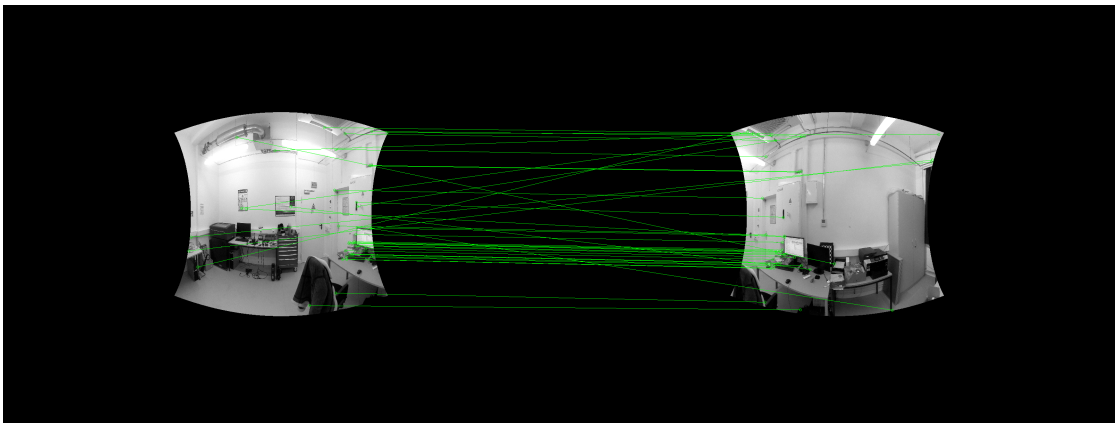


Abbildung 4.11.: Zu sehen sind die durch SIFT und FLANN ermittelten gemeinsamen Merkmale von zwei Kamerabildern.

In der Abbildung 4.11 ist die Mehrheit der grünen Linien horizontal und parallel. Verfolgt man eine dieser Linien vom Anfang im linken Bild bis zum Ende im rechten Bild, stellen sie den gleichen Punkt der betrachteten Szene dar. Fährt man jedoch eine der Linien entlang, welche diagonal verlaufen, kann dies nicht beobachtet werden. Es handelt sich dabei um fehlerhafte Zuordnungen.

Abbildung 4.12.: Ergebnis von zwei mittels der Stitching-Funktion zusammengeführten Kamerabildern

Abbildung 4.12 zeigt das Resultat von zwei zusammengeführten Kamerabildern. Kommen anschließend die Bilder der verbliebenen zwei Kameras hinzu, ergibt sich das Panoramabild in Abbildung 4.12.

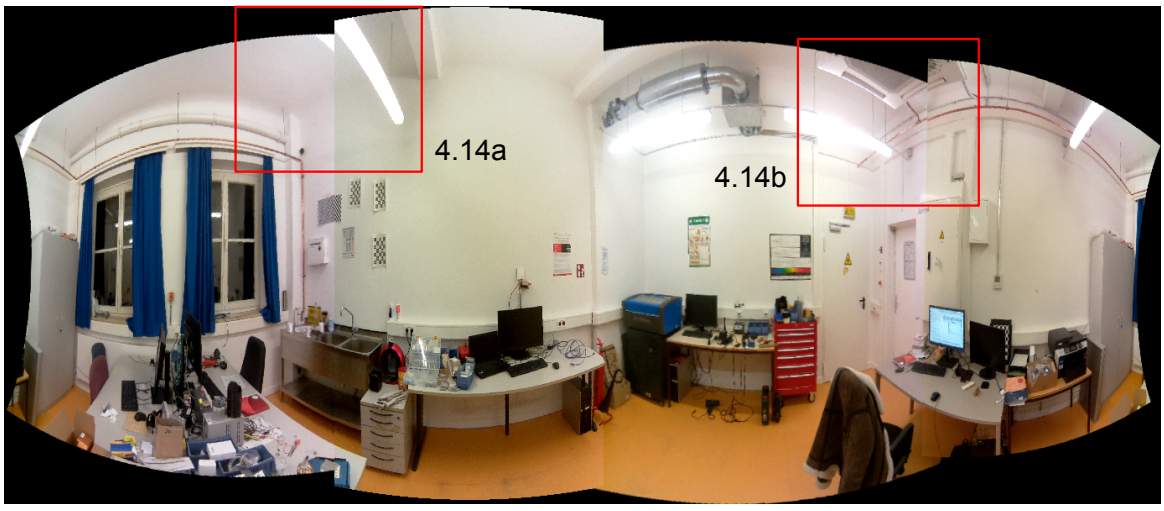


Abbildung 4.13.: Stitching von vier Kameras zu einem sphärischen Panorama mit dem entwickelten Algorithmus (Markierte Fehler in Abbildung 4.14)

Wie die Abbildungen 4.14a und 4.14b zeigen, ergeben sich harte Kanten an den Übergängen zwischen den einzelnen Kameras. Auch wenn darauf geachtet wurde, jede Kamera mit den gleichen Einstellungen zu betreiben, sind vor allem farbliche Unterschiede zu erkennen. Gerade im Low-Cost-Bereich treten Produktionsschwankungen bei den Kameras und Objektiven auf, die sich durch unterschiedliche Farbverläufe, Bildrauschen und Verzeichnungen bemerkbar machen. Auf letztere wird im nächsten Kapitel in Abschnitt 5.1 genauer eingegangen.

Ein weiterer Punkt ist der sogenannte Parallaxenfehler. Dieser tritt auf, wenn zwei Kameras aus verschiedenen Blinkwinkeln eine Szene fotografieren und sich dabei nicht exakt an der gleichen Position befinden.

(a) Erster fehlerhafter Ausschnitt

(b) Zweiter fehlerhafter Ausschnitt

Abbildung 4.14.: Vergrößerte Fehler aus dem erstellten Panorama in Abbildung 4.13

Wenn man einen Gegenstand vor einem Hintergrund von einer festen Position aus in verschiedenen Winkeln betrachtet, bleibt der Hintergrund gleich. Verschiebt sich jedoch der Standort des Betrachters, verändert sich der Hintergrund, dieses Phänomen wird als Parallaxe bezeichnet.

Auch die Projektion in eine Sphäre kann an diesem Phänomen nichts ändern. Bei der Erstellung eines Panoramas sind Parallaxenfehler somit möglichst zu vermeiden und die einzelnen Aufnahmen immer von der Eintrittspupille als Drehpunkt aus zu erstellen. Die Eintrittspupille einer Kamera (siehe Abbildung 4.15) ist eine virtuelle Größe und beschreibt

das Projektionszentrum der Kameralinse. Wenn eine Kamera um genau diesen Punkt gedreht wird und dabei eine Umgebung beobachtet wird, treten keine Parallaxenfehler auf.

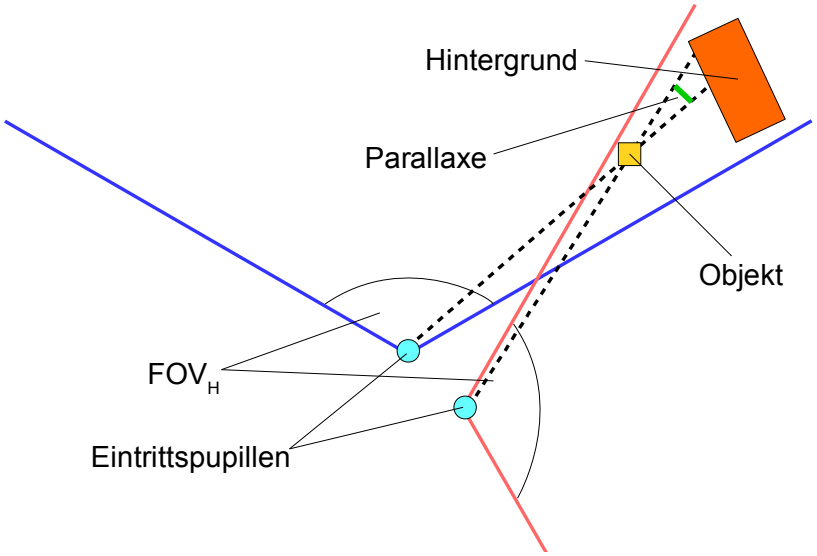


Abbildung 4.15.: Illustration des Pallaxenfehlers

In der Fotografie gibt es Panoramastative, auf denen mittels einer speziellen Vorrichtung eine montierte Kamera so entlang der optischen Achse nach hinten verschoben werden kann, dass die Eintrittspupille mit der Drehachse übereinstimmt.

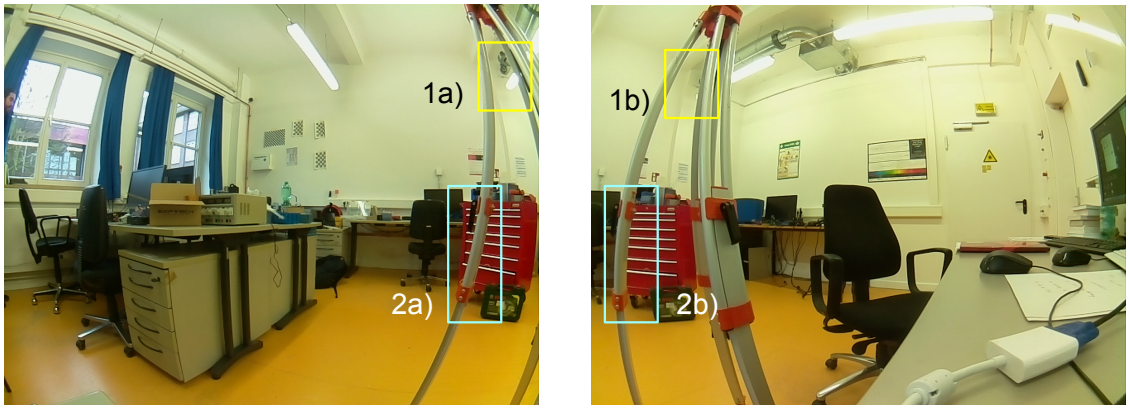
Da es sich bei diesem Projekt nicht nur um eine, sondern um vier Kameras handelt, die in einem Winkel von 90° zueinander angeordnet sind, ist der Parallaxenfehler unvermeidbar. Wenn die Kameras so aufgestellt werden würden, dass sich die jeweiligen Eintrittspupillen der Kameras im Drehmittelpunkt befänden, müssten sie übereinander gestapelt werden. Damit würde zwar im Horizontalen die Parallaxe verschwinden, nun aber vertikal auftreten. Um den Fehler möglichst gering zu halten, wurden deshalb die Halterungen für die Kameras im Gehäuse so nah beieinander wie möglich konstruiert.

Aus einem Nachteil in der Aufnahme von Panoramabildern kann in einem anderen Kontext jedoch sogar ein Vorteil gezogen und sich bewusst zu Nutze gemacht werden. So können durch den Effekt der Parallaxe Entfernungen von Objekten bestimmt werden. Das räumliche Sehvermögen des Menschen ist einzig durch die versetzte Anordnung von zwei

Augen möglich. Dabei gilt: Je größer die Parallaxe, desto weiter ist ein Gegenstand vom Beobachter entfernt.

Dieses, auch als stereoskopisches Sehen bezeichnete, Phänomen stellt einen Teil des maschinellen Sehens dar, bei dem Fähigkeiten des menschlichen visuellen Systems nachgeahmt werden, um durch Computer Lösungen für verschiedenste Aufgabenstellungen zu erhalten [39]. Die Anwendungsgebiete reichen dabei von Fahrzeug- und Robotertechnik zur Entfernungsmessung bis hin zu 3D-Shutterbrillen, um Computerspielen einen räumlichen Eindruck von Tiefe zu verleihen. In den folgenden Kapiteln 5 und 6 wird auf Stereoskopie im Detail eingegangen.

Um in der Praxis den Parallaxenfehler nachzuweisen, wurden absichtlich Aufnahmen (Abbildungen 4.16a und 4.16b) gemacht, in denen dieser besonders deutlich auftritt. Der Effekt wird begünstigt, wenn ein Objekt, das im Sichtfeld von beiden Kameras liegt, sehr nah an die Kamera positioniert und dabei der Hintergrund beobachtet wird. Die Kameras sind in einem Winkel von 90° angeordnet und Objekt ist ein Stativ, die entsprechenden Bereiche wurden hervorgehoben:



(a) Aufnahme der linken Kamera (b) Aufnahme der rechten Kamera

Abbildung 4.16.: Zu sehen sind zwei Aufnahmen im 90°-Winkel zueinander. Beim Vergleich von den Ausschnitten 1a) und 1b) bzw. 2a) und 2b) ist deutlich erkennbar, dass sich das Stativ an anderer Position in Bezug auf den Hintergrund befindet.

Um die Ergebnisse des geschriebenen Programms zu bewerten, wurde die professionelle Stitching-Software PTGui [29] verwendet. Die Bilder müssen dort lediglich in der richtigen

Reihenfolge hochgeladen und der Typ des Objektivs (Full-frame Fisheye) angegeben werden, woraufhin die Software die gemeinsamen Merkmale erkennt und weiß, wie sie die einzelnen Bilder zu transformieren hat. PTgui hat das horizontale Field of View auf 119,2° geschätzt.

Abbildung 4.17.: Mit PTGui erzeugtes Panoramabild. Die Fehler im markierten roten Kästchen sind in Abbildung 4.18 vergrößert dargestellt.

Im entstandenen Panorama (Abbildung 4.17) sind die Übergänge zwischen den Aufnahmen der einzelnen Kameras kaum zu bemerken, da das Programm die Farbverläufe und Helligkeiten automatisch anpasst.

Doch auch mit dieser professionellen Software treten Ungereimtheiten auf (siehe Abbildung 4.18), die auf den Parallaxenfehler zurückzuführen sind.

Abbildung 4.18.: Hervorgehobener Fehler aus dem Panorama in Abbildung 4.17

5. Erzeugung eines Tiefenbildes durch Stereoskopie

In diesem Kapitel wird der Prototyp um stereoskopische Eigenschaften erweitert, was ermöglichen soll, Entfernungen von Objekten zu bestimmen und ein Tiefenbild der Umgebung zu generieren. In Anlehnung an die Ergebnisse des vorherigen Kapitels in Abschnitt 4.3 steht dabei der Parallaxenfehler im Mittelpunkt. Was in der Panoramafotografie nach Möglichkeit zu vermeiden ist, wird in diesem Fall bewusst herbeigeführt.

Stereoskopie bezeichnet die Wiedergabe von Bildern einer bestimmten Szene aus unterschiedlichen Blickwinkeln. Dadurch wird ein räumlicher Eindruck von Tiefe erzeugt und die Bestimmung der Entfernung zwischen dem Beobachter und Objekten ermöglicht. Im Unterschied zu der Panoramakamera sind die optischen Achsen der Kameras nicht in einem Winkel von 90° zueinander angeordnet, sondern verlaufen nun bei der stereoskopischen Kamera parallel in einem fest definierten Abstand.

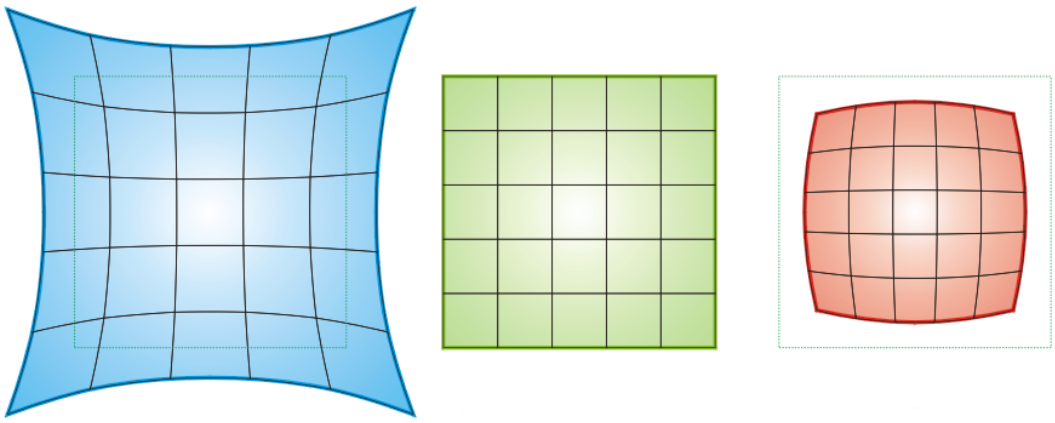
Zuerst wird in Abschnitt 5.1 erklärt, warum in optischen System mit Linsen Abbildungsfehler auftreten, und in Abschnitt 5.2, wie diese Fehler für eine Kamera korrigiert werden können.

Für die stereoskopische Komponente wird daraufhin eine zweite Kamera mit einbezogen und gezeigt, wie beide in Verbindung gebracht werden (Abschnitt 5.3). Mithilfe der zwei Kameras in Abschnitt 5.5 ein Tiefenbild erstellt. Zum Schluss wird die Implementierung beschrieben (Abschnitt 5.6) und ein Fazit gezogen (Abschnitt 5.7).

5.1. Verzeichnung durch Linsen

In der Stereoskopie ist es wichtig, dass die aus unterschiedlichen Blickwinkeln aufgenommenen Bilder maßstabsgetreu sind. Denn die Koordinaten von Objekten werden über die Angabe von korrespondierenden Punkten maßstäblich berechnet. Ideal wäre daher eine Lochkamera (wie in Abschnitt 2.2 beschrieben), die in der Realität jedoch nicht umsetzbar ist. Das Loch kann nicht unendlich klein sein und Lichtstrahlen vom Objekt würden sich streuen, die Abbildung auf der Bildebene wäre unscharf. Außerdem würde es zu Verkrümmungen an den Rändern des Loches kommen, die die Projektion verzerren würden. Abhilfe schaffen hier Linsen, die das Licht auf einen bestimmten Punkt bündeln. Wenn nicht mit sehr hochwertigen und kostspieligen Kameras gearbeitet wird, treten mit Linsen jedoch sogenannte Verzeichnungen auf.

Man unterscheidet zwischen radialen und tangentialen Verzeichnungen, wobei erstgenannte den größeren Anteil haben. Es handelt sich dabei um Abbildungsfehler, bei denen sich der Maßstab der Abbildung in Bezug auf das Originalbild ändert. Wenn das Bild sich an den Rändern der Abbildung verkleinert, nennt man dies kissenförmige Verzeichnung, umgekehrt tonnenförmig. Es können auch Kombinationen aus beiden Fällen auftreten.



a) kissenförmig b) verzeichnungsfrei c) tonnenförmig

Abbildung 5.1.: Zu sehen sind die beiden gängigsten Fälle der radialen Verzeichnung: a) die Kissenförmige und c) die Tonnenförmige. Abbildung b) stellt den verzeichnungsfreien Fall dar (Quelle: [10]).

Radiale Verzeichnungen wird mit den ersten vier Termen der Taylorreihe k_0, k_1, k_2, k_3 gelöst, wobei $k_0 = 1$ ist.

$$x_{rad} = x' \cdot (1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) \tag{5.1}$$

$$y_{rad} = y' \cdot (1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) \tag{5.2}$$

$$r^2 = x'^2 + y'^2 \tag{5.3}$$

x und y entsprechen den Originalkoordinaten des verzeichneten Punktes und x_{rad} und y_{rad} den korrigierten Werten: $x' = \frac{X_C}{Z_C}, \quad y' = \frac{Y_C}{Z_C}$

Bei Weitwinkel- oder Fisheye-Objektiven reichen die ersten vier Koeffizienten unter Umständen nicht aus und es wird ein erweitertes sogenanntes rationales Modell [5] verwendet bei dem drei Koeffizienten hinzukommen:

$$x_{rad,rat} = x' \cdot \frac{1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6}{1 + k_4 \cdot r^2 + k_5 \cdot r^4 + k_6 \cdot r^6} \tag{5.4}$$

$$y_{rad,rat} = y' \cdot \frac{1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6}{1 + k_4 \cdot r^2 + k_5 \cdot r^4 + k_6 \cdot r^6} \tag{5.5}$$

Am zweithäufigsten sind tangentiale Verzeichnungen. Diese treten insbesondere bei minderwertiger Verarbeitung auf, wenn der Kamerasensor nicht parallel zur Linse verbaut ist. Entweder wurde dabei von vornherein während der Produktion der Kamera nicht genau gearbeitet oder ein schlechter Kleber führt nach einiger Zeit zu einer Verlagerung der Linse oder des Sensors.

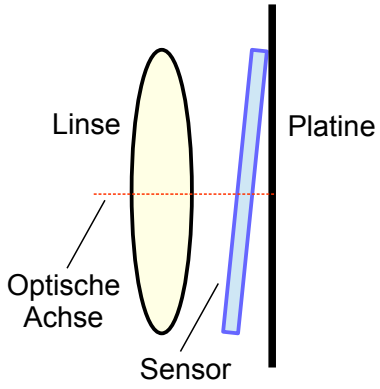


Abbildung 5.2.: Tangentiale Verzeichnungen treten bei schlechter Verarbeitung auf, wenn der Kamerasensor nicht parallel zur Linse ist.

Tangentiale Verzeichnungen werden in OpenCV wie folgt gelöst. Die Herleitung ist Brown [23] zu entnehmen:

$$
\begin{aligned}
x_{tan} &= x' + [2 \cdot p_1 \cdot x' \cdot y' + p_2 \cdot (r^2 + 2 \cdot x'^2)] \\
y_{tan} &= y' + [p_1 \cdot (r^2 + 2 \cdot y'^2) + 2 \cdot p_2 \cdot x' \cdot y'] \\
r^2 &= x'^2 + y'^2
\end{aligned}
$$

Bei der Kombination von tangentialen und radialen Verzeichnungen ergeben sich damit folgende Gleichungen:

$$
\begin{aligned}
x'' &= x' \cdot (1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) + [2 \cdot p_1 \cdot x' \cdot y' + p_2 \cdot (r^2 + 2 \cdot x'^2)] \\
y'' &= y' \cdot (1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) + [p_1 \cdot (r^2 + 2 \cdot y'^2) + 2 \cdot p_2 \cdot x' \cdot y'] \\
r^2 &= x'^2 + y'^2
\end{aligned}
$$

u und v sind die projizierten Bildkoordinaten auf der Bildebene:

$$
\begin{aligned}
u &= f_x \cdot x'' + c_x \\
v &= f_y \cdot y'' + c_y
\end{aligned}
$$

In OpenCV und anderen Kalibrierungstools werden die Verzeichnungskoeffizienten als 5x1-Vektor bzw. mit dem rationalen Modell als 8x1-Vektor dargestellt:

$$\mathbf{D} = (k_1,\ k_2,\ p_1,\ p_2,\ k_3[,\ k_4,\ k_5,\ k_6]) \tag{5.6}$$

5.2. Kalibrierung der einzelnen Kameras

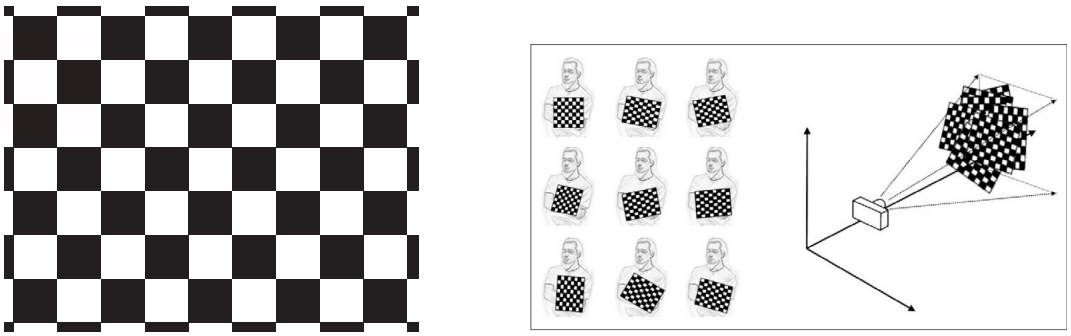
Um die auftretenden radialen und tangentialen Verzeichnungen zu korrigieren und damit die Kamera dem Modell der Lochkamera anzunähern, muss eine Kalibrierung vorgenommen werden. Ziel ist es, die intrinsischen Parameter, also die Kameramatrizen und Verzeichnungskoeffizienten, sowie extrinsischen Parameter zu bestimmmen. Letztere beschreiben die Orientierung der Kamera im Raum.

Der erste Schritt zur Kalibrierung der Stereokamera ist die Kalibrierung der einzelnen Kameras. Es könnte auch direkt mit der Stereokalibrierung begonnen werden, da diese in OpenCV neben den Informationen, wie die beiden Kameras miteinander zusammenhängen sowohl die intrinsischen als auch die extrinsischen Parameter für die jeweiligen Kameras berechnet. Bessere Ergebnisse wurden allerdings erzielt, wenn zuerst jede Kamera einzeln kalibriert und anschließend die intrinsischen Parameter als Initialwerte an die Funktion der Stereokalibrierung übergeben werden.

Auch wenn es sich bei den Kameras um das gleiche Modell vom selben Hersteller handelt, muss jedes Exemplar einzeln kalibriert werden, da Abweichungen in der Produktion unvermeidbar sind und zu unterschiedlichen Verzeichnungen führen.

Die Kalibrierung basiert auf der Verwendung eines Kalibriermusters. In diesem Fall wurde als Muster ein Schachbrett gewählt, also ein einheitliches, auf Papier gedrucktes Muster aus abwechselnd schwarzen und weißen Quadraten mit einer Kantenlänge von 32 mm (siehe Abbildung 5.3a). Horizontal hat es 9 Quadrate und vertikal 7, zur Stabilisierung wurde der Ausdruck auf einem Holzbrett fixiert. Ebenso würde die Kalibrierung mit einem regelmäßigen Muster aus Kreisen oder zum Beispiel einem dreidimensionalen Würfel mit

einem Schachbrettmuster auf jeder Seite funktionieren. Wichtig ist lediglich ein eindeutig bestimmbares Gitternetz aus markanten Punkten. Bei Kreisen sind dies die jeweiligen Kreisschwerpunkte und bei Quadraten die Schnittpunkte der Kantenlängen.



(a) Verwendetes Schachbrettmuster zur Kalibrierung. Ausgedruckt beträgt die Kantenlänge eines Quadrates 32 mm.

(b) Muster wird in verschiedenen Winkeln aufgenommen (Quelle: [4]).

Abbildung 5.3.: Zu sehen ist das Kalibriermuster und der Vorgang der Kalibrierung.

Um die intrinsischen und extrinsischen Parameter zu ermitteln, sind theoretisch nur zwei Aufnahmen des Schachbrettmusters erforderlich. Da im Allgemeinen und insbesondere im Fall der kostengünstigen Raspberry Pi Kamera ein Rauschen in den Bildern vorliegt, empfiehlt OpenCV mindestens 10 Bilder zu verwenden. Aus den Ergebnissen wird im Anschluss der Mittelwert gebildet.

Bei der Erstellung der Aufnahmen wurde außerdem darauf geachtet, das Schachbrett in unterschiedlichen Winkeln und Entfernungen zur Kamera zu positionieren (siehe Abbildung 5.3b), um die Kalibrierungsfunktion mit möglichst vielen verschiedenen Werten zu speisen.

Für einen ersten Test der Stereokamera wurde mithilfe einer Laserverarbeitungsmaschine ein steckbares Modul aus Kraftplex geschnitten und mit Heißkleber fixiert (siehe Abbildung 5.4). Die optischen Achsen der beiden Kameras sind in der Vorrichtung parallel und in einem Abstand von 64 mm zueinander angebracht. Bei einer Stereokamera wird diese Entfernung als stereoskopische Basislinie bezeichnet. Um eine raumtreue Wiedergabe zu gewährleisten, wurde der spezifische Wert von 64 mm gewählt, da er dem durchschnittlichen menschlichen Augenabstand enstspricht.

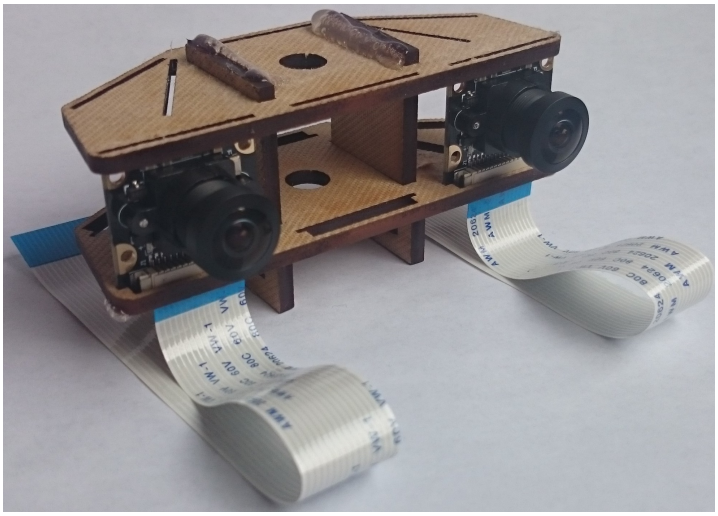


Abbildung 5.4.: Zu Testzwecken angefertigtes steckbares Modul, bei dem zwei Kameras im Augenabstand (64 mm) zueinander eingebaut sind.)

Im Test wurden mit 60 Aufnahmen pro Kamera mehr als ausreichend Bilder gemacht, bei denen das Muster von beiden Kameras aus gleichzeitig und vollständig zu sehen ist. Dies spielt insbesondere für die anschließende Stereokalibrierung (siehe Abschnitt 5.3) eine wichtige Rolle. Es hat sich als sinnvoll erwiesen, für die Einzelkalibrierung mit 30 bis 40 erfolgreich erkannten Kalibriermustern zu arbeiten, weshalb das Programm nach 40 Mustern stoppt und die Kameraparameter berechnet.

- *cv2.CALIB_USE_INTRINSIC_GUESS*: Ermöglicht es, Anfangswerte der Funktion zu übergeben.
- *cv2.CALIB_FIX_PRINCIPAL_POINT*: Belässt das Offset des Bildhauptpunktes auf seinem Wert, sofern einer übergeben wurde, ansonsten wird vom Bildzentrum ausgegangen.
- *cv2.CALIB_FIX_ASPECT_RATIO*: Brennweiten werden auf Eingangswert belassen, wenn gesetzt.
- *cv2.CALIB_ZERO_TANGENT_DIST*: Die tangentialen Verzeichnungsparameter werden auf Null gesetzt.
- *cv2.CALIB_RATIONAL_MODEL*: Aktiviert das rationale Modell.

- *cv2.CALIB_FIX_K1* bis *cv2.CALIB_FIX_K6*: Belässt den gewählten radialen Verzeichnungsparameter auf seinem Wert, wenn ein Anfangswert übergeben wurde, andernfalls wird er auf Null gesetzt.

Mit dem beschriebenen Prototypen ergeben sich beispielhaft die unbearbeiteten und verzeichneten Bilder aus Sicht der linken (5.5a) und rechten Kamera (5.5b). Anhand dieser beiden Aufnahmen werden im Verlauf dieses Kapitels die Kalibrierung der einzelnen Kameras und die der Stereokamera beschrieben.

(a) Originalbild der linken Kamera (b) Originalbild der rechten Kamera

Abbildung 5.5.: Verzeichnete Aufnahmen der Stereokamera, deren einzelne Kameras sich in einem Abstand von 64 mm zueinander befinden.

In Kapitel 2.2 wurde das in der Computergrafik übliche Modell der Kamera erklärt und abschließend die Homographie hergeleitet (Gleichung 2.11). Mit dieser werden für jedes Schachbrettmuster die Transformation der Punkte zwischen Objekt- und Bildebene ermittelt. In der Homographie sind sowohl die Parameter für die innere als auch für die äußere

Orientierung der Kamera enthalten.

$$\mathbf{q} = \mathbf{H} \cdot \mathbf{Q}$$

$$\mathbf{q} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \mathbf{Q}$$

$$\mathbf{q} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix} \cdot \mathbf{Q}$$

Wie die Koeffizienten der Homographie berechnet werden, ist im Grundlagenkapitel in Abschnitt 2.1.2 erklärt. Es werden vier oder mehr korrespondierende Punkte benötigt.

(a) Erkanntes Schachbrett links (b) Erkanntes Schachbrett rechts

Abbildung 5.6.: Zu sehen sind zwei verzeichnete Aufnahmen, auf denen jeweils das Kalibriermuster erkannt wurde. Zur Veranschaulichung bestätigt dies OpenCV durch die Markierung der inneren Ecken. Durch die Punktreihen werden Linien gezogen un die Reihen diagonal durch weitere Linien verbunden.

Die ermittelten Werte für die Kameramatrizen der linken $\mathbf{K_L}$ und rechten Kamera $\mathbf{K_R}$ sind

wie folgt:

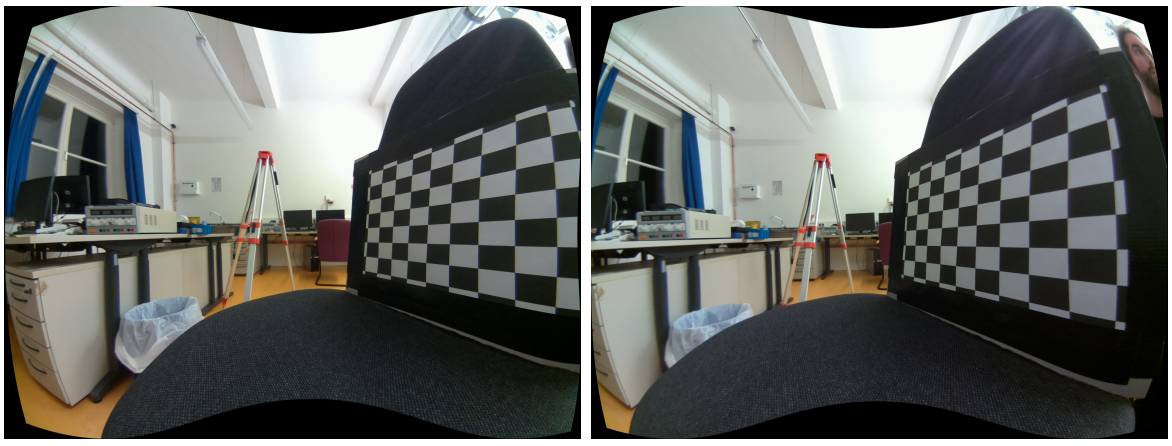
$$\mathbf{K_L} = \begin{bmatrix} f_{x_L} & 0 & c_{x_L} \\ 0 & f_{y_L} & c_{y_L} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 779.243 & 0 & 852.845 \\ 0 & 775.456 & 635.539 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.7}$$

$$\mathbf{K_R} = \begin{bmatrix} f_{x_R} & 0 & c_{x_R} \\ 0 & f_{y_R} & c_{y_R} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 782.122 & 0 & 793.057 \\ 0 & 779.203 & 641.018 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.8}$$

Die besten Ergebnisse konnten bei der Einzelkalibrierung erzielt werden, wenn der Verzechnungsparameter k_3 (*cv2.CALIB_FIX_K1*) und die tangentialen Verzeichnungen (*cv2.CALIB_ZERO_TANGENT_DIST*) auf Null gesetzten wurden. Für die Verzeichnungsparameter der linken $\mathbf{D_L}$ und rechten Kamera $\mathbf{D_R}$ wurde dies ermittelt:

$$\mathbf{D_L} = (k_{1_L},\ k_{2_L},\ p_{1_L},\ p_{2_L},\ k_{3_L}) = (-0.280,\ 0.056,\ 0,\ 0,\ 0)$$
$$\mathbf{D_R} = (k_{1_R},\ k_{2_R},\ p_{1_R},\ p_{2_R},\ k_{3_R}) = (-0.287,\ 0.061,\ 0,\ 0,\ 0)$$

Mit diesen durch OpenCV berechneten intrinsischen Parametern sehen die korrigierten Aufnahmen der Stereokamera nun folgendermaßen aus:



(a) Korrigiertes Bild der linken Kamera (b) Korrigiertes Bild der rechten Kamera

Abbildung 5.7.: Dargestellt sind zwei Aufnahmen der Stereokamera, die mit den ermittelten intrinsischen Parametern der Einzelkalibrierung korrigiert wurden. In den Randbereichen sind die Linien des Schachbrettmusters nicht gerade.

5.3. Kalibrierung der Stereokamera

Während es im vorherigen Kapitel in Abschnitt 5.2 vor allem darum ging, die intrinsischen Parameter herauszufinden, um die Verzeichnung der einzelnen Kameras zu korrigieren, kommt nun hinzu, anhand der extrinsischen Parameter die beiden Kameras miteinander in Verbindung zu setzen, also gemeinsam als Stereokamera zu betrachten.

Wie bei der Einzelkalibrierung können auch in der OpenCV-Funktion der Stereokalibrierung *cv2.stereoCalibrate* eine Reihe von Flags gesetzt werden. Dabei können die gleichen Flags, wie in der Einzelkalibrierung gesetzt, an dieser Stelle werden nur die hinzugekommen Flags beschrieben.

- *cv2.CALIB_FIX_INTRINSIC*: Setzt die intrinsischen Parameter auf die Werte, sie werden nicht weiter optimiert.
- *cv2.CALIB_FIX_FOCAL_LENGTH*: Verändert die Brennweiten nicht während der Optimierung.
- *cv2.CALIB_SAME_FOCAL_LENGTH*: Wenn gesetzt, wird davon ausgegangen, dass beide Kameras die gleiche Brennweite haben.

Bei der Stereokalibrierung wird nun in OpenCV das rationale Modell 5.1 zur Korrektur von radialen Verzeichnungen per Flag aktiviert, dadurch werden mehr Koeffizienten der Taylorreihe einbezogen (siehe Gleichungen 5.4 und 5.5).

Es hat zu besseren Ergebnissen geführt, nur 10 bis 20 Muster für die Stereokalibrierung zu verwenden, deshalb stoppt das Programm nach genau 20 erkannten Schachbrettern und geht zur Berechnung über.

Durch setzen der *cv2.CALIB_USE_INTRINSIC_GUESS* werden die intrinsischen Parameter der linken Kamera $\mathbf{K_L}$ (Gleichung 5.7) und rechten Kamera $\mathbf{K_R}$ (Gleichung 5.8) aus der Einzelkalibrierung als Initialwerte an OpenCV übergeben und sollen dadurch optimiert

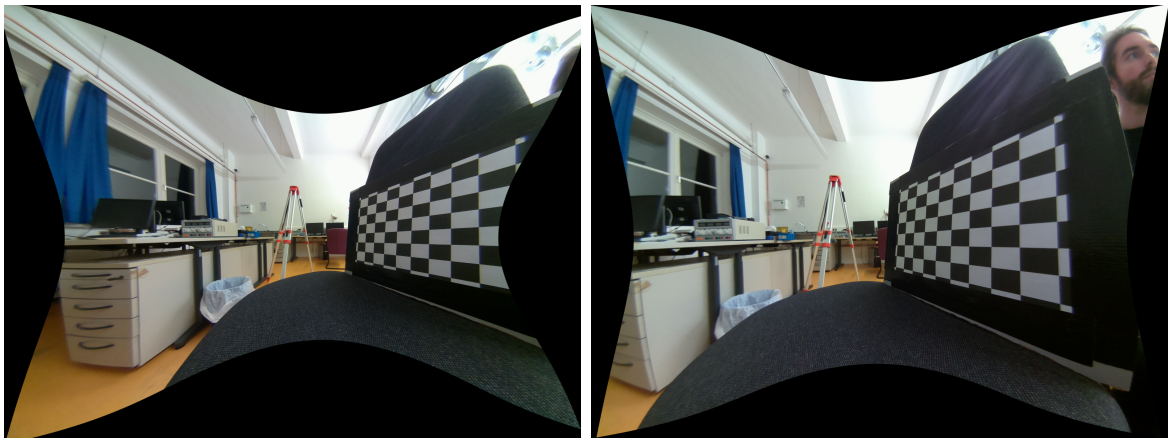
werden. Es ist festzustellen, dass sich die Kameramatrizen nicht geändert haben:

$$\mathbf{K_{L,S}} = \begin{bmatrix} f_{x_L} & 0 & c_{x_L} \\ 0 & f_{y_L} & c_{y_L} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 779.243 & 0 & 852.845 \\ 0 & 775.456 & 635.539 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{K_{R,S}} = \begin{bmatrix} f_{x_R} & 0 & c_{x_R} \\ 0 & f_{y_R} & c_{y_R} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 782.122 & 0 & 793.057 \\ 0 & 779.203 & 641.018 \\ 0 & 0 & 1 \end{bmatrix}$$

Durch die Aktivierung des rationalen Modells wurden die Verzeichnungsparameter $\mathbf{D_L}$ und $\mathbf{D_R}$ optimiert. Die Parameter für die tangentialen Verzeichnungen sind Null, da die entsprechende Flag wieder gesetzt wurde.

$$\begin{aligned} \mathbf{D_{L,S}} &= (k_{1_L},\ k_{2_L},\ p_{1_L},\ p_{2_L},\ k_{3_L},\ k_{4_L},\ k_{5_L},\ k_{6_L}) \\ &= (-0.042,\ 0.023,\ 0,\ 0,\ 0.012,\ 0.319,\ -0.066,\ 0.04) \\ \mathbf{D_{R,S}} &= (k_{1_R},\ k_{2_R},\ p_{1_R},\ p_{2_R},\ k_{3_R},\ k_{4_R},\ k_{5_R},\ k_{6_R}) \\ &= (-0.201,\ 0.012,\ 0,\ 0,\ 0.039,\ 0.178,\ -0.167,\ 0.098) \end{aligned}$$



(a) Korrigiertes Bild der linken Kamera (b) Korrigiertes Bild der rechten Kamera

Abbildung 5.8.: Die Abbildung zeigt die korrigierten Bilder der Stereokamera nach der Stereokalibrierung.

Ziel der Stereokalibrierung ist es, die Koordinatensysteme der beiden Kameras in ein Gemeinsames zu überführen. Dafür ist es wichtig, die Rotation und Translation zwischen den Kamerakoordinatensystemen zu ermitteln (siehe Abbildung 5.9).

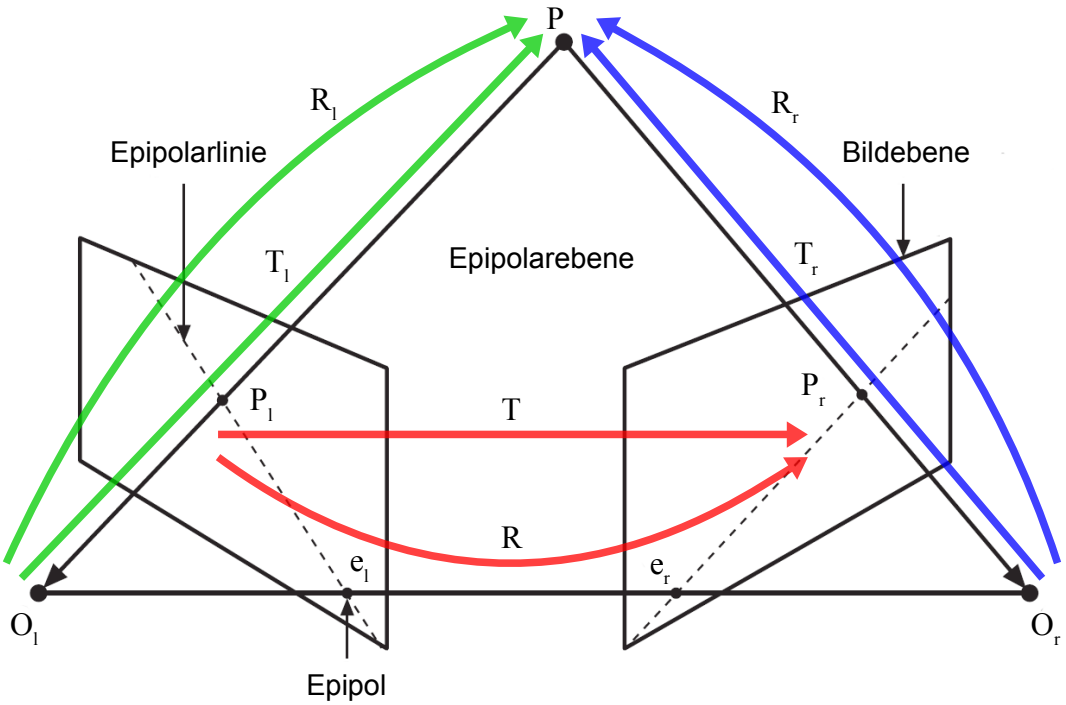


Abbildung 5.9.: Abbildungen des Weltpunktes auf die Bildebenen Illustration der zwei Kamerakoordinatensysteme (Quelle: in Anlehnung an [4])

Der Punkt $\mathbf{P}$ ist ein beliebiger Punkt auf einem Objekt, der gleichzeitig von der linken sowie rechten Kamera beobachtet und in Weltkoordinaten angegeben wird. $\mathbf{P_l}$ und $\mathbf{P_r}$ sind Abbildungen des Weltpunktes auf den Bildebenen der beiden Kameras. Für die linke bzw. rechte Kamera gilt in Kamerakoordinaten:

$$\mathbf{P_l} = \mathbf{R_l} \cdot \mathbf{P} + \mathbf{T_l} \tag{5.9}$$

$$\mathbf{P_r} = \mathbf{R_r} \cdot \mathbf{P} + \mathbf{T_r} \tag{5.10}$$

Zwischen den Punkten $\mathbf{P_l}$ und $\mathbf{P_r}$ kann folgende Beziehung hergestellt werden, dabei wird

das rechte ins linke Kamerakoordinatensystem überführt:

$$\mathbf{P_l} = \mathbf{R}^{\mathbf{T}} \cdot (\mathbf{P_r} - \mathbf{T}) \tag{5.11}$$

Die Lösungen für Rotation $\mathbf{R}$ und Translation $\mathbf{T}$ zwischen den beiden Kamerakoordinatensystem können anschließend aus den drei Gleichungen 5.9, 5.10 und 5.11 ermittelt werden:

$$\mathbf{R} = \mathbf{R_r} \cdot (\mathbf{R_l})^{\mathbf{T}}$$
$$\mathbf{T} = \mathbf{T_r} - \mathbf{R} \cdot \mathbf{T_l}$$

Die Kalibrierung der Stereokamera hat in OpenCV folgende Werte zum Ergebnis:

$$\mathbf{R} = \begin{bmatrix} 1 & 0.013 & 0 \\ -0.013 & 1 & 0.007 \\ 0 & -0.007 & 1 \end{bmatrix}$$
$$\mathbf{T} = \begin{bmatrix} -0.064 \\ 0.001 \\ 0 \end{bmatrix} \tag{5.12}$$

Die erste Zeile des Translationsvektors (Gleichung 5.12) beträgt -0.064. In welcher Einheit dieser Wert ist, hängt davon ab, wie ein einzelnes Quadrat des Musters definiert wurde, da nur Maßstäbe berechnet werden. Ein Quadrat hat eine Kantenlänge von 32 mm und wurde mit 0.032 an das Programm übergeben. Dementsprechend ist die Einheit Meter und der Wert entspricht 64 mm, also genau der Distanz, in der die Kameras horizontal voneinander entfernt im Testmodul angeordnet sind. Des Weiteren kann der zweiten Zeile entnommen werden, dass die Kameras 1 mm in vertikaler Richtung voneinander verschoben sind.

5.4. Rektifizierung von Kamerabildern

In Abschnitt 4.3 wurde erklärt, dass der Parallaxenfehler auftritt, wenn Kameras von unterschiedlichen Positionen aus Aufnahmen machen. Ein beobachtetes Objekt verschiebt sich dadurch in Bezug auf den Hintergrund. Diese Parallaxe bzw. Positionsdifferenz wird in der Bildverarbeitung als Disparität bezeichnet.

In den Aufnahmen der linken und rechten Kamera gibt es miteinander korrespondierende Pixel, die dieselben Elemente einer aufgenommenen Szene darstellen, aber aufgrund der verschiedenen Perspektiven unterschiedliche Pixelkoordinaten in den Bildern haben. Ziel ist es, diese Korrespondenz zwischen den Bildelementen zu bestimmen. Die Suche nach den korrespondierenden Punkten kann reduziert werden, wenn sie eindimensional, also zeilenweise erfolgt.

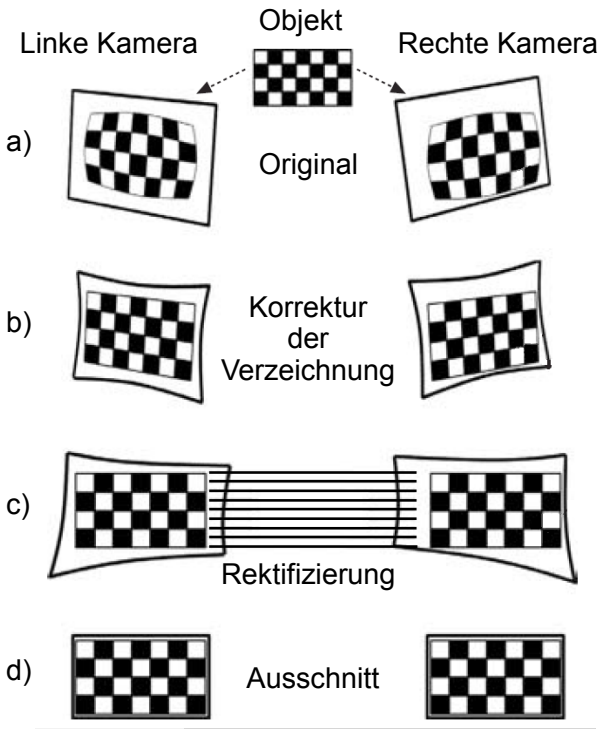
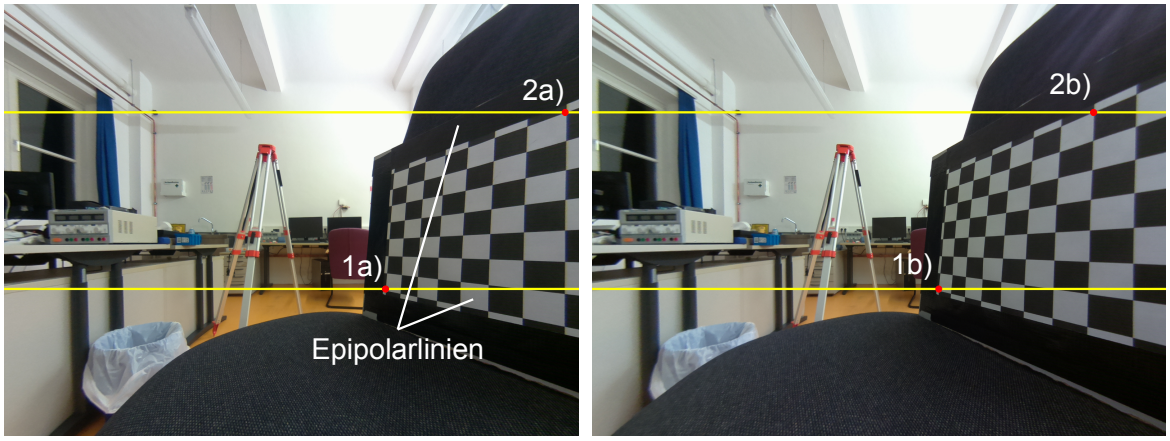


Abbildung 5.10.: Die Abbildung veranschaulicht den Prozess vom Originalbild bis zur Rektifizierung. (Quelle: in Anlehnung an [4])

Im Testaufbau wurde zwar darauf geachtet, dass die optischen Achsen der Kameras par-

allel sind, aber eine exakte Ausrichtung gestaltet sich alleine schon wegen Produktionsunterschieden schwierig. Diese Ausrichtung geschieht bei der Rektifizierung virtuell. Dabei werden die in Abbildung 5.9 dargestellten Epipolarlinien horizontal ausgerichtet, sodass beide Kameras in derselben Ebene liegen (siehe Abbildung 5.11). Die Drehung wird aus den Kameramatrizen und extrinsischen Parametern berechnet. Bei diesem Vorgang geht ein Teil des Bildes und damit des Sichtfeldes der Kamera verloren, dawie in Abbildung 5.10 dargestellt ist.



(a) Aufnahme der linken Kamera (b) Aufnahme der rechten Kamera

Abbildung 5.11.: Die Abbildung zeigt die rektifizierten Aufnahmen der linken und rechten Kamera. 1a) und 1b) bzw. 2a) und 2b) stellen jeweils denselben Objektpunkt auf dem Muster dar und befinden sich auf Epipolarlinien.

5.5. Erzeugung einer Disparitätenkarte

Die Disparität gibt Auskunft über die Tiefe eines Pixels (siehe Abbildung 5.12). Eine Disparitätenkarte (siehe Abbildung 5.14) ist ein Graustufenbild, das über die Entfernung von Objekten in einem Raum Auskunft gibt. Um die Disparitätenkarte zu berechnen, wird eine Korrespondenzanalyse durch den Vergleich von Intensitätswerten durchgeführt. Es werden in der Verzeichnung korrigierte und rektifizierte Bilder vorausgesetzt. Für ein Pixel im linken Bild x_l wird in der gleichen Zeile im rechten Bild das dazugehörige Pixel x_r ausfindig gemacht, das den gleichen Punkt des beobachteten Objektes zeigt. Die Differenz zwischen

den Pixelkoordinaten ist damit die Disparitäten und wird folgendermaßen bestimmt:

$$d = x_l - x_r \tag{5.13}$$

Darüber hinaus kann mithilfe der Basislinie B zwischen den Kameras und der Brennweite f kann durch Triangulation die Tiefe eines Punktes ermittelt werden:

$$Z = \frac{f \cdot T}{x_l - x_r} \tag{5.14}$$

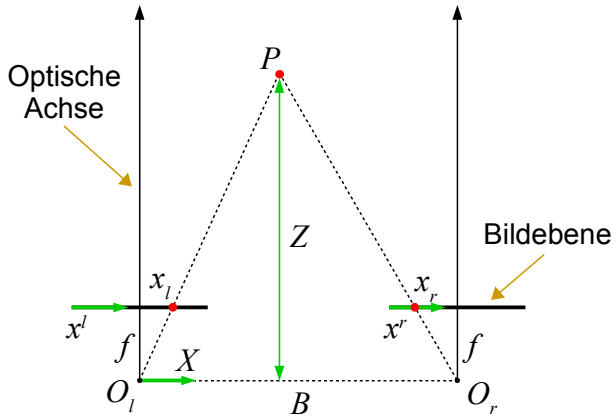


Abbildung 5.12.: Zweidimensionale Betrachtung der zwei Kamerakoordinatensysteme einer Stereokamera zur Veranschaulichung des Zusammenhangs zwischen Disparität und Tiefe.

Die Abbildung 5.13 veranschaulicht den Zusammenhang anhand von Beispielbildern. Es werden jeweils durch zwei Punkte, die dasselbe Element einer Szene zeigen, der Zusammenhang zwischen Disparität und Entfernung gezeigt.

Je größer die Disparität, desto weiter im Vordergrund befindet sich das Pixel und desto heller ist der Grauwert, mit dem es in der Disparitätenkarte dargestellt wird. Befindet sich ein Objekt weiter im Hintergrund, so wird es in dunkleren Grautönen abgebildet.

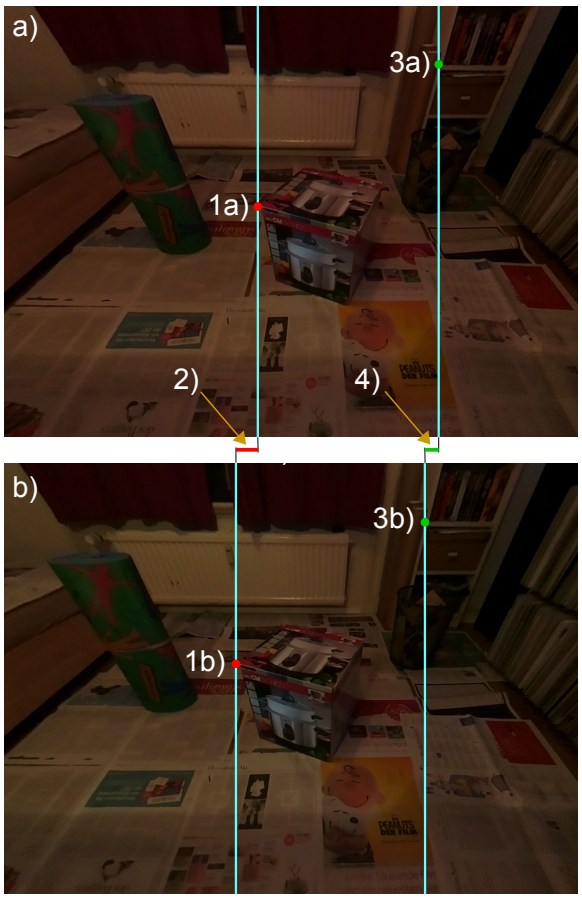


Abbildung 5.13.: Zu sehen sind zwei rektifizierte Aufnahmen der Stereokamera. Es wurde absichtlich eine Szene gewählt, die viele verschiedene Texturen enthält. Der Algorithmus arbeitet mit mehr Textur genauer, da Objektpunkte besser unterschieden und korrespondierende Punkte in den Bildern leichter ermittelt werden können.

Der Algorithmus, mit dem in OpenCV eine Disparitätenkarte erstellt wird, nennt sich *Semiglobal Block-Matching*. Dabei handelt es sich um eine aus Effizienzgründen modifizierte Version des *Semiglobal Matching*-Algorithmus von Heiko Hirschmüller [13]. Es werden dabei nicht einzelne Pixel zeilenweise in den beiden korrigierten und rektifizierten Bildern miteinander vergleichen, sondern es wird mit Blöcken bzw. Fenstern bestehend aus mehreren Pixeln gearbeitet. Außerdem wurde im Vergleich zur ursprünglichen Version eine einfachere Kostenfunktion implementiert und Vor- und Nachbearbeitungsfunktionen für die Disparitätenkarte hinzugefügt.

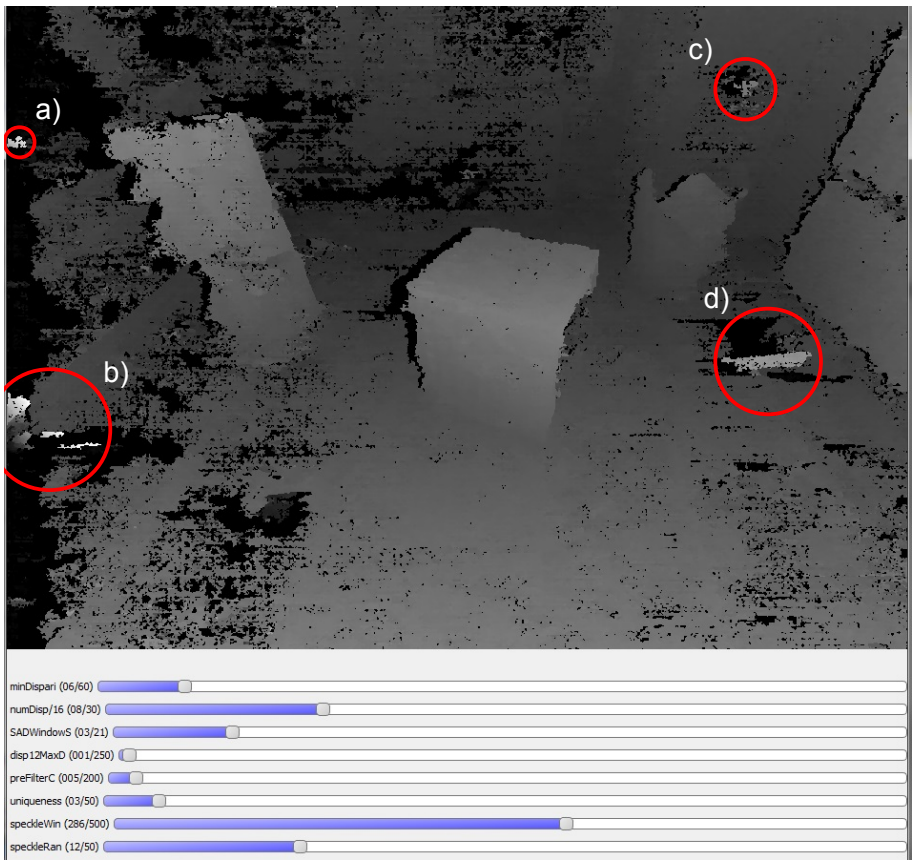


Abbildung 5.14.: Die Abbildung zeigt die Disparitätenkarte aus den beiden Bildern aus Abbildung 5.13. Das Programm wurde so geschrieben, dass mit Schiebereglern die Parameter der *Semiglobal Block-Matching*-Funktion verändert und so die optimalen Einstellungen ermittelt werden können. Bei den Markierungen a), b), c) und d) handelt es sich um Bildfehler, sogenannte Speckles.

Der *Semiglobal Block-Matching*-Algorithmus von OpenCV umfasst eine Reihe von Parametern, die das Ergebnis der Disparitätenkarte beeinflussen. Es kommt vor, dass die Bilder durch die Rektifizierung verschoben werden, dies kann mit dem *minDisparity*-Parameter ausgeglichen werden, wobei der Wert 0 keiner Verschiebung entspricht (siehe [23]). Der Algorithmus arbeitet mit der Summe der absouluten Differenzen der Intensitätswerte innerhalb eines definierten Suchfensters. Die Größe dieses Fensters wird mit *SADWindowSize* festgelegt. Um die Suche nach korrespondierenden Pixel noch weiter zu verfeinern, kann mit *numDisparities* die Suchweite in Pixeln angegeben werden.

Der Parameter *preFilterCap* gehört zu einem Teil der Vorverarbeitung. Beide Bilder werden dabei gefiltert, um Unterschiede in der Belichtung zu kompensieren und die Strukturen in der Textur zu erhöhen. Der Parameter bestimmt die obere Grenze des Intensitätsunterschieds. *disp12MaxDiff* beschreibt die maximal erlaubt Differenz in Pixeln, die zwischen korrespondierenden Pixeln im linken und rechten Bild auftreten darf. Ein negativer Wert deaktiviert dieses Verfahren.

Zu der Nachbearbeitung zählt der Parameter *uniquenessRatio*. Wenn im rechten Bild ein korrespondierender Punkt ermittelt wurde, jedoch nicht weit entfernt davon ein Punkt am zweitbesten korrespondiert, wird der Punkt nicht verwendet. Die Wahrscheinlichkeit ist groß, dass es sich um eine fehlerhafte Zuordnung handeln könnte. Der Parameter legt das Verhältnis fest, in dem sich der beste Treffer von allen anderen unterscheiden muss.

Innerhalb eines Suchfensters kann es zu starken Schwankungen in der Disparität führen, zum Beispiel durch einen starken Distanzunterschied an Objektkanten. Dadurch können Bildfehler auftreten, die sich als weiße Punkte in der Disparitätenkarte bemerkbar machen, sogenannte *Speckles*. In der Nachbearbeitung kann über eine festgelegte Fenstergröße (*speckleWindowSize*) überprüft werden, ob die Differenz der kleinsten und größten Disparität innerhalb eines definierten Bereichs (*speckleRange*) liegt und dementsprechend die *Speckles* gefiltert werden.

Abbildung 5.14 zeigt die mit dem Stereomodul erstellte Disparitätenkarte (die Werte sind noch einmal in Tabelle 5.19 aufgeführt). Die Markierungen a), b), c) und d) zeigen Speckles, die durch keine Variation der Parameter herausgefiltert werden konnten. Speckles treten vor allem bei einem abrupten Helligkeitsgefälle auf. Es ist außerdem wichtig, dass die Szene viele Texturen enthält. Einfarbige Flächen werden häufig nicht korrekt zugeordnet und in schwarz dargestellt.

5.6. Programmierung

OpenCV fasst die komplexe Kalibrierung der Kameras in einer übersichtlichen Zahl von Funktionen zusammen (siehe Abbildung 5.15). Als erstes werden die aus unterschiedlichen Winkeln und Distanzen erstellten Aufnahmen des Schachbrettmusters

mittels *cv2.imread* eingelesen (5.15a) und die Kanten der Schachbrettquadrate durch *cv2.findChessboardCorners* gesucht (5.15b). An den Kanten werden Linien gezogen und deren Schnittpunkte gespeichert. Sobald das Muster 40 mal erfolgreich erkannt wurde, werden die Daten an die Funktion die Funktion *cv2.calibrateCamera* (5.15c) übergeben und die intrinsischen, einschließlich der Kameramatrizen und Verzeichnungsparameter, sowie die extrinsischen Parameter durch berechnet.

Anschließend wird die Flag *cv2.CALIB_USE_INTRINSIC_GUESS* in der Funktion *cv2.stereoCalibrate* (5.15e) gesetzt und damit die ermittelten intrinsischen Parameter als Initialwerte verwendet, die zuvor berechneten Extrinsischen werden somit verworfen. Darüber hinaus wird mit der Flag *cv2.CALIB_RATIONAL_MODEL* das rationale Modell für Verzeichnung (siehe Abschnitt 5.1) aktiviert, wodurch die Verzeichnungsparameter um $k4$ bis $k6$ erweitert und auch die Parameter $k0$ bis $k3$ optimiert werden sollen.

Für die Stereokalibrierung können dieselben Aufnahmen wie für die Einzelkalibrierung verwendet werden, sofern darauf geachtet wurde, dass von beiden Kameras aus das Muster vollständig zu sehen ist. Da für beide Kameras die Bilder gleichzeitig nach dem Muster abgescannt (5.15d) werden, ist es wichtig, die Aufnahmen in der richtigen Reihenfolge einzulesen. Nur wenn in beiden Aufnahmen gleichzeitig das Muster erfolgreich erkannt wird, werden die entsprechenden Punkte gespeichert. Nach 20 Treffern geht die Stereokalibrierung zur Berechnung und Optimierung der Kameraparameter.

Anhand der extrinsischen Parameter der beiden Kameras werden Rotation und Translation der Koordinatensysteme zueinander berechnet und anschließend mit *cv2.stereoRectify* (5.15f) die Parameter zur Rektifizierung bestimmt. Abschließend werden mithilfe von NumPy alle ermittelten Werte als Arrays in einer .npz-Datei abgespeichert (5.15g). Bei NumPy [37] handelt es sich um eine Programmbibliothek für Python, die effizient mit großen Datenstrukturen umgeht.

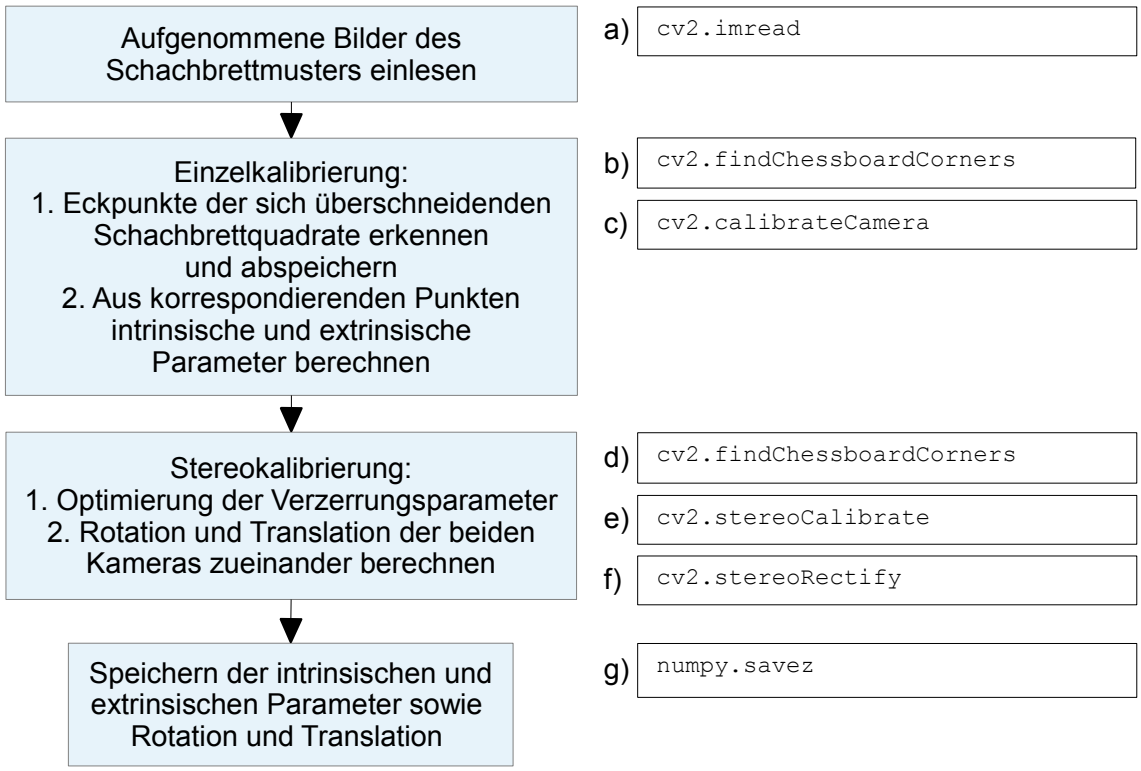


Abbildung 5.15.: Dargestellt ist das Ablaufdiagramm der Stereokalibrierung mit den wichtigsten Funktionen in OpenCV/Python.

Um die Disparitätenkarte zu erstellen, werden die Parameter zuerst wieder aus der Datei geladen (5.16a) und die zwei Aufnahmen der Stereokamera eingelesen (5.16b). Die Verzeichnungen werden daraufhin korrigiert und die Bilder rektifiziert (5.16c), sodass sie bereit für den Semi-Global Block Matching Algorithmus sind (5.16d). Um Kontrolle über die Vielzahl von verschiedenen Parametern der Funktion zu haben, können diese über Schieberegler eingestellt werden (5.16e). Disparitätenkarte dauerhaft mit den aktuell eingestellten Werten aktualisiert und angezeigt (5.16f). Nachfolgend ist das Programm als Ablaufdiagramm dargestellt.

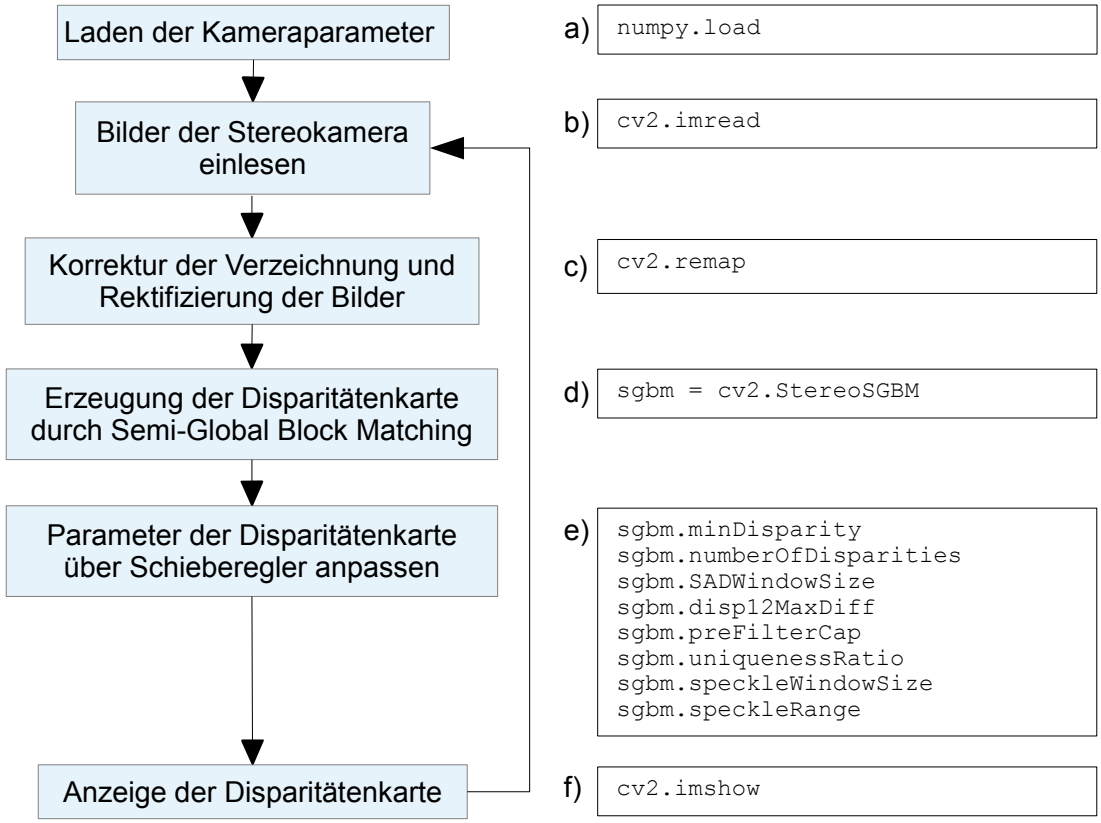


Abbildung 5.16.: Zu sehen ist das Ablaufdiagramm für die Generierung der Disparitätenkarte mit den wichtigsten Funktionen in OpenCV/Python.

5.7. Bewertung und Zusammenfassung

Die bisherigen Tests wurden mit dem in Abbildung 5.4 zu sehendem Modul durchgeführt. Um die vorhandene Konstruktion der Panoramakamera zu erweitern, wurde ein weiteres Gehäuse für vier Kameras aufgesetzt (siehe Abbildung 3.3a). Somit können je zwei Kameras in jede Richtung eingebaut werden und Stereoskopie ermöglichen

Allerdings befinden sich die Kameras dadurch nicht mehr horizontal nebeneinander, sondern vertikal übereinander. Da die SGBM-Funktion von OpenCV keine vertikale Stereokamera unterstützt, müssen die Aufnahmen in der Programmierung um 90° im Uhrzeigersinn gedreht werden, um dem Algorithmus eine horizontale Anordnung vorzutäuschen.

Die untere Kamera repräsentiert das linke Bild der Stereokamera und die Obere das rechte Bild.

(a) Untere Kamera als linkes Bild (b) Obere Kamera als rechtes Bild

Abbildung 5.17.: Die im konstruierten Gehäuse verbauten Kameras befinden sich übereinander und müssen um 90° gedreht werden, es sind rektifizierte Aufnahmen zu sehen.

Durch die Änderung der Konfiguration müssen die Kameras neu kalibriert werden, zumindest die extrinsischen Parameter habe keine Gültigkeit mehr, da sich die Rotation und Translation der Koordinatensysteme geändert hat. Die Basislinie zwischen den Kameras ist nun 40 mm lang. In diesem Konstellation rektifizierte Aufnahmen sind in Abbildung 5.17 zu sehen. Die Markierung P zeigt auf die Vorderseite des Papierkorbs und stellt den nächsten Punkt zur Kamera dar. Um diesen geht es insbesondere im nächsten Kapitel.

Die mit diesem Aufbau erzeugte Disparitätenkarte ist in Abbildung 5.14 zu sehen und wurde mit den Parameterwerten in Tabelle 5.19 erreicht. Im Unterschied zum vorherigen Versuch war es diesmal möglich, alle Bildfehler zu filtern. Dabei muss erwähnt werden, dass mehrere Raspberry Pi Kameras zu Verfügung standen und das optimale Paar ermittelt wurde. Andere Kombinationen führten zu noch mehr Fehlern als in der Disparitätenkarte

in Abbildung 5.14. Es handelt sich um Low-Cost-Kameras, die in der Verarbeitung stark variieren. Auch sind in dieser Szene keine extremen Kontraste zu finden, die der Hauptgrund für Speckles sind. Wenn große Flächen der gleichen Textur auftreten (in der Abbildung der gelbe Boden), können keine Distanzen ermittelt werden, was sich durch Schwarzwerte in der Karte äußert.

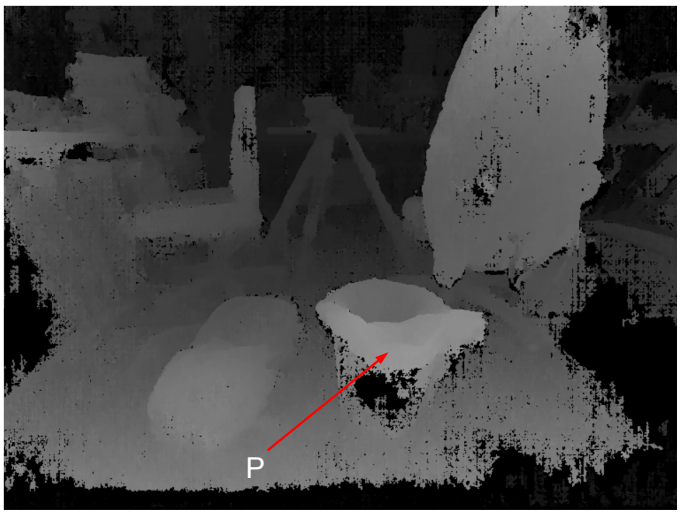


Abbildung 5.18.: Aus den Bildern in Abbildung 5.17 erstellte Disparitätenkarte. Markiert ist wieder der Papierkorb P, in dessen Bereich die hellsten Grauwerte also die größten Disparitäten der Karte sind. Dies stimmt damit überein, dass es das Objekt mit der geringsten Entfernung zur Kamera ist.

Parameter	**Werte** 5.14	**Werte** 5.17b
minDisparity	6	1
numberOfDisparities	128	80
SADWindowSize	3	3
disp12MaxDiff	1	7
preFilterCap	5	5
uniquenessRatio	3	7
speckleWindowSize	286	386
speckleRange	12	4

Abbildung 5.19.: Die Tabelle listet die Werte auf, mit denen die Disparitätenkarten von den Abbildungen 5.14 und 5.17b erstellt wurden.

6. 3D-Rekonstruktion und Registrierung einer drohenden Kollision

Dieses Kapitel beschäftigt sich mit der 3D-Rekonstruktion in Form von Punktwolken. Ziel ist die Erkennung von Objekten in einer Punktwolke um Kollisionen in der unmittelbaren Umgebung des Kamerasystems zu vermeiden. Der Abschnitt 6.1 befasst sich mit der Erzeugung einer Punktwolke auf Basis der Disparitätenkarte, die im Mittelpunkt des vorherigen Kapitels 5 stand. Da Punktwolken große Datenmengen enthalten, wird daraufhin in Abschnitt 6.2 Octrees eingeführt, mit denen dreidimensionale Datensätze komprimiert werden können. Anschließend wird in Abschnitt 6.3 mithilfe von Octrees eine Kollisionserkennung programmiert und als letztes die Ergebnisse zusammengefasst (Abschnitt **??**).

6.1. Generierung einer Punktwolke

Um die Umgebung zu rekonstruieren und Entfernungen berechnen zu können, soll aus den gesammelten Daten eine Punktwolke erstellt werden. Eine Punktwolke ist eine Menge von gemessenen Punkten in einem Koordinatensystem, die eine dreidimensionalen Raumstruktur ergeben. Vor allem im Bereich des 3D-Laserscanning ist es üblich, mit Punktwolken zu arbeiten. Dabei wird zum Beispiel die Oberfläche eines Objektes abgetastet und die Geometrie in einer Wolke von Punkten gespeichert. Neben den Koordinaten können einem Punkt auch weitere Datensätze zugeordnet werden, wie Messgenauigkeiten oder Farbwerte.

Die Erstellung der Wolke erfolgt im Weiteren durch OpenCV und knüpft an die Ergebnisse aus Kapitel 5 an. In der Stereoskopie ist die Grundlage für die 3D-Rekonstruktion die Dis-

paritätenkarte. Zudem wird die Reprojektionsmatrix $\mathbf{Q}$ der Stereokamera benötigt, die aus den intrinsischen und extrinsischen Parametern der Kamerakalibrierung gebildet wird.

Neben der Brennweite f, wird der Wert von T_x benötigt. Dabei handelt es sich um die X-Koordinate des Translationsvektors, der für die Überführung des rechten ins linke Kamerakoordinatensystems verantwortlich ist. Vom linken Bild werden die Abweichungen des Bildhauptpunktes vom Zentrum horizontal c_x sowie vertikal c_y verwendet. Lediglich die Abweichung des Hauptpunktes in X-Richtung c'_x wird von den Parametern der rechten Kamera verlangt [3].

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/T_x & (c_x - c'_x)/T_x \end{bmatrix} \tag{6.1}$$

Mit der Projektionsmatrix kann ein zweidimensionaler Punkt (x,y) daraufhin in 3D-Koordinaten transformiert werden. Nötig ist neben den Koordinaten die Disparität d an dieser Position (siehe 5.5):

$$\mathbf{Q} \cdot \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \tag{6.2}$$

Um die Punktwolke zusätzlich mit Farbinformationen zu versehen, werden die RGB-Werte des linken rektifizierten Bildes mit den Koordinaten in Verbindung gesetzt. Bei der Kalibrierung wurde das rechte ins linke Kamerakoordinatensystem überführt und somit richtet sich auch die Disparitätenkarte nach diesem.

In OpenCV wird die Transformation aus Gleichung 6.2 für jede Koordinate von der Funktion *cv2.reprojectImageTo3D* berechnet. Die Wolke wird anschließend im ASCII-Format als PLY-Datei (Polygon File Format) gespeichert, wobei die Funktion *writePlyFile* der OpenCV-Dokumentation entnommen wurde [26]. PLY ist ein Standard-Format für Punktwolken, das aus dem Laserscannig kommt und von den gängigsten Programmen verarbeitet werden. In

dieser Arbeit kam CloudCompare zum Einsatz [6]. Für die Erstellung der Disparitätenkarte und Punktwolke wurden jeweils die Zeiten gemessen (siehe Abbildung 6.3).

Um den Prozess von zwei Bildern zur Punktwolke nachvollziehen zu können, wurde im Weiteren das Beispiel aus Abbildung 5.17 im vorherigen Kapitel verwendet. Dabei wurde das Objekt (P) hervorgehoben, zu dem die Kamera die geringste Entfernung aufweist. Ziel ist es, dieses zu erkennen.

Daraufhin wurde aus der Disparitätenkarte in Abbildung 5.18 die Punktwolke in Abbildung 6.1 erstellt, diese umfasst 1.650.660. Die PLY-Datei ist 64,912 KB groß. Die Markierung a) in der Abbildung verweist auf den relevanten Ausschnitt, der den Teil darstellt der in der Disparitätenkarte zu sehen ist. Bei den restlichen Punkten handelt es sich um Fehlmessungen. Wenn in der Karte zum Beispiel keine bis wenig Textur erkannt wurde, werden diese Punkten in sehr dunklen Grautönen bis hin zu schwarz dargestellt. Folglich wird der Punkt von OpenCV weit entfernt angenommen.

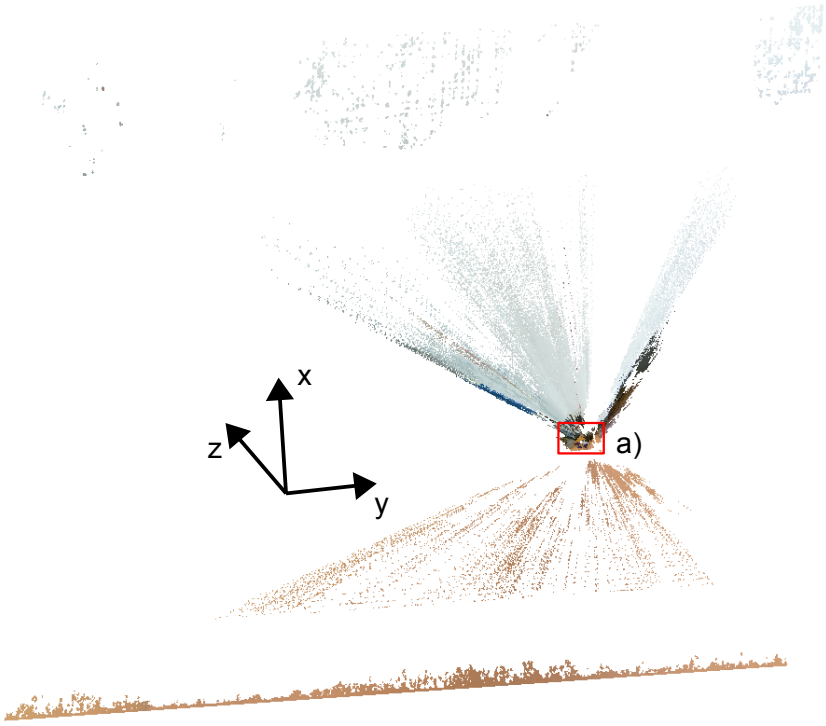
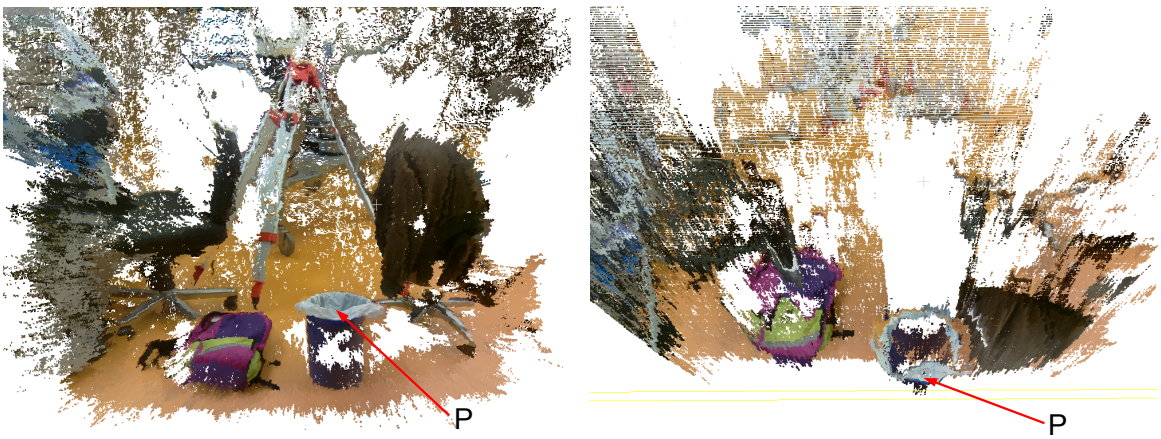


Abbildung 6.1.: Ansicht der Punktwolke in CloudCompare. Der Papierkorb (P) wurde markiert.

In Bezug auf die Einheit der Koordinaten wurde beim Vergleich mit den Distanzen im Raum festgestellt, dass die Werte um den Faktor 10 zu niedrig sind.



(a) (b)

Abbildung 6.2.: Zu sehen ist der relevante Ausschnitt aus der Punktwolke. Wie im vorherigen Kapitel ist das zur Kamera nächste Objekt mit P gekennzeichnet.

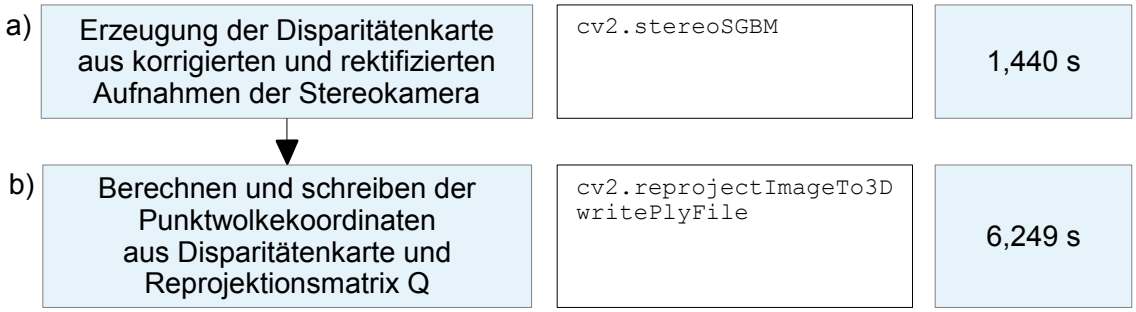


Abbildung 6.3.: Dieser Ablauf basiert auf dem Diagramm 5.16, der um die dargestellten Funktionen erweitert wird. Zudem wurden die Zeiten der jeweiligen Abläufe auf der Workstation gemessen.

6.2. Komprimierung einer Punktwolke durch Octree

Da die Datenmenge der Punktwolke sehr groß ist und es in Bezug auf eine Hinderniserkennung dementsprechend rechen- und speicherintensiv ist, Objekte zu detektieren, soll

Octree verwendet werden. Es handelt sich dabei um eine dreidimensionale Datenstruktur in der Datensätze hierarchisch untergliedert werden [40]. Der Datenraum der Punktwolke wird hierbei in gleichmäßige Würfel eingeteilt. In der obersten Ebene umgibt ein Würfel den gesamten Datenraum, in der nachfolgenden Ebene wird er in acht Würfel unterteilt. Die entstandenen Würfel werden in der darauf folgenden Ebene wiederum in acht kleinere Würfel zerteilt. Bei diesem Prozess nähert sich die Gesammtzahl der Würfel immer weiter der Gesamtzahl der Punkte in der Punktwolke an. Die Würfel werden also so lange zerteilt bis keine weitere Teilung mehr möglich ist, also nur noch eine Koordinate pro Würfel besteht. Jedoch ist es nicht notwendig in die unterste Ebene des Octrees vorzudringen. Um ein Objekt in der Punktwolke zu erkennen, ist es zum Beispiel nicht wichtig jeden einzelnen Punkt zu verarbeiten. Um die Entfernung zum Objekt zu bestimmen, muss keine koordinatengenaue Ortung erfolgen. Hierdurch ergibt sich ein Performancegewinn.

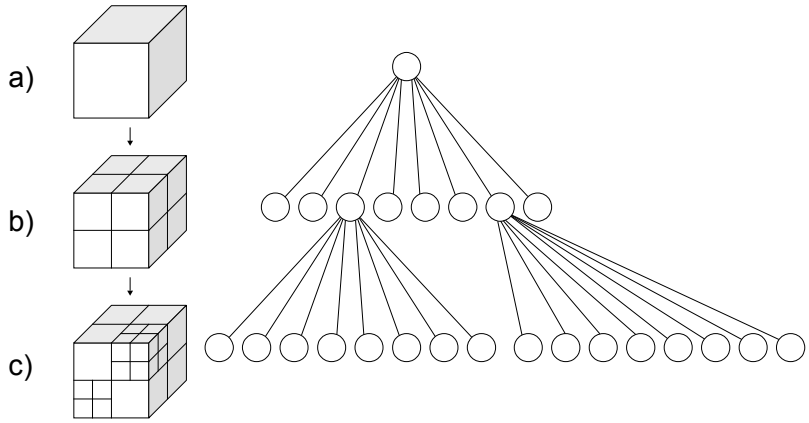


Abbildung 6.4.: Die Abbildung zeigt hierarchische Untergliederung auf der Octree basiert (Quelle: [38])

6.3. Programmierung der Kollisionserkennung

Prädestiniert für den Umgang mit Punktwolken ist die Point Cloud Library [34]. Die frei zur Verfügung stehende Programmbibliothek bietet eine Vielzahl von Algorithmen zur Verarbeitung, wie diverse Filter und Funktionen zur Segmentierung. Die gewünschte Komprimierung mittels der Octree-Struktur wird durch OctoMap [14] ermöglicht. Die ebenfalls

freie Bibliothek kann mit der Point cloud Library kombiniert werden, um sie somit für Octree zu optimieren.

Auf dieser Basis wurde ein Programm geschrieben, dass die erzeugte Punktwolke in eine Octree Struktur umwandelt. In dieser Struktur wird das Objekt mit der geringsten Distanz zur Kamera ermittelt. Dabei wird zuerst die PLY-Datei geladen und durchläuft den Passthrough-Filter der PCL. Dieser legt einen gültigen Wertebereich fest. Punkte mit davon abweichenden Werten werden aus der Z-Achse entfernt, da davon auszugehen ist, dass es sich bei ihnen um Fehlmessungen handelt.

Mithilfe vom Robot Operating System [33] wurde die Entfernungsmessung realisiert. Dabei wird die *DynamicEDT3D* Klasse aufgerufen, mit der es möglich ist, eine 3D-Distanzkarte zu erstellen. Diese enthält eine Funktion, um die Distanz zu dem nächsten Objekt von einem definierten Punkt im Octree zu ermitteln. Bei Angabe des Punktes (0,0,0) wird die Distanz zur Kamera und die Position des Objektes bestimmt.

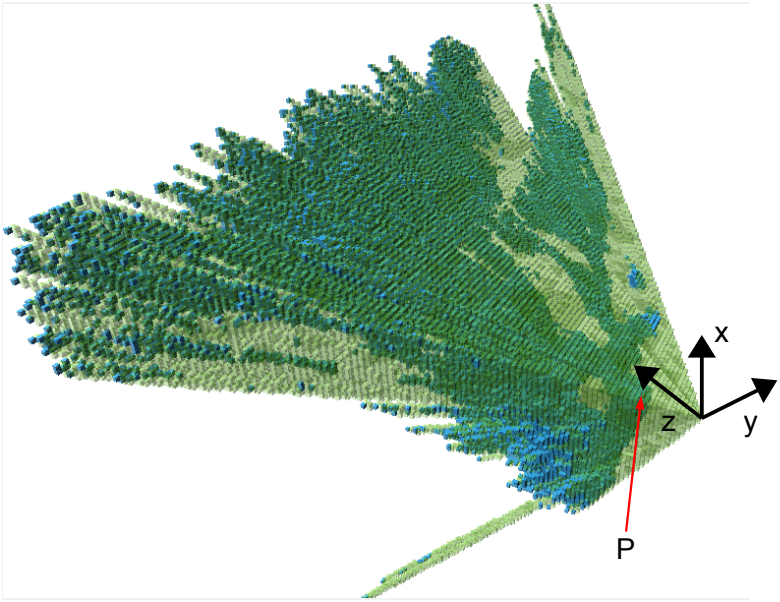


Abbildung 6.5.: Dargestellt ist die Octree-Struktur der beschnittenen Punktwolke. Wieder ist der Papierkorb, an dem er Prozess vom rektifizierten Bild bis zum Octree nachvollzogen werden kann.

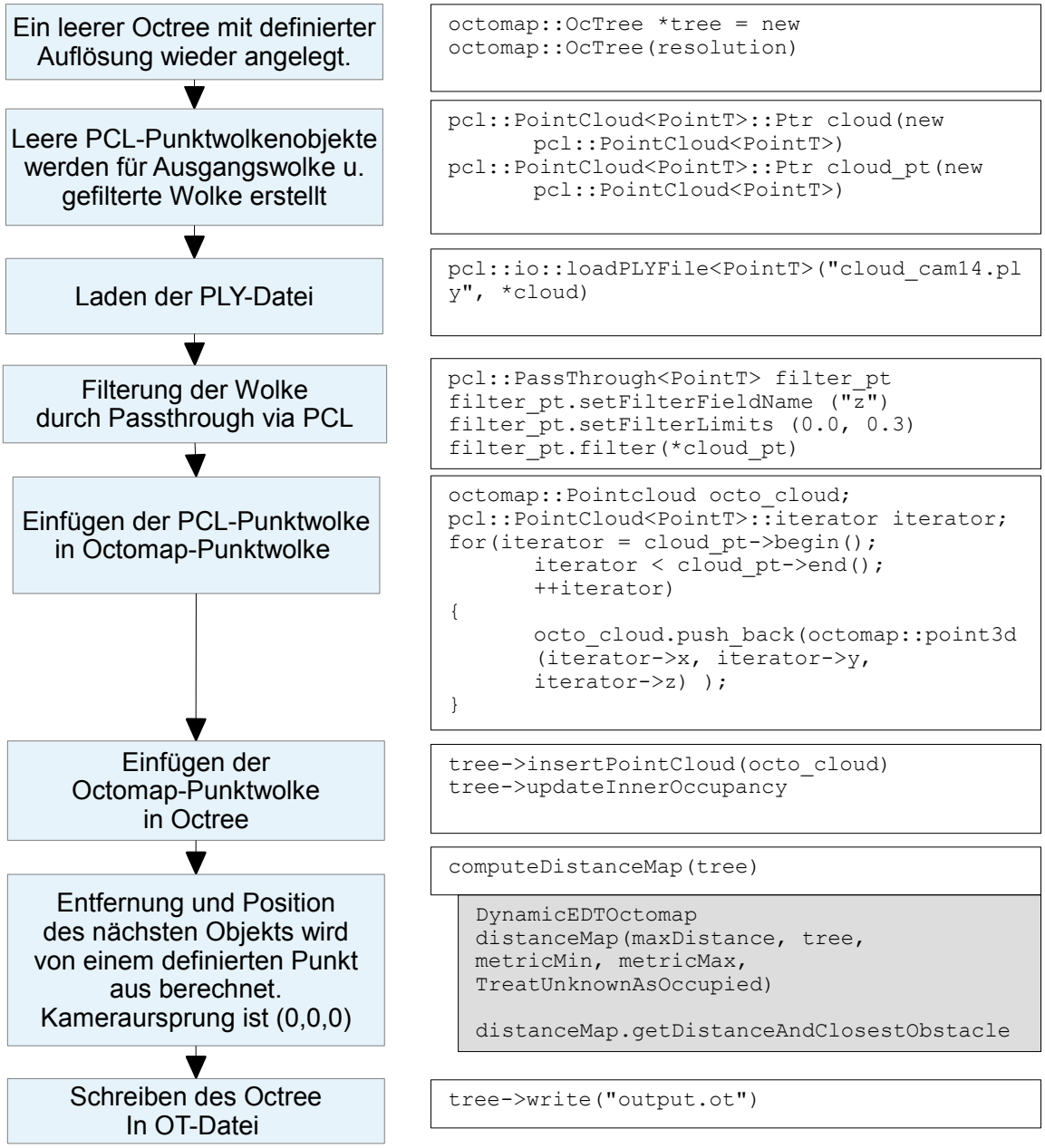


Abbildung 6.6.: Ablaufdiagramm der Kollisionserkennung in C++ in Verbindung mit PCL und Octomap

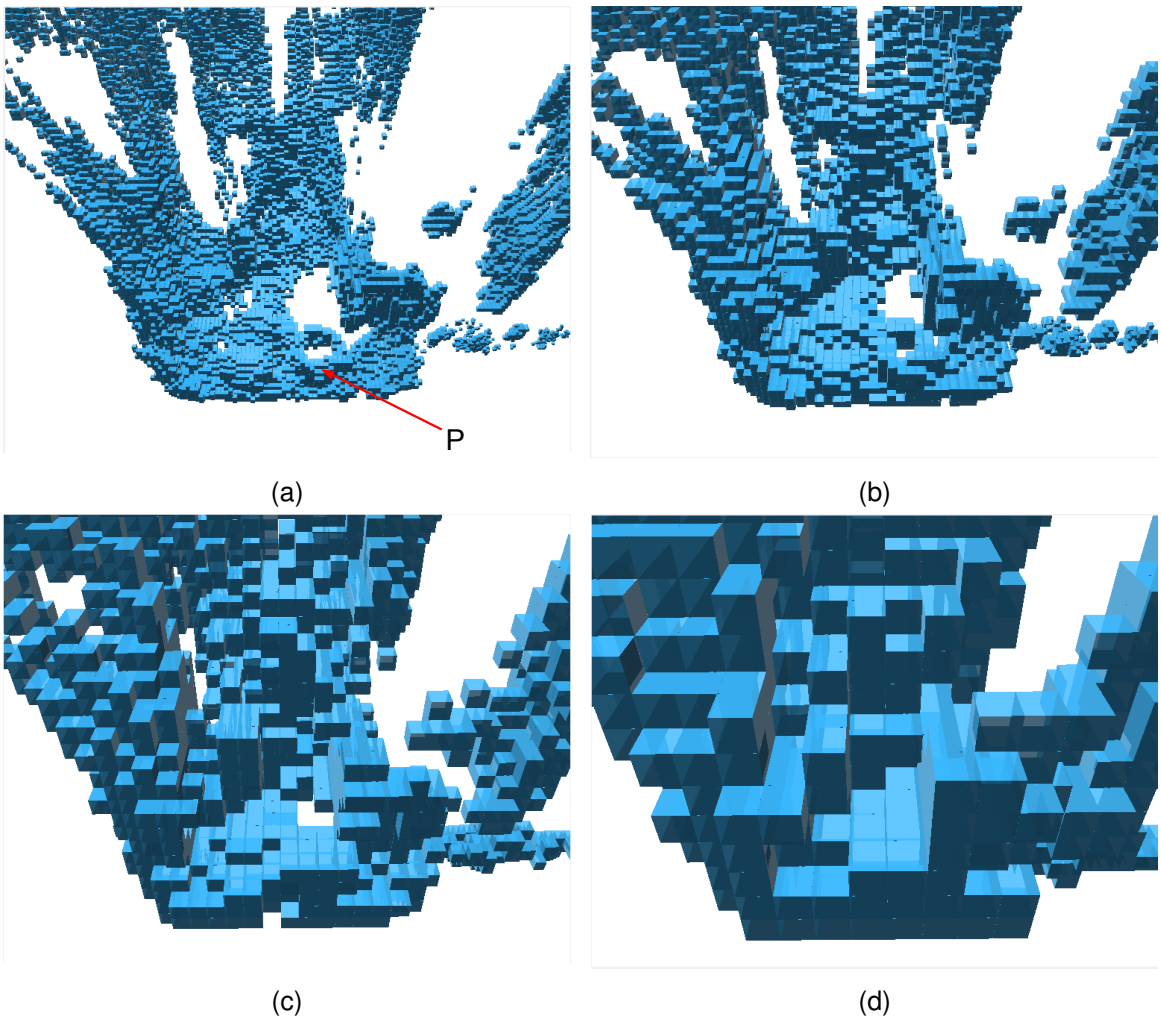


Abbildung 6.7.: Zu sehen ist der aus der Punktwolke 6.1 erstellte Octree in verschiedenen Ebenen. Der bereits in fig:last markierte Punkt ist zur Veranschaulichung markiert.

7. Zusammenfassung und Ausblick

Das Ziel der vorliegenden Arbeit war die Entwicklung eines Kamerasystems zur Kollisionsvermeidung in der mobilen Robotik. Es wurde aus vier Kameras ein Panoramabild erstellt um visuell mögliche Hindernisse zu registrieren. Dabei wurde festgestellt, dass der Parallaxenfehler entsteht, wenn Aufnahmen nicht von der gleichen Position aus gemacht wurden. Dieser führt bei den Übergängen zwischen den Bildern zu optischen Fehlern. Trotz dessen kann durch die Kamera ein Sichtfeld von 360° erzeugt und mit dessen Hilfe die Navigation eines mobilen Roboters unterstützt werden.

Eine Verbesserungsmöglichkeit stellt die Implementierung eines Blendings zwischen den einzelnen Bildern des Panoramas dar. Dabei werden die Bereiche der Bilder, die den gleichen Bildausschnitt zeigen überlagert. Dadurch können Helligkeitsunterschiede und Farbverläufe angeglichen werden.

Zwar handelt es sich bei der angefertigten Konstruktion um einen Prototypen, aber auch dieser ist aufgrund seiner kompakten Bauweise theoretisch schon für mobile Roboter geeignet. Zusätzlich setzt sich das System aus kostengünstigen Elementen zusammen, sowohl im Hinblick auf die Hardware, als auch auf die angefertigte Konstruktion. Diese ist durch einen 3D-Drucker und Lasercutter entstanden.

Die Panoramakamera wurde in jede Richtung durch eine Kamera ergänzt, so dass durch Stereoskopie ein Tiefenbild der Umgebung erstellt werden konnte. Es wurde festgestellt, dass die Ergebnisse einer Stereokamera maßgeblich von der Qualität und Kalibrierung der Kameras sowie von der beobachteten Umgebung abhängen. Starke Kontraste zwischen Hell und Dunkel sowie keine oder wenig Textur, also einfarbige Flächen, führen zu Speckles. Auch an Objektkanten können wegen hohen Distanzunterschieden Fehler auftreten.

Unter optimalen Bedingungen kann eine qualitativ hochwertige Disparitätenkarte generiert werden. Aus dieser kann im Anschluss eine Punktwolke erstellt werden.

Um die Datenmenge zu minimieren wurde die Punktwolke in eine Octree-Struktur umgewandelt. In dieser wurde die Entfernung zum Objekt mit der geringsten Distanz berechnet, was als Basis für eine Kollisionserkennung dienen kann.

Die Erstellung der Disparitätenkarte sowie der Punktwolke nimmt viel Zeit in Anspruch, wodurch es in der Praxis durch die verzögerte Verarbeitung zu Problemen in der Kollisionserkennung kommen kann. Es ist zu erwarten, dass der Ablauf durch eine Implementierung in C++ in der Geschwindigkeit erhöht werden kann. Auch sollte in Erwägung gezogen werden, den Raspberry Pi durch einen leistungsfähigeren Computer zu ersetzen.

Literaturverzeichnis

[1] BOURKE, P. : *Computer Generated Angular Fisheye Projections*. http://paulbourke.net/dome/fisheye/. Version: 2001. – Zuletzt aufgerufen am 24.11.2017

[2] BOURKE, P. : *Converting dual fisheye images into a spherical (equirectangular) projection*. http://paulbourke.net/dome/dualfish2sphere/. Version: 2016. – Zuletzt aufgerufen am 12.12.2017

[3] BRADSKI, G. : The OpenCV Library. In: *Dr. Dobb's Journal of Software Tools* (2000). https://opencv.org/

[4] BRADSKI, G. ; KAEHLER, A. : *Learning OpenCV*. O'Reilly Media, 2008. – ISBN 978–0–596–51613–0

[5] CLAUS, D. ; FITZGIBBON, A. W.: A Rational Function Lens Distortion Model for General Cameras. In: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*. IEEE Computer Society (CVPR '05). – ISBN 0–7695–2372–2, 213–219

[6] CLOUDCOMPARE: *3D point cloud and mesh processing software*. http://www.danielgm.net/cc/

[7] CORKE, P. : *Robotics, Vision and Control*. Springer, 2013. – ISBN 978–3–642–20143–1

[8] CRATES: *World map blank without borders*. https://commons.wikimedia.org/wiki/File:World_map_blank_without_borders.svg. Version: 2009. – Zuletzt aufgerufen am 24.11.2017

[9] DASSAULT SYSTEMES: *CATIA V5*. https://www.3ds.com/products-services/catia/products/v5/portfolio/

[10] FANTAGU: *Verzeichnung3*. https://commons.wikimedia.org/wiki/File:Verzeichnung3.png. Version: 2009. – Zuletzt aufgerufen am 24.11.2017

[11] FISCHLER, M. A. ; BOLLES, R. C.: Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In: *Commun. ACM* 24 (1981), Jun., Nr. 6, 381–395. http://dx.doi.org/10.1145/358669.358692. – DOI 10.1145/358669.358692. – ISSN 0001–0782

[12] FLESHGRINDER: *Blank globe*. https://commons.wikimedia.org/wiki/File:Blank_globe.svg. Version: 2009. – Zuletzt aufgerufen am 24.11.2017

[13] HIRSCHMULLER, H. : Stereo Processing by Semiglobal Matching and Mutual Information. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 30 (2008), Febr., Nr. 2, 328–341. http://dx.doi.org/10.1109/TPAMI.2007.1166. – DOI 10.1109/TPAMI.2007.1166. – ISSN 0162–8828

[14] HORNUNG, A. ; WURM, K. M. ; BENNEWITZ, M. ; STACHNISS, C. ; BURGARD, W. : OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. In: *Autonomous Robots* (2013). http://dx.doi.org/10.1007/s10514-012-9321-0. – DOI 10.1007/s10514–012–9321–0. – Software available at http://octomap.github.com

[15] IVMECH: *IVPort Raspberry Pi Camera Module Multiplexer*. http://www.ivmech.com/magaza/en/development-modules-c-4/ivport-raspberry-pi-camera-module-multiplexer-p-90

[16] JÄHNE, B. : *Digitale Bildverarbeitung und Bildgewinnung*. Springer Vieweg. http://dx.doi.org/10.1007/978-3-642-04952-1. http://dx.doi.org/10.1007/978-3-642-04952-1. – ISBN 978–3–642–04951–4

[17] JOSHI, P. : *OpenCV with Python By Example*. Packt Publishing Ltd., 2015. – ISBN 978–1–78528–393–2

[18] LOWE, D. G.: Distinctive Image Features from Scale-Invariant Keypoints. In: *International Journal of Computer Vision* 60 (2004), S. 91–110

[19] MAKERBOT: *Replicator 2X Experimental 3D Printer*. https://store.makerbot.com/printers/replicator2x/

[20] MARCIN LIGAS, P. B.: *Conversion between Cartesian and geodetic coordinates on a rotational ellipsoid by solving a system of nonlinear equations*. http://www.iag-aig.org/attach/989c8e501d9c5b5e2736955baf2632f5/V60N2_5FT.pdf. Version: 2011. – Zuletzt aufgerufen am 25.12.2017

[21] MATHWORKS: *Lineare Abbildungsmethode durch affine Abbildung*. https://de.mathworks.com/discovery/affine-abbildung.html. – Zuletzt aufgerufen am 15.12.2017

[22] MATTSSON, J. : *Approximation of Texture Coordinates*. https://amycoders.org/tutorials/tm_approx.html. – Zuletzt aufgerufen am 20.12.2017

[23] MINICHINO, J. ; HOWSE, J. : *Learning OpenCV 3 Computer Vision with Python*. Packt Publishing, 2015. – ISBN 978–1–78528–384–0

[24] MUJA, M. ; LOWE, D. G.: Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. In: *International Conference on Computer Vision Theory and Application VISSAPP'09)*, INSTICC Press, 2009, S. 331–340

[25] OMNIVISION: *OV5647 Sensor Datenblatt*. http://cdn.sparkfun.com/datasheets/Dev/RaspberryPi/ov5647_full.pdf. – Zuletzt aufgerufen am

[26] OPENCV: *Simple example of stereo image matching and point cloud generation*. https://github.com/opencv/opencv/blob/master/samples/python/stereo_match.py. – Zuletzt aufgerufen am 21.12.2017

[27] OPENCV DEV TEAM: *Pinhole camera model*. https://docs.opencv.org/2.4/_images/pinhole_camera_model.png. – Zuletzt aufgerufen am 24.11.2017

[28] PAPULA, L. : *Mathematik für Ingenieure und Naturwissenschaftler Band 3*. Springer, 2016. – ISBN 978–3–658–11923–2

[29] PTGUI: *Image stitching software*. https://ptgui.com/

[30] PYTHON(X,Y): *The scientific Python distribution*. https://python-xy.github.io/

[31] RASPBERRY PI FOUNDATION: *Website*. https://raspberrypi.org/

[32] REHM, P. D.-I. W.: *Funktionen von 3D-Grafikkarten*. https://www.tu-chemnitz.de/informatik/RA/news/stack/kompendium/vortr_2000/3d_fkt/index.htm. Version: 2000. – Zuletzt aufgerufen am 30.12.2017

[33] ROS: *exampleEDTOctomap.cpp*. http://docs.ros.org/jade/api/dynamic_edt_3d/html/exampleEDTOctomap_8cpp_source.html. – Zuletzt aufgerufen am 3.1.2018

[34] RUSU, R. B. ; COUSINS, S. : 3D is here: Point Cloud Library (PCL). In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE Computer Society, 1-4

[35] SCHLECHTWEG, P. S.: *Computergraphik Grundlagen - IV. Koordinatensysteme und geometrische Transformationen*. http://www.mttcs.org/Skripte/Pra/Material/vorlesung3.pdf. – Zuletzt aufgerufen am 2.12.2017

[36] TIGHTVNC: *Remote Control / Remote Desktop Software*. http://tightvnc.com/

[37] WALT, S. v. d. ; COLBERT, S. C. ; VAROQUAUX, G. : The NumPy Array: A Structure for Efficient Numerical Computation. In: *Computing in Science and Engg.* 13 (2011), März, Nr. 2, 22–30. http://dx.doi.org/10.1109/MCSE.2011.37. – DOI 10.1109/MCSE.2011.37. – ISSN 1521–9615

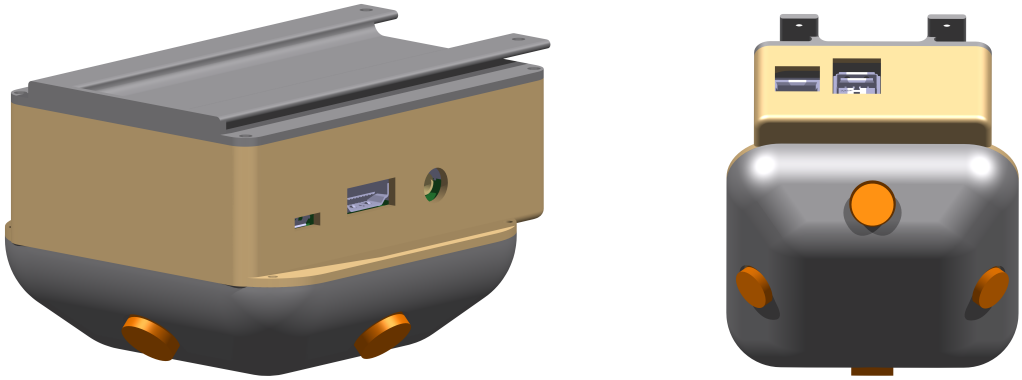
[38] WHITETIMBERWOLF: *Octree2*. https://upload.wikimedia.org/wikipedia/commons/2/20/Octree2.svg. Version: 2010. – Zuletzt aufgerufen am 24.11.2017

[39] WIKIPEDIA: *Maschinelles Sehen*. https://de.wikipedia.org/wiki/Maschinelles_Sehen. – Zuletzt aufgerufen am 30.12.2017

[40] WIKIPEDIA: *Octree*. https://de.wikipedia.org/wiki/Octree. – Zuletzt aufgerufen am 17.12.2017

A. Testergebnisse der alternativen Konstruktion

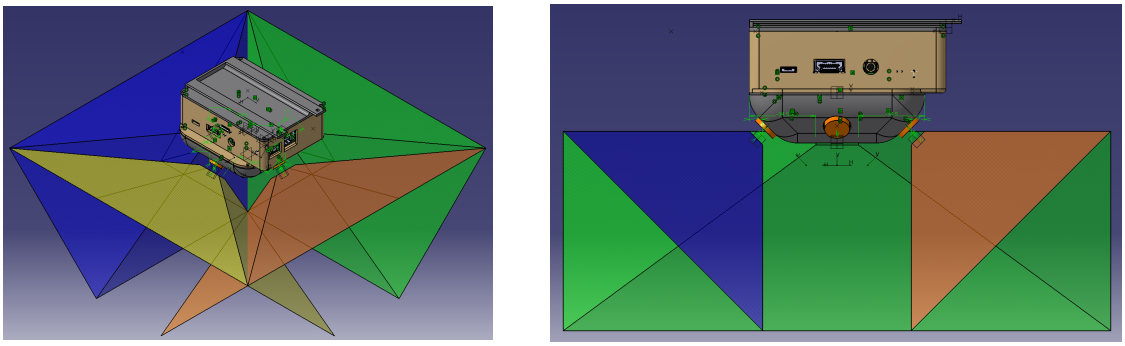
In einem vorherigen Projekt wurde vom Autor dieser Arbeit bereits eine Panoramakamera entwickelt, konstruiert und durch Rapid Prototyping gefertigt (siehe Abbildung A.1). Die Programmierung, um Bilder zusammenzuführen, blieb jedoch aus. Vier Kameras sind in diesem Fall horizontal in einem Winkel von 90° zueinander verbaut, jede Kamera ist zusätzlich vertikal um 45° geneigt. Es wurden die gleichen Raspberry Pi Kameras wie in dieser Arbeit verwendet. Das Sichtfeld ist dementsprechend horizontal ca. 125° und vertikal ca. 90° (siehe Abbildung A.2).



(a) (b)

Abbildung A.1.: Die in Catia erstellte CAD-Konstruktion als gerendertes Modell.

Am Anfang dieser Arbeit wurde versucht, mit der vorhandenen Konstruktion, ein Panorama zu erstellen. In Abbildung A.3a sind zwei Originalaufnahmen mit starker Verzeichnung dargestellt, die per Homographie zusammengefügt wurden (Programmierung siehe Ablaufdiagramm 4.8). Ein etwas besseres Ergebnis wurden in Abbildung A.3b mit zwei in

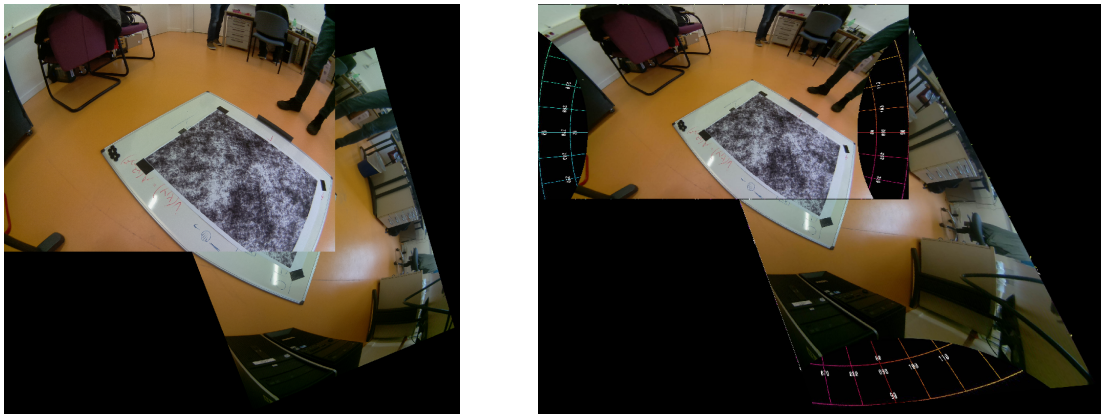


(a) (b)

Abbildung A.2.: Darstellung der Sichtfelder der vier Kameras. In Abbildung A.2b ist der tote Winkel unterhalb der Kamera zu sehen.

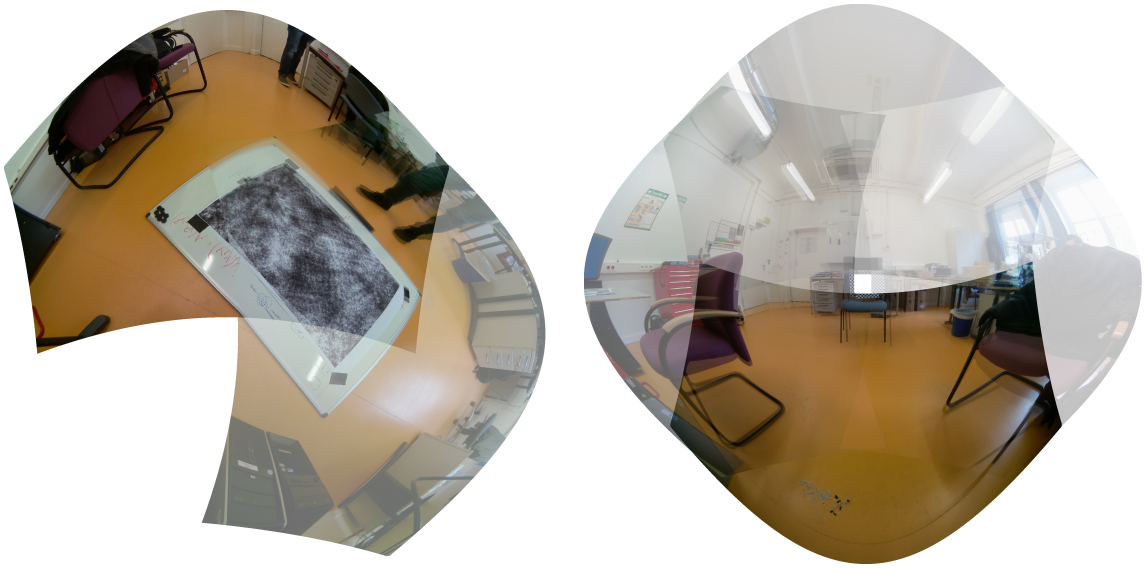
eine Sphäre abgebildeten (siehe Abschnitt 4.1) Aufnahmen erzielt. Doch am Resultat ist vorherzusehen, dass mit einer dritten und vierten Kamera kein vollständiges Panorama entstehen würde.

Außerdem wurde versucht, die Kamerabilder statisch zusammenzuführen. Dabei wurden die Bilder wieder in eine hohle Kugel projiziert und diesmal zusätzlich der Winkel φ, also der geographische Breitengrad der Sphäre, verändert. Im Anschluss wurden die Bilder nicht per Stitching automatisch zusammmengeführt, sondern an festen Positionen in ein Gesamtbild eingefügt. Die Positionen wurden von Hand definiert. In Abbildung A.4a für zwei Kameras und in Abbildung A.4b für vier Kameras ist zu sehen, dass befriedigende Ergebnisse erreicht wurden. Dies gilt allerdings immer nur für bestimmte Entfernungen zur Umgebung. Ändert sich die betrachtete Szene, müssen die Positionen im Gesamtbild neu definiert werden. Ursache für dieses Problem ist der Parallaxenfehler (siehe Abschnitt 4.3).



(a) Unveränderte Kameraaufnahmen (b) In Sphäre projizierte Aufnahmen

Abbildung A.3.: Zu sehen sind jeweils Versuche, zwei Kamerabilder dynamisch per Stitching zusammenzuführen.



(a) Aufnahmen von zwei Kameras (b) Aufnahmen von vier Kameras

Abbildung A.4.: Darstellung der statischen Zusammenführung der Kamerabilder

B. Inhalt der beiliegenden CD

- Bachelorthesis als PDF-Datei
- Quelltexte der erstellten Programme
- In CATIA konstruierte CAD-Modelle

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 19. Januar 2018	
Ort, Datum	Unterschrift