Bachelorarbeit

Martin Schuhfuß

Realisierung eines Laser-Triangulationssensors zur 3D-Objektvermessung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Martin Schuhfuß

Realisierung eines Laser-Triangulationssensors zur 3D-Objektvermessung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Andreas Meisel
Zweitgutachter: Prof. Dr. rer. nat. Gunter Klemke

Abgegeben am 5. Oktober 2007

Martin Schuhfuß

Thema der Bachelorarbeit
Realisierung eines Laser-Triangulationssensors zur 3D-Objektvermessung
Stichworte
Computer Vision, Laser-Triangulation, Laser-Scanner, 3D Vermessung, Kalibrierung, perspektivische Verzerrung, Transformation, lineare Algebra, homogene Koordinaten
Kurzzusammenfassung
In dieser Arbeit wird die Grundlage für die Realisierung eines 3D Laserscanners beschrieben. Es werden die mathematischen und bildverarbeitungstechnischen Grundlagen der Lasertriangulation und der perspektivischen Abbildungen erläutert und Ansätze für die Kalibrierung eines solchen Systems und die 2-Dimensionale Vermessung einer Objektoberfläche beschrieben und implementiert.

Martin Schuhfuß

Title of the paper
Development of a Laser-Triangulation sensor for 3D-Scanning
Keywords
Computer Vision, Laser-Triangulation, 3D-Scanning, calibration, perspective transformation, linear algebra, homogenous coordinates, least squares approximation
Abstract
This bachelor-thesis will cover the basis for the realisation of a 3D laser-scanner. It will illustrate the basic principles of laser-triangulation and perspective transformations and the corresponding fundamentals in mathematics and image-processing. Furthermore an approach for the calibration of this system and the two-dimensional survey of an object surface will be described and implemented.

Inhaltsverzeichnis

Abbildungsverzeichnis **1**

1. Einleitung **2**
1.1. Verfahren und Anwendungen der 3D-Messtechnik . . . 3
1.2. Verwandte Technologien . . . 4
1.3. Zielsetzung dieser Arbeit . . . 5
1.4. Gliederung . . . 5

2. Grundlagen **6**
2.1. Mathemathische Grundlagen . . . 6
2.1.1. Bildtransformationen . . . 6
2.1.2. Lösung linearer Ausgleichsprobleme . . . 8
2.2. Grundlagen der Bildverarbeitung . . . 10
2.2.1. Lineare Filterung mit Faltungskernen . . . 10
2.2.2. Kantendetektion . . . 11
2.2.3. Hough-Raum und Hough-Transformation . . . 13
2.2.4. Bilineare Interpolation . . . 14

3. Technischer Aufbau **16**
3.1. Der Grundrahmen . . . 16
3.2. Laser- und Kamera-Träger . . . 16
3.3. Kalibriervorrichtung . . . 17
3.4. Spezifikationen von Laser und Kamera . . . 17

4. Konzeption **19**
4.1. Kalibrierung . . . 19
4.1.1. Vorbereitung . . . 20
4.1.2. Bestimmung der Referenzpunkte in der Bildebene . . . 22
4.1.3. Berechnung der Transformationsmatrix . . . 28
4.2. Objektvermessung . . . 30
4.2.1. Vorbereitung . . . 30
4.2.2. Vorverarbeitung des Kamerabildes . . . 30
4.2.3. Extraktion der Oberflächenpunkte aus dem Bild . . . 31
4.2.4. Berechnung der Koordinaten in der Laserebene . . . 32

5. Messergebnisse **34**
5.1. Genauigkeit der Kalibrierung 34
5.2. Messgenauigkeit 35

6. Implementierung **37**
6.1. Struktur 37
6.2. Verwendung der Software 38
6.2.1. Durchführung einer Kalibrierung 38
6.2.2. Durchführung einer Objektvermessung 39
6.3. Verwendete Software-Bibliotheken 40
6.3.1. LTILib 40
6.3.2. Boost C++ Libraries 40
6.4. Kompilieren des Projektes 41
6.4.1. Linux 42
6.4.2. Windows 42

7. Fazit **43**

A. Quelltexte **45**
A.1. Dateien im Quelltextverzeichnis 45
A.2. Interfaces 46
A.3. Aufruf der Kalibrierung und Objektvermessung 48

B. Verwendete Hilfsmittel **50**
B.1. Software-Entwicklung 50
B.1.1. Entwicklungswerkzeuge 50
B.1.2. Software-Bibliotheken 50
B.2. Ausarbeitung 50

C. Inhalt der beiligenden CD **51**

Literaturverzeichnis **52**

Abbildungsverzeichnis

1.1. Der Large Statue Scanner der Firma Cyberware 2
1.2. Ein Pioneer III Roboter mit 2 SICK-Laserscannern 3
1.3. Triangulationsmessung mit einem Linienlaser 4

2.1. Perspektivische Transformation 8
2.2. Veranschaulichung der linearen Filterung 11
2.3. Sobel-Faltungskerne 12
2.4. Beispiel zur Kantendetektion 13
2.5. Veranschaulichung der Hough-Transformation 14
2.6. Bilineare Interpolation 15

3.1. Der Grundrahmen 17
3.2. Laser- und Kameraträger 18
3.3. Kalibrierungsvorrichtung 18

4.1. Das verwendete Kalibrierungsmuster 20
4.2. Ausrichtung des Lasers 21
4.3. Kalibrierungsbild mit *Region of interest* 24
4.4. Kalibrierungsbild mit näherungsweise bestimmten Kantengeraden 25
4.5. Beispiele für Intensitätsverläufe senkrecht zu einer Kantengeraden 26
4.6. Veranschaulichung des Kantenkorrekturverfahrens 27
4.7. Beispielbilder Objektvermessung 31
4.8. Beispiel für diffuse Reflektionen 32
4.9. Filtermaske zur Unterdrückung von diffusen Reflektionen 32
4.10. Beispiel Reflektionsfilter 32
4.11. Beispiel für einen Querschnitt durch die Laserlinie 33

5.1. Ergebnis der Vermessung des Testobjektes 35

1. Einleitung

Die Verarbeitung von 3D-Modellen mit Computern ist ein Bereich der in den letzten 10 Jahren durch die zunehmende Leistungsfähigkeit von Computern gewaltig an Bedeutung gewonnen hat. Sehr eindrucksvoll wird uns dies durch Spezialeffekte in Kinofilmen wie auch in modernen Computerspielen vor Augen geführt. Die virtuelle bzw. simulierte Realität nimmt langsam immer mehr Gestalt an. Auch in vielen anderen Bereichen ist diese Entwicklung deutlich spürbar: Autonome Fahrzeuge schaffen es mittlerweile, 130 Meilen durch unbekanntes Terrain zu fahren[1], indem sie sich ein dreidimensionales Bild von ihrer Umwelt machen. Anwendungen wie Google-Earth ermöglichen es, sich in einem originalgetreuen dreidimensionalen Modell einer Stadt[2] frei zu bewegen.

Erste Entwicklungen von Verfahren zur digitalen Erfassung von 3D-Modellen gehen auf die 80er Jahre zurück. Im Jahr 1992 starteten Prof. Marc Levoy und seine Studenten an der Stanford University in Kalifornien mit dem „Digital Michelangelo Project“[3] ein Projekt, dessen Ziel es war, mithilfe von Laser-Scannern Kunstwerke von Michelangelo in Florenz digital zu erfassen. Realisiert wurde dies 1998–1999 mit Hilfe eines von der Firma Cyberware hergestellten Laserscanners[4]. Das Ergebnis waren sehr hochaufgelöste Modelle (Der Punktabstand betrug minimal $0.25mm$) von insgesamt 6 Statuen aus bis zu 500 Millionen Polygonen.

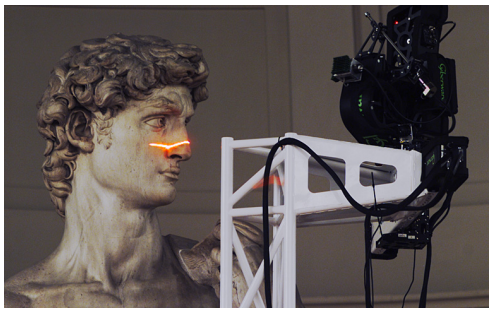


Abbildung 1.1.: Der Large Statue Scanner der Firma Cyberware.
(Quelle: `http://graphics.stanford.edu/projects/mich/`)

---

[1] DARPA Grand Challenge 2005: `http://www.darpa.mil/grandchallenge05/`

[2] zu sehen unter `http://www.3d-stadtmodell-berlin.de` oder `http://www.cybercity.tv/`

[3] The Digital Michelangelo Project: `http://graphics.stanford.edu/projects/mich/`

[4] `http://www.cyberware.com/products/scanners/lss.html`

1.1. Verfahren und Anwendungen der 3D-Messtechnik

Heutzutage gehören 3D-Messsysteme – insbesondere Laserscanner – in vielen Bereichen unterschiedlichster Fachrichtung zum Alltag. So können mit 3D-Scannern in der Fertigungstechnik und Qualitätssicherung produzierte Teile automatisch vermessen und auf ihre Qualität untersucht werden. Bei der Produktion von Filmen und Computerspielen sind solche Systeme mittlerweile unverzichtbar, da sie es zum Einen ermöglichen, menschliche Körperbewegungen im Computer zu erfassen und auf animierte Figuren zu übertragen und es zum Anderen nicht mehr notwendig ist, die animierten Figuren ausschließlich am Computer zu gestalten.

Die unterschiedlichen 3D-Messverfahren werden im Allgemeinen in mechanische und berührungslose bzw. optische Verfahren eingeteilt. Bei den mechanischen Verfahren wird mit einem hochgenau positionierbaren Tastkopf die Oberfläche eines Objektes abgetastet. Ein solches System kann sehr hohe Genauigkeiten erzielen (bis zu einer Genauigkeit von $\pm 0.1 \mu m$), hat allerdings auch einige Nachteile. Zum einen ist mit einem solchen System keine so hohe Erfassungsgeschwindigkeit möglich wie dies mit optischen Verfahren der Fall ist und zum anderen können besonders empfindliche Objekte bei der Vermessung beschädigt oder gar zerstört werden, da ein direkter Kontakt mit dem Objekt für die Messung notwendig ist.

Die optischen Verfahren werden ihrerseits wieder in einige Unterkategorien eingeteilt. Hier gibt es zunächst einmal die direkten Messverfahren, die eine Entfernungsmessung ohne größeren Rechenaufwand ermöglichen. Zu dieser Gruppe gehört zum Beispiel das Lichtlaufzeit-Verfahren (*„Time-of-flight“*), bei dem die Laufzeit eines Laserstrahls zum Objekt und zurück gemessen wird. Dieses Verfahren zeichnet sich dadurch aus, dass selbst auf eine Entfernung von mehreren Kilometern noch sehr genaue Messungen möglich sind. Die Genauigkeit ist jedoch durch die maximale Genauigkeit der Zeitmessung auf einige Millimeter beschränkt. Beispiele für solche Scanner sind die sehr häufig beim Bau von autonomen Fahrzeugen verwendeten Scanner der Firma SICK[5] (siehe Abbildung 1.2).

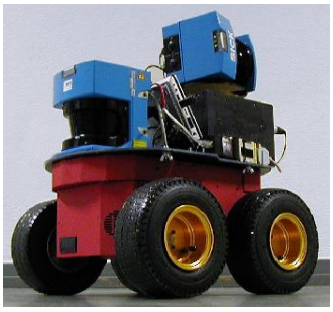


Abbildung 1.2.: Ein Pioneer III Roboter mit 2 SICK-Laserscannern zur 3D-Kartierung.
(Quelle: `http://www.informatik.uni-bonn.de/~schulz/robprak06.html`)

Die zweite Gruppe der optischen Messverfahren sind die indirekten Messmethoden. Hierzu

[5] Homepage der Firma SICK: `http://www.sick.de`

gehören insbesondere die verschiedenen Triangulationsverfahren, bei denen die Position eines Messpunktes aus der relativen Position und Richtung einer Kamera zu einer Lichtquelle berechnet wird.

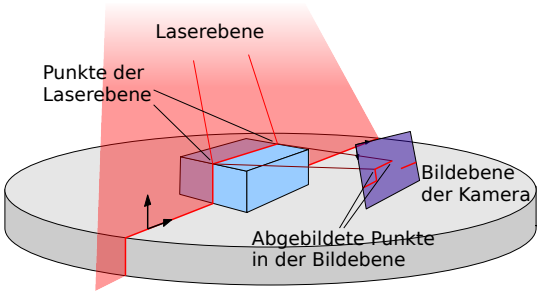


Abbildung 1.3.: Triangulationsmessung mit einem Linienlaser

Diese Verfahren sind nur bei eher geringen Entfernungen zum Messobjekt (bis etwa $10m$) sinnvoll anwendbar, da aufgrund der Streuung der Lichtquelle die Messgenauigkeit mit größer werdendem Abstand abnimmt. Im Nahbereich lassen sich hiermit jedoch weitaus bessere Ergebnisse erzielen als mit dem Lichtlaufzeit-Verfahren (die erzielbare Genauigkeit liegt im Bereich von hundertstel Millimetern).

Die Triangulationsverfahren lassen sich in aktive und passive Verfahren einteilen. Als aktive Verfahren (auch Lichtschnittverfahren) werden die Verfahren bezeichnet, bei denen in einem Kamerabild die von einer Lichtquelle beleuchteten Punkte analysiert werden. Als Lichtquellen kommen Punktlaser, Linienlaser oder Projektoren, die ein Lichtmuster (strukturiertes Licht) auf das Messobjekt projizieren zum Einsatz. Je nach verwendeter Lichtquelle variiert die Erfassungsgeschwindigkeit der Messpunkte. Wird ein Punktlaser eingesetzt, so lässt sich nur ein Punkt zur Zeit vermessen, bei strukturiertem Licht kann die gesamte sichtbare Oberfläche eines Objektes zeitgleich erfasst werden.

Die passiven Triangulationsverfahren kommen ohne eine spezielle Beleuchtung aus. Besonders erwähnenswert ist in diesem Bereich die Stereobildverarbeitung, bei der eine Triangulation mit Korrespondenzpunkten aus Kamerabildern von zwei Kameras, deren Positionen und Richtungen zueinander bekannt sind, berechnet wird.

1.2. Verwandte Technologien

Alle 3D-Messverfahren haben gemainsam, dass sie eine mitunter sehr große Menge von Punkten erzeugen. Über diese Punkte ist zwar bekannt, dass sie alle auf der Oberfläche der Messobjektes liegen, jedoch enthalten die gewonnenen Daten noch keine Information darüber, wie diese Punkte zu der Oberfläche des Objektes zusammengesetzt werden. Dies geschieht in einem Nachbearbeitungsschritt, der *„Mesh-Construction“* genannt wird und die Punkte in geeigneter Weise zu einem Gitternetz verbindet. Ein weiterer Schritt in der Nachbearbeitung der

Daten ist das *„Mesh-Optimizing“*, bei dem die Anzahl der Polygone im Gitternetz reduziert wird, ohne dabei die Genauigkeit des Modells zu verlieren.

Ebenfalls sind für manche Objekte mehrere Scandurchgänge notwendig, da unter Umständen nicht die Gesamte Oberfläche des Objektes mit einem einzigen Scandurchgang erfasst werden kann. In einem solchen Fall müssen die Gitternetze der einzelnen Scandurchgänge zu einem Gitternetz kombiniert werden. Dieser Nachbearbeitungsschritt wird auch *„Mesh-Alignment“* genannt.

1.3. Zielsetzung dieser Arbeit

Diese Arbeit soll den Grundstein für den Aufbau eines 3D-Laserscanners legen. Das zu vermessende Objekt soll in den Scanner auf einen per Schrittmotor gesteuerten Drehteller gestellt werden. Während des Messvorgangs wird der Drehteller dann langsam um 360 Grad gedreht, während eine Kamera für jeden Einzelschritt der Drehung ein Bild speichert. Aus den Einzelbildern kann dann ein 3D-Modell des Messobjektes errechnet werden.

Ziel dieser Arbeit ist es, Verfahren für die Kalibrierung dieses Laserscanners sowie für die Erfassung eines Einzelbildes und dessen Umrechnung in Realkoordinaten zu entwickeln und zu implementieren.

1.4. Gliederung

Kapitel 1 dient als Einführung in die Thematik der 3D-Objektvermessung, schildert den heutigen Stand der Technik und erläutert die Zielsetzung dieser Arbeit.

Kapitel 2 gibt eine Einführung in die für das Verständnis der Arbeit notwendigen Grundlagen der linearen Algebra und der digitalen Bildverarbeitung.

Kapitel 3 stellt die für diese Arbeit verwendete Hardware und den Messaufbau vor.

Kapitel 4 erläutert die Konzeption der Kalibrierung und der Objektvermessung sowie die dahinterstehenden Überlegungen.

Kapitel 5 schildert die Ergebnisse der vorgenommenen Messungen und erklärt die Faktoren, die die Genauigkeit der Messung beeinflussen.

Kapitel 6 beschäftigt sich mit der im Rahmen dieser Arbeit erstellten Software. Neben Erläuterungen zur Struktur der Implementierung finden sich in diesem Kapitel auch Hinweise zu der Bedienung und der Installation der Software.

Kapitel 7 gibt einen Rückblick auf die vorliegende Arbeit und einen Ausblick auf wieterführende Arbeiten.

2. Grundlagen

Da sich diese Arbeit sehr viel mit den Bereichen der linearen Algebra und der digitalen Bildverarbeitung beschäftigt, soll in diesem Kapitel auf die Grundlagen und verwendeten Verfahren etwas näher eingegangen werden.

Inhalt

2.1. Mathematische Grundlagen 6
2.1.1. Bildtransformationen 6
Affine Transformationen 6
Homogene Koordinaten und perspektivische Transformationen 7
2.1.2. Lösung linearer Ausgleichsprobleme 8
QR-Zerlegung 9
Singulärwert-Zerlegung 9
2.2. Grundlagen der Bildverarbeitung 10
2.2.1. Lineare Filterung mit Faltungskernen 10
2.2.2. Kantendetektion 11
Canny-Algorithmus 12
2.2.3. Hough-Raum und Hough-Transformation 13
2.2.4. Bilineare Interpolation 14

2.1. Mathematische Grundlagen

2.1.1. Bildtransformationen

Affine Transformationen

In der Geometrie bezeichnen affine Tranformationen alle auf einen Punkt bzw. einer Menge von Punkten anwendbaren geometrischen Operationen wie Rotation, Skalierung, Scherung, Spiegelung und Verschiebung. Alle diese Operationen haben gemeinsam, dass die Abbildungen von 3 Punkten, die auf einer Geraden liegen ihrerseits auch kollinear sind. Desweiteren sind auch die Abbildungen von parallelen Geraden ebenfalls parallel. Mathematisch lassen sich affine Transformationen als eine lineare Transformation $\mathbf{T}$ gefolgt von einer Verschiebung $\vec{b}$ beschreiben.

$$\vec{a} \quad \mapsto \quad \mathbf{T}\vec{a}+\vec{b} = \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix} \begin{pmatrix} a_x \\ a_y \end{pmatrix} + \begin{pmatrix} b_x \\ b_y \end{pmatrix} \tag{2.1}$$

Um alle affinen Transformationen (inklusive der Verschiebung) in einer einzigen Transformationsmatrix abbilden zu können, werden die Vektoren um eine dritte Komponente erweitert, die in diesem Fall konstant gleich eins ist. Dadurch lassen sich die affinen Transformationen vereinfacht darstellen mit

$$\vec{a} \quad \mapsto \quad \mathbf{T}\vec{a} = \begin{pmatrix} t_{11} & t_{12} & b_x \\ t_{21} & t_{22} & b_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_x \\ a_y \\ 1 \end{pmatrix} \tag{2.2}$$

Homogene Koordinaten und perspektivische Transformationen

Jeder Punkt $\vec{p}$ in der euklidischen Ebene mit kartesischen Koordinaten $\begin{pmatrix} p_x & p_y \end{pmatrix}^T$ lässt sich durch einen Punkt $\vec{p}^*$ in der projektiven Ebene mit den homogenen Koordinaten $\begin{pmatrix} hp_x & hp_y & h \end{pmatrix}^T$ beschreiben. Umgekehrt gilt:

$$\vec{p^*} = \begin{pmatrix} p_x^* \\ p_y^* \\ p_h^* \end{pmatrix} \quad \rightarrow \quad \vec{p} = \begin{pmatrix} p_x \\ p_y \end{pmatrix} \qquad \text{mit} \begin{cases} p_x = \frac{p_x^*}{p_h^*} \\ p_y = \frac{p_y^*}{p_h^*} \end{cases} \tag{2.3}$$

Hieraus ergibt sich, dass zwei Punkte $\vec{p}^*$ und $\vec{q}^*$ in der projektiven Ebene mt homogenen Kordinaten $\vec{p}^* = \begin{pmatrix} ax & ay & a \end{pmatrix}^T$ und $\vec{q}^* = \begin{pmatrix} bx & by & b \end{pmatrix}^T$ immer denselben Punkt in der euklidischen Ebene beschreiben. Dieser Zusammenhang wird auch als *homogene Gleichheit* bezeichnet.

Durch die Einführung der homogenen Koordinaten wird es möglich, perspektivische Transformationen zu beschreiben, also solche Transformationen wie sie benötigt werden, wenn zum Beispiel eine Abbildung aus einem Raum bzw. einer Ebene im Raum auf die Bildebene einer Kamera beschrieben werden soll. Diese Abbildungen sind nicht parallelenerhaltend, bilden also parallele Geraden nicht auf parallele Geraden ab. Sie werden wie die affinen Tranformationen durch eine Transformationsmatrix $\mathbf{H}$ beschrieben.

$$\vec{a^*} \quad \mapsto \quad \mathbf{H}\vec{a^*} = \begin{pmatrix} h_{11} & \cdots & h_{13} \\ \vdots & \ddots & \vdots \\ h_{31} & \cdots & h_{33} \end{pmatrix} \begin{pmatrix} a_x^* \\ a_y^* \\ a_h^* \end{pmatrix} \tag{2.4}$$

Eine Veranschaulichung der perspektivischen Transformation ist in Abbildung 2.1 auf S. 8 gegeben. Weitere und ausführlichere Informationen zu diesem Thema finden sich in (Rogers und Adams, 1976), (Hartley und Zisserman, 2000) oder (Meisel, 2007)

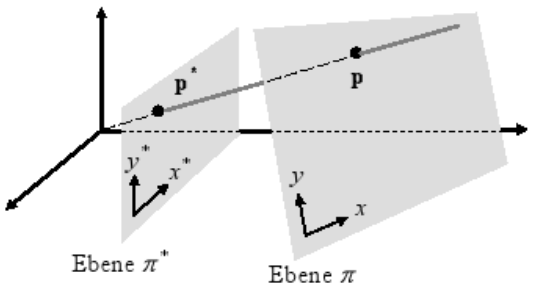


Abbildung 2.1.: Perspektivische Transformation: durch die Transformationsmatrix $\mathbf{H}$ wird ein Punkt P aus der Ebene $\pi \in \mathbb{R}^2$ auf einen Punkt P^* in der Ebene $\pi^* \in \mathbb{R}^2$ abgebildet. Sowohl π als auch π^* sind hierbei euklidische Ebenen.
(Bild entnommen aus „*Anwendungen - 3D Bildverarbeitung*“ (Meisel, 2007))

2.1.2. Lösung linearer Ausgleichsprobleme

Lineare Ausgleichsprobleme treten überall dort auf, wo die Parameter eines mathematischen Modells aus einer Reihe von Messwerten bestimmt werden müssen. Da Messungen in der Praxis immer mit Messfehlern behaftet sind, lassen sich die Parameter des Modells nicht direkt aus den Messwerten bestimmen. Daher ist es notwendig, ein Verfahren anzuwenden mit dem die bestmögliche Näherung an die tatsächlichen Werte der Modell-Parameter ermittelt werden kann.

Durch Einsetzen der einzelnen Messwerte in die Modellgleichung entsteht ein Gleichungssystem mit ebensovielen Gleichungen wie Messwerten. Dieses Gleichungssystem lässt sich in Matrixschreibweise als $\mathbf{M}\vec{x} = \vec{b}$ darstellen, wobei $\vec{x}$ der Vektor der gesuchten Modell-Parameter ist und $\mathbf{M}$ und $\vec{b}$ die Messwerte enthalten. Dieses lineare Gleichungssystem ist in der Regel nicht eindeutig lösbar[6].

Die bestmögliche Näherung dieses linearen Ausgleichsproblems ist durch denjenigen Vektor $\vec{x}$ gegeben, für den die Summe der Fehlerquadrate

$$\sum_{i=1}^{m} \varepsilon_i^2 = |\vec{\varepsilon}|^2 = \left|\mathbf{M}\vec{x} - \vec{b}\right|^2 \tag{2.5}$$

minimal ist.

Es lässt sich zeigen[7], dass ein solches lineares Ausgleichsproblem immer eindeutig lösbar ist.

---

[6] Eine eindeutige Lösung existiert genau dann nicht, wenn $\mathbf{M} \in \mathbb{R}^{m \times n}$ eine Matrix mit $m > n$ ist und $\mathbf{M}$ keine linear abhängigen Zeilenvektoren $\vec{m}_i$ enthält. Weniger formal ausgedrückt, existiert dann keine eindeutige Lösung, wenn das zugehörige Gleichungssystem mehr Gleichungen als Unbekannte enthält.

[7] Die Beweisführung und weitergehende Erklärungen hierzu finden sich z.B. in (Mackens und Voss, 1993) oder (wikipedia.org, 2007c).

Die Lösung ist dann gegeben durch

$$\vec{x} = \mathbf{M}^{\dagger}\vec{b} \qquad \text{mit } \mathbf{M}^{\dagger} = (\mathbf{M}^{\mathbf{T}}\mathbf{M})^{-1}\mathbf{M}^{\mathbf{T}} \tag{2.6}$$

Die Matrix $\mathbf{M}^{\dagger}$ wird verallgemeinerte Inverse (auch Pseudoinverse oder Moore-Penrose-Inverse) der Matrix $\mathbf{M}$ genannt.

In dieser Arbeit wurden zwei unterschiedliche Verfahren zur Lösung der linearen Ausgleichsprobleme eingesetzt, die im folgenden kurz vorgestellt werden sollen. Beschreibungen der Einsatzgebiete finden sich in Kapitel 4 in den Abschnitten 4.1.2 auf S.23 und 4.1.3 auf S.28

QR-Zerlegung

Bei der QR-Zerlegung wird die Matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ in zwei Matrizen $\mathbf{Q}$ und $\mathbf{R}$ zerlegt, so dass $\mathbf{M} = \mathbf{QR}$ gilt. Hierbei ist $\mathbf{Q}$ eine orthogonale Matrix[8] und $\mathbf{R}$ eine obere Dreiecksmatrix. Zur Berechnung der QR-Zerlegung wird die Householder-Transformation[9] verwendet. Es ergibt sich damit für das lineare Ausgleichsproblem $\mathbf{M}\vec{x} = \vec{b}$

$$\mathbf{QR}\vec{x} = \vec{b} \quad \Leftrightarrow \quad \mathbf{R}\vec{x} = \mathbf{Q}^{\mathbf{T}}\vec{b} \tag{2.7}$$

Da es sich bei $\mathbf{R}$ um eine obere Dreiecksmatrix handelt, lässt sich dieses Gleichungssystem nun durch Rückwärtseinsetzen (Gauss-Elimination) nach $\vec{x}$ auflösen.

Weitere Informationen zu der QR-Zerlegung und der Householder-Transformation finden sich in (wikipedia.org, 2007d).

Singulärwert-Zerlegung

In Gleichung (2.6) wurde gezeigt, dass es durch Verwendung der Pseudoinversen eine allgemeine Lösung für das lineare Ausgleichsproblem gibt. Mit der Singulärwertzerlegung (SVD[10]) existiert ein zwar aufwändiges aber auch numerisch sehr stabiles Verfahren, mit dem die Pseudoinverse $\mathbf{M}^{\dagger}$ berechnet werden kann.

Jede Matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ kann in drei Matrizen $\mathbf{U} \in \mathbb{R}^{m \times m}$, $\Sigma \in \mathbb{R}^{m \times n}$ und $\mathbf{V} \in \mathbb{R}^{n \times n}$ zerlegt werden, so dass gilt:

$$\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}} \tag{2.8}$$

---

[8] Orthogonale Matrizen sind Matrizen, deren Inverse gleich ihrer Transponierten ist. Daher gilt $\mathbf{Q}^{\mathbf{T}}\mathbf{Q} = \mathbf{I}$.

[9] Bei der Householder-Transformation handelt es sich um ein numerisches Verfahren zur Berechnung der QR-Zerlegung einer Matrix. Details über den Algorithmus und dessen Implementierung finden sich in (Press u. a., 1992)

[10] SVD = *engl.* **S**ingular **V**alue **D**ecomposition

Hierbei sind die Matrizen $\mathbf{U}$ und $\mathbf{V}$ orthogonale Matrizen, die Matrix Σ ist eine Diagonalmatrix und enthält die singulären Werte von $\mathbf{M}$ als Diagonalelemente. Die Pseudoinverse $\mathbf{M}^\dagger$ der Matrix $\mathbf{M}$ lässt sich nun berechnen durch

$$\mathbf{M}^\dagger = \mathbf{V}\Sigma^\dagger\mathbf{U}^\mathbf{T} \tag{2.9}$$

In dieser Gleichung ist $\Sigma^\dagger$ ebenfalls eine Diagonalmatrix, deren Diagonalelemente die Kehrwerte der Diagonalelemente aus Σ sind, also $\sigma_{ii}^\dagger = \frac{1}{\sigma_{ii}}$.

Basierend auf den singulären Werten einer Matrix lässt sich auch die Konditionszahl κ einer Matrix als das Verhältnis des größten zu dem kleinsten singulären Wert definieren. Diese Zahl gibt Aufschluss darüber wie empfindlich die Unbekannte $\vec{x}$ des linearen Gleichungssystems auf kleine Veränderungen der Werte in $\mathbf{M}$ bzw. $\vec{b}$ reagiert. Bei einer großen Konditionszahl wird von einem schlecht konditionierten Gleichungssystem gesprochen, bei kleiner Konditionszahl von einem gut konditionierten.[11]

2.2. Grundlagen der Bildverarbeitung

In der Implementierung zu dieser Arbeit wurden zahlreiche bekannte Verfahren der Bildverarbeitung eingesetzt. Die wichtigsten dieser Verfahren sollen nun ein wenig näher betrachtet werden.

2.2.1. Lineare Filterung mit Faltungskernen

Viele Bildverarbeitungsverfahren wie z.B. die unterschiedlichen Verfahren zur Kantendetektion (siehe Abschnitt 2.2.2) oder das Weichzeichnen eines Bildes lassen sich mithilfe von Faltungs- oder Filterkernen und dem Faltungsoperator realisieren.

Der Grauwert eines jeden Pixels im gefilterten Bild ergibt sich hierbei aus einer gewichteten Summe der Grauwerte benachbarter Pixel im Quellbild. Welche benachbarten Pixel mit welcher Gewichtung in diese Summe mit eingehen, wird mit dem Faltungskern angegeben. Dieser Zusammenhang soll in Abbildung 2.2 verdeutlicht werden.

Mathematisch ausgedrückt ergibt sich der Grauwert $f^*(x,y)$ eines gefilterten Bildes f^* aus den Grauwerten des Quellbildes f mit gleichen Dimensionen und einem Faltungskern k der Größe $m \times n$ (wobei m und n ungerade Zahlen sind) durch die Gleichung[12]:

$$f^*(x,y) = \sum_{s=-a}^{a}\sum_{t=-b}^{b} k(s,t)f(x+s,y+t) \qquad \text{mit} \begin{cases} a = \frac{m-1}{2} \\ b = \frac{n-1}{2} \end{cases} \tag{2.10}$$

[11] Weitere Informationen zur Singulärwertzerlegung und ihrer Anwendung finden sich in (Mackens und Voss, 1993), (Gramlich, 2004), (Jackèl, 2005), (wikipedia.org, 2007e) und (Weisstein, 2006). Die Implementierung des SVD-Algorithmus wird in (Press u. a., 1992) näher beschrieben.

[12] entnommen aus (Gonzales und Woods, 2002)

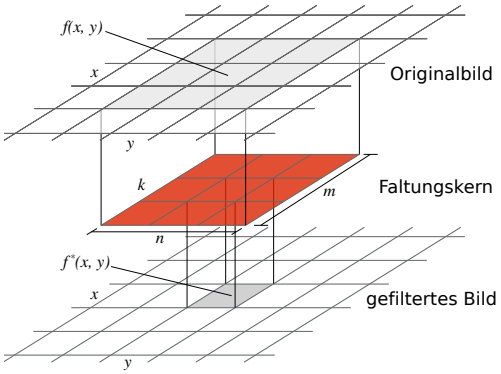


Abbildung 2.2.: Veranschaulichung der linearen Filterung

2.2.2. Kantendetektion

Ein sehr häufig im Rahmen der digitalen Bildverarbeitung eingesetztes Verfahren ist die Kantendetektion. Im wesentlichen geht es dabei darum, die Grauwertübergänge (die Kanten) in einem Bild hervorzuheben und homogene Bereiche, also Bereiche mit ähnlichen Farb- bzw. Grauwerten, zu unterdrücken.[13]

Die Kontinuität eines Bildes in einem bestimmten Bereich kann mithilfe der Gradienten[14] des Bildes untersucht werden. Um diese zu bestimmen, werden Faltungskerne (vgl. Abschnitt 2.2.1) eingesetzt, mit deren Hilfe die diskreten partiellen Ableitungen der Grauwertfunktion des Bildes berechnet werden können. Ein Beispiel für einen solchen Satz von Faltungskernen sind die Sobel-Faltungskerne , die in Abbildung 2.3 dargestellt sind.

[13] Im folgenden wird lediglich das Verfahren der Kantenerkennung in Grauwertbildern näher erläutert. In Farbbildern werden die Gradienten entweder komponentenweise oder über die Formulierung des Farbkontrastes berechnet. Das Verfahren hierzu wird in (Gonzales und Woods, 2002), s.335 ff. näher erläutert.

[14] Der Gradient ∇f einer Funktion mit mehreren Variablen ist ein Vektor, dessen Komponenten die partiellen Ableitungen der Funktion enthalten:

$$\nabla f(x,y) = \begin{pmatrix} G_x & G_y \end{pmatrix}^T = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{pmatrix}^T$$

-1	-2	-1
0	0	0
1	2	1

(a)

1	0	-1
2	0	-2
1	0	-1

(b)

Abbildung 2.3.: Sobel-Faltungskerne: (a) zeigt den Faltungskern für die Gradienten in x-Richtung, (b) den für die Gradienten in y-Richtung

Das Ergebnis der Anwendung dieser Faltungskerne sind 2 Grauwertbilder G_x und G_y, die die erste bzw. zweite Komponente der Gradienten darstellen. Hieraus lassen sich nun die Gradientenbeträge G sowie die Richtung der Gradienten α[15] bestimmen:

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.11a}$$

und

$$\alpha = \arctan \frac{G_y}{G_x} \tag{2.11b}$$

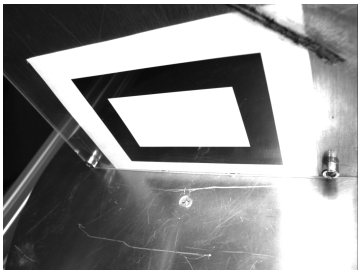
Die Abbildungen 2.4a–e auf S. 13 zeigen ein Beispiel für die Kantenerkennung mithilfe der Sobel-Faltungskerne.

Canny-Algorithmus

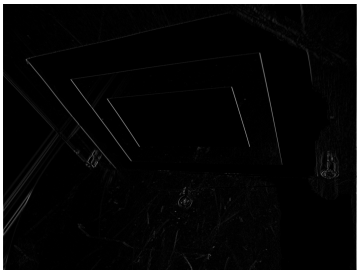
In dieser Arbeit wurde für die Kantenerkennung der Canny-Algorithmus eingesetzt. Bei diesem Algorithmus handelt es sich um ein auf den Sobel-Faltungskernen aufbauendes Verfahren, welches gegenüber der direkten Verwendung des Gradientenbildes einige Vorteile bietet:

- Da das Quellbild vor der Berechnung der Gradienten mit einem gaußschen Weichzeichner mit einstellbarer Größe und Varianz geglättet wird, lässt sich über diese beiden Parameter der gewünschte Detaillierungsgrad des Ergebnisses steuern.
- Alle Kanten innerhalb des Bildes werden auf eine Breite von einem Pixel ausgedünnt.
- Das Kantenbild ist ein Binärbild.

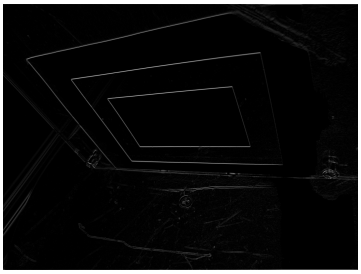
[15] Es ist zu beachten, dass es sich bei G, G_x, G_y und α jeweils um Funktionen mit den Parametern x und y handelt.



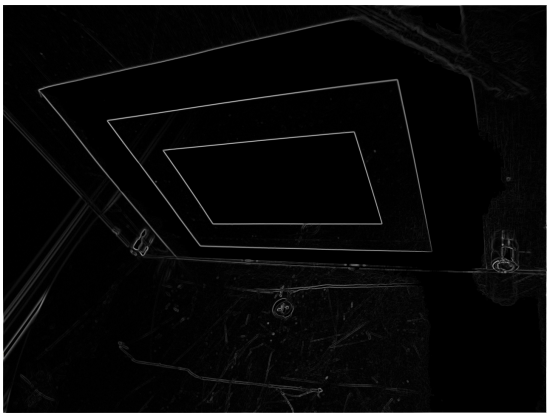
(a) Originalbild



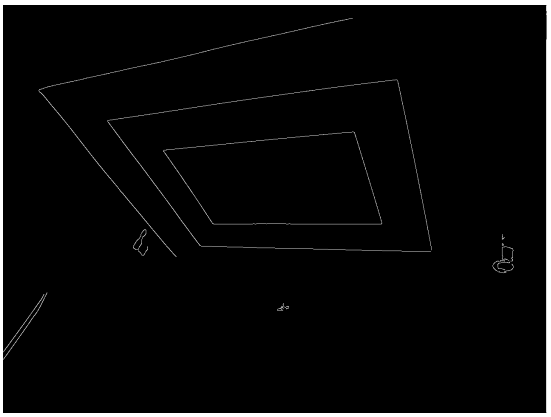
(b) Kantenstärke in x-Richtung



(c) Kantenstärke in y-Richtung



(d) kombinierte Kantenstärke



(e) Ergebnis des Canny-Algorithmus

Abbildung 2.4.: Beispiel zur Kantendetektion.

2.2.3. Hough-Raum und Hough-Transformation

In einem Bild die darin enthaltenen Geraden zu finden, ist ein häufig zu lösendes Problem in der Bildverarbeitung. Ein Verfahren, mit denen dieses Problem gelöst werden kann ist die Hough-Transformation.

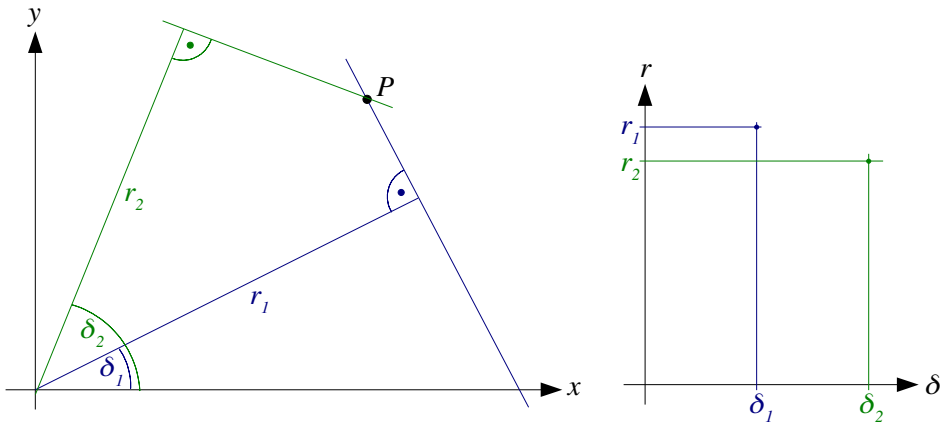
Jede Gerade im euklidischen Raum lässt sich durch die zwei Parameter δ und r ihrer normierten Normalen beschreiben. Der Parameter δ gibt hierbei den Winkel der Normalen zur x-Achse des Koordinatensystems und r die Länge der Normalen an.

Bei der Hough-Transformation handelt es sich um ein Verfahren, mit dem ein Bild[16] auf einen Parameterraum („*Hough-Raum*“) mit den beiden Parametern δ und r abgebildet wird. Hierbei existiert zu jedem Punkt im Bild eine bestimmte Anzahl[17] von Punkten im Parameterraum,

[16] Hierzu wird meistens ein binäres Kantenbild verwendet, wie es zum Beispiel durch den Canny-Algorithmus (vgl. Abschnitt 2.2.2) erzeugt wird.

[17] Da der Parameterraum die beiden Parameter δ und r nicht kontinuierlich, sondern diskret erfasst, ist die Anzahl der Punkte im Parameterraum begrenzt.

durch die Geraden beschrieben werden, die durch diesen Punkt verlaufen. Diese Transformation wird in Abbildung 2.5 auf S. 14 veranschaulicht.



(a) Euklidischer Raum **(b)** Parameterraum

Abbildung 2.5.: Veranschaulichung der Hough-Transformation: der Punkt P lässt sich durch eine unendliche Anzahl von hypothetischen Geraden beschreiben, von denen hier zwei dargestellt sind. Jeder Punkt im Parameterraum (dem Hough-Raum) beschreibt eine dieser hypothetischen Geraden.

Für jeden Bildpunkt, dessen Intensitätswert über einem anzugebenden Schwellwert liegt, werden nun im Parameterraum die Werte aller Punkte inkrementiert, durch die Geraden beschrieben werden, die durch diesen Bildpunkt verlaufen könnten.

Wurden alle Pixel des Bildes bearbeitet, lassen sich die in dem Bild vorhandenen Geraden aus dem Parameterraum ablesen. Hierzu werden innerhalb des Parameterraums alle lokalen Maxima, deren Wert größer als ein angegebener Schwellwert (dieser beschreibt die Mindestanzahl von Bildpunkten, die auf der durch den Punkt im Parameterraum beschriebenen Geraden liegen und damit die Mindestlänge der Geraden) ist, gesucht.

2.2.4. Bilineare Interpolation

Interpolationsverfahren werden in der Bildverarbeitung eingesetzt, um Farb- bzw. Grauwerte an Positionen zwischen einzelnen Pixeln zu ermitteln. Bei der bilinearen Interpolation wird hierzu ein gewichteter Mittelwert der nächsten vier benachbarten Pixel gebildet. Abbildung 2.6 verdeutlicht das Verfahren zur Bildung dieses Mittelwertes.

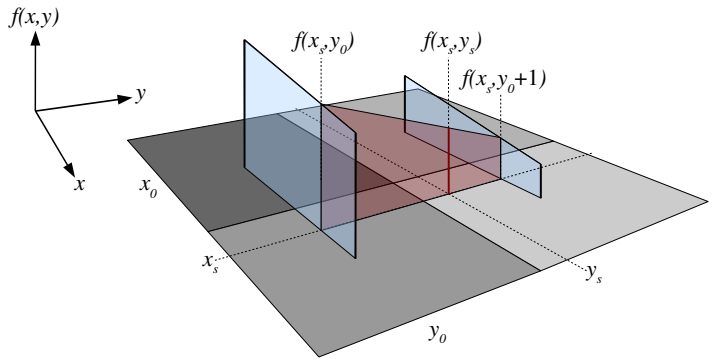


Abbildung 2.6.: Bilineare Interpolation: Der Grauwert in einem Punkt (x,y) ist gegeben durch $f(x,y)$. Der gesuchte Grauwert $f(x_s,y_s)$ wird mittels Interpolation entlang beider Achsen ermittelt.

Für den interpolierten Grauwert $f(x_s,y_s)$ im Punkt (x_s,y_s) erhält man somit[18]:

$$\begin{aligned} f(x_s,y_s) &= f(x_0,y_0)\,(1-x_s)(1-y_s) + f(x_0+1,y_0)\,x_s(1-y_s) \\ &+ f(x_0,y_0+1)\,(1-x_s)y_s + f(y_0+1,x_0+1)x_s y_s \end{aligned} \tag{2.12}$$

[18] Herleitung dieser Gleichung siehe (wikipedia.org, 2007a)

3. Technischer Aufbau

Inhalt

3.1. Der Grundrahmen 16
3.2. Laser- und Kamera-Träger 16
3.3. Kalibriervorrichtung 17
3.4. Spezifikationen von Laser und Kamera 17

Der technische Aufbau des Laserscanners besteht aus einer Industriekamera, einem Laser mit einer zylindrischen Linse für die Linienprojektion und einem Drehteller, der von einem hochgenau positionierbaren Schrittmotor angetrieben wird. In diesem Kapitel werden die Details dieses Aufbaus näher beschrieben.

3.1. Der Grundrahmen

Der Grundrahmen ist vollständig aus Rexroth-Strebenprofilen gebaut und besteht aus einem Kasten mit zwei in Höhe und Entfernung zur Mitte verstellbaren Trägern, an denen der Laser und die Kamera montiert werden können. Mittig am Boden des Gestells angebracht ist ein Träger für den Schrittmotor mit dem Drehteller. Der gesamte Rahmen ist 74cm hoch, 68cm breit und 48cm tief. Abbildung 3.1 zeigt das fertig aufgebaute Gestell mit montierter Kamera und Laser.

3.2. Laser- und Kamera-Träger

Die Träger für Laser und Kamera sind so konstruiert, dass sich Laser und Kamera an jede beliebige Position innerhalb des Grundrahmens verschieben lassen (siehe Abbildung 3.2a). Sowohl der Laser als auch die Kamera sind auf einem Stativkopf befestigt, der sich auf einem Schlitten entlang des Querträgers verschieben lässt (vgl. Abbildung 3.2b und d). Die Positionierung der Träger und Schlitten lässt sich arretieren, so dass nach vorgenommener Kalibrierung keine versehentlichen Änderungen an der Positionierung das Messergebnis negativ beeinflussen können.

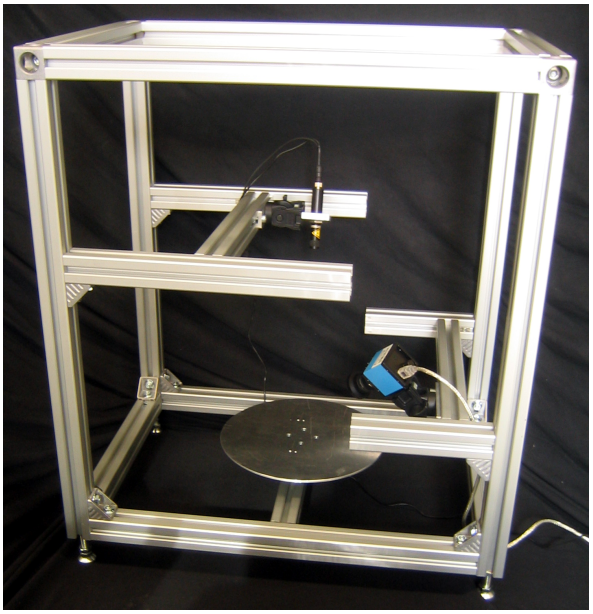


Abbildung 3.1.: Der Grundrahmen

3.3. Kalibriervorrichtung

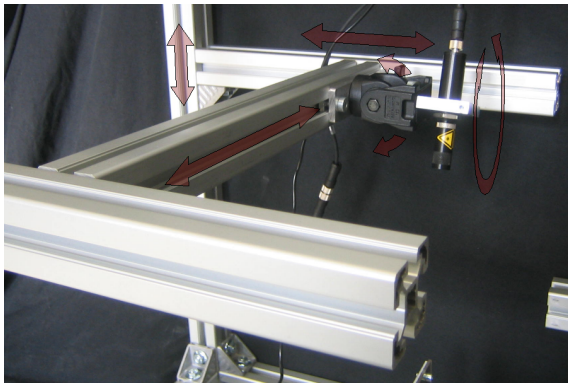
Die Vorrichtung für die Kalibrierung besteht aus einer Aluminium-Platte und einer Plexiglasscheibe, die vor die diese Platte geschraubt wird (siehe Abbildung 3.3b auf S. 18). Zwischen Aluminium-Platte und Plexiglasscheibe wird ein auf Papier gedrucktes Kalibrierungsmuster eingespannt. Die Kalibriervorrichtung wird für die Kalibrierung mit zwei Winkeln auf dem Drehteller verschraubt und so ausgerichtet, dass das eingespannte Blatt mit dem Kalibrierungsmuster genau auf der Mittellinie des Drehtellers liegt.

3.4. Spezifikationen von Laser und Kamera

Bei dem eingesetzten Lasermodul handelt es sich um ein von der Firma Z-Laser hergestelltes 5 mW Lasermodul mit einer frei fokussierbaren Linienoptik und einem Öffnungswinkel von 30 Grad[19]. Die verwendete Kamera ist eine monochrome Industriekamera der Firma The Imaging Source (Typbezeichnung: DMK 41BF02) mit einer Auflösung von 1280x1024 Bildpunkten und Firewire Schnittstelle[20]. Als Objektiv wurde ein speziell für den Computer-Vision Bereich konzipiertes Objektiv von computar verwendet (Typbezeichnung: H0514-MP).

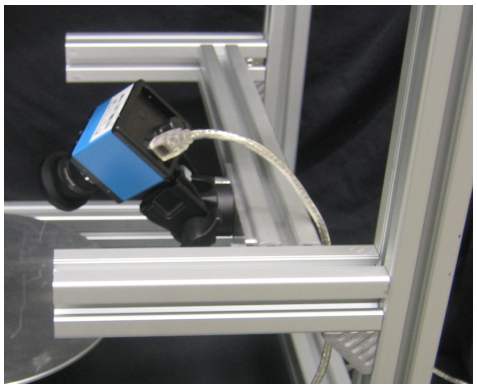
[19] Homepage des Herstellers: `http://www.z-laser.com/zlaser_de/visionlaser/zv-ttl`

[20] Zu finden unter `http://www.theimagingsource.com/de/products/cameras/`

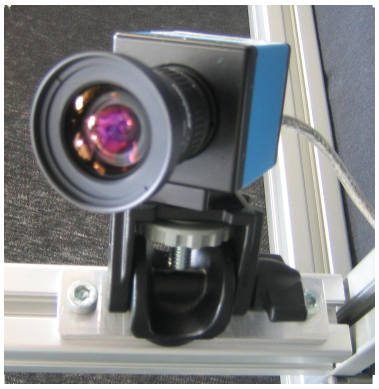


(a) Laserträger mit eingezeichneten Freiheitsgraden

(b) Laser mit Halterung, Stativkopf und Schlitten

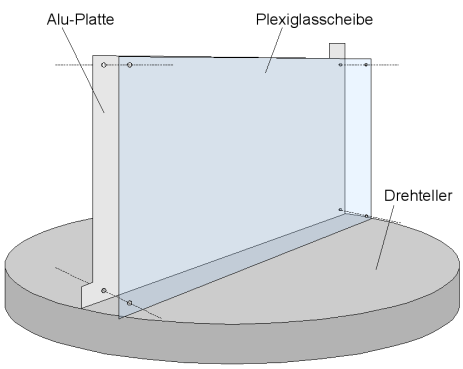


(c) Kameraträger

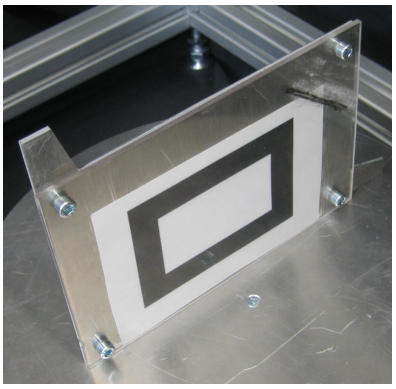


(d) Kamera mit Stativkopf und Schlitten

Abbildung 3.2.: Laser- und Kameraträger



(a) schematischer Aufbau der Kalibriervorrichtung



(b) Kalibriervorrichtung mit Kalibrierungsmuster

Abbildung 3.3.: Kalibrierungsvorrichtung

4. Konzeption

Im dem nun folgenden Kapitel werden die Hintergründe, die Konzeption und die mathematischen Zusammenhänge der implementierten Verfahren für die Kalibrierung und Objektvermessung näher erläutert. Die hierbei verwendeten mathematischen und bildverarbeitungstechnischen Grundlagen werden in Kapitel 2 ab S.6 beschrieben.

Inhalt

4.1. Kalibrierung 19
4.1.1. Vorbereitung 20
Ausrichtung des Lasers 20
Ausrichtung und Einstellung der Kamera 21
4.1.2. Bestimmung der Referenzpunkte in der Bildebene 22
Algorithmus zu Bestimmung der Referenzpunkte 22
Schritt 1: Näherungsweise Bestimmung der Kantenposition 23
Schritt 2: Bestimmung der exakten Kantenposition 23
Schritt 3: Sortieren der Geraden und Bestimmung der Schnittpunkte . 26
4.1.3. Berechnung der Transformationsmatrix 28
4.2. Objektvermessung 30
4.2.1. Vorbereitung 30
4.2.2. Vorverarbeitung des Kamerabildes 30
4.2.3. Extraktion der Oberflächenpunkte aus dem Bild 31
4.2.4. Berechnung der Koordinaten in der Laserebene 32

4.1. Kalibrierung

Um eine Objektvermessung durchführen zu können, ist es notwendig aus Koordinaten im Kamerabild (der Bildebene) die entsprechenden Koordinaten in der Laserebene berechnen zu können. Wie in Abschnitt 2.1.1 bereits erwähnt wurde, wird hierfür die Transformationsmatrix einer perspektivischen Transformation benötigt, mit der Koordinaten in der Bildebene auf Koordinaten in der Laserebene abgebildet werden können.

Um diese Transformationsmatrix zu berechnen, werden, wie im Folgenden gezeigt wird, mindestens vier Punkte benötigt, deren Koordinaten in der Laserebene ($\vec{l}$, gemessen in *mm*) genau bekannt sind. Die dazugehörigen Koordinaten der Abbildungen dieser Punkte ($\vec{c}$, gemessen in *pixel*) können dann im Kamerabild bestimmt werden (siehe Abschnitt 4.1.2).

Um die Koordinaten der Referenzpunkte angeben zu können ist die Wahl eines Bezugspunktes (Ursprung des Koordinatensystems der Laserebene) notwendig. Dieser wird zunächst auf die linke untere Ecke des Kalibrierungsmusters gelegt.

Im Folgenden soll nun die Konzeption des für die Kalibrierung entwickelten Verfahrens näher erläutert werden. Das hier beschriebene Kalibrierungsverfahren ist an das in der Publikation *„Building an Accurate Range Finder with Off the Shelf Components“* (Jezouin u. a., 1988) vorgestellte Verfahren angelehnt.

In diesem Verfahren wird ein Kalibrierungsmuster mit bekannten Abmessungen in die Laserebene gebracht, wodurch eine Reihe von Referenzpunktpaaren ermittelt werden kann. Diese Referenzpunktpaare ermöglichen es, die Parameter der bereits erwähnten perspektivischen Transformation zu berechnen.

In den folgenden Abschnitten werden die einzelnen Schritte, die nötig sind um die Kalibrierung durchzuführen, im Detail erläutert und an Beispielen verdeutlicht.

4.1.1. Vorbereitung

Ein Kalibrierungsmuster, wie in Abbildung 4.1 dargestellt, wird auf ein Blatt Papier gedruckt, welches in die Kalibriervorrichtung eingespannt wird (siehe Abbildung 3.3 auf S. 18).

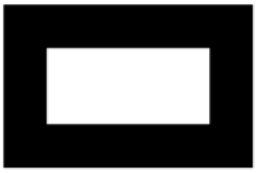
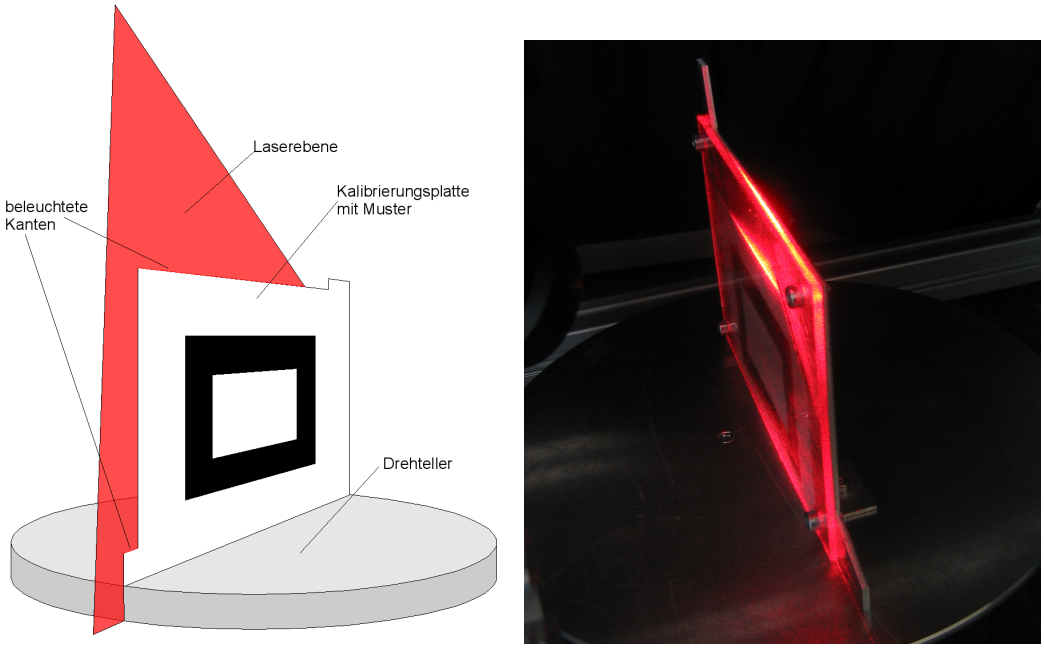


Abbildung 4.1.: Das verwendete Kalibrierungsmuster

Hierbei sind die genauen Abmessungen des Kalibrierungsmusters von entscheidender Bedeutung. Der Bezugspunkt für die Koordinaten der acht Eckpunkte des Kalibrierungsmusters ist hierbei frei wählbar.

Ausrichtung des Lasers

Da die Kalibriervorrichtung auf dem Drehteller verschraubt und damit unbeweglich ist, muss der Laser nun so justiert werden, dass er sowohl die obere Kante der Kalibriervorrichtung als auch die herausstehende Ecke am unteren Ende der Platte beleuchtet (siehe Abbildung 4.2 auf S. 21).



(a) Schematische Darstelluung der Laserausrichtung **(b)** Versuchsaufbau bei der Ausrichtung des Lasers

Abbildung 4.2.: Ausrichtung des Lasers

Die Linse des Lasers sollte so eingestellt werden, dass die auf dem Drehteller sichtbare Linie möglichst scharf zu erkennen ist. Ist die Laserebene ausgerichtet, können die Eckpunkte des Kalibrierungsmusters als bekannte Punkte in der Laserebene betrachtet werden.

Ausrichtung und Einstellung der Kamera

Nachdem der Laser ausgerichtet wurde, müssen nun die Ausrichtung und Einstellungen der Kamera vorgenommen werden. Die Kamera sollte so positioniert werden, dass der Winkel zwischen Kamera und Laser etwa 45 Grad beträgt. Das Kalibrierungsmuster darf auf keinen Fall über den Rand des Kamerabildes hinausragen. Ist die Position und Ausrichtung der Kamera eingestellt, müssen noch die Werte für Brennweite und Blende des Objektives sowie die Empfindlichkeit und Belichtungszeit des Sensors justiert werden.

Grundsätzlich ist es hierbei empfehlenswert die Größe der Blendenöffnung der Kamera so klein wie möglich zu halten, da daraus ein großer Tiefenschärfebereich resultiert[21], wodurch die Kanten und Eckpunkte des Kalibrierungsmusters auf ihrer gesamten Breite scharf abgebildet werden können. Die ebenfalls daraus resutierende kleinere Lichtausbeute kann durch eine

[21] Der Tiefenschärfebereich eines Objektives ist der Tiefenbereich (mindestabstand bis höchstabstand vor dem Objektiv) in dem ein Objekt vor der Kamera scharf abgebildet werden kann.

entsprechende Beleuchtung, eine längere Belichtungszeit oder ein Anheben der Empfindlichkeit des Sensors kompensiert werden.

Nachdem die Einstellungen von Laser und Kamera vorgenommen wurden, müssen die Halterungen arretiert werden, um Fehler durch ein versehentliches Verstellen zu vermeiden.

4.1.2. Bestimmung der Referenzpunkte in der Bildebene

Um die Koordinaten $\vec{c}$ der acht Referenzpunkte in der Bildebene so genau wie möglich zu bestimmen, sind einige Schritte notwendig. Es wird zunächst eine Abschätzung über die Positionen und Richtungen der Kanten des Kalibrierungsmusters vorgenommen. Anschließend werden diese Abschätzungen korrigiert und die Eckpunkte des Kalibrierungsmusters als Schnittpunkte der korrigierten Kantengeraden bestimmt.

Abbildung 2.4a auf S. 13 zeigt ein Beispiel für ein zur Kalibrierung verwendetes Kamerabild. Die einzelnen Schritte des Kalibrierungsverfahrens werden im Folgenden an diesem Beispiel verdeutlicht.

Algorithmus zu Bestimmung der Referenzpunkte

Im Folgenden wird das für die Bestimmung der Koordinaten der Referenzpunkte entwickelte Verfahren als Algorithmus beschrieben. Eine detaillierte Beschreibung der einzelnen Schritte ist in den folgenden Abschnitten zu finden.

1. Aus dem Quellbild wird mit dem Canny-Algorithmus ein Kantenbild und ein Gradientenrichtungsbild erzeugt.
2. Alle Punkte des Kantenbildes, die außerhalb der *Region of Interest* liegen, werden auf Null gesetzt.
3. Mit Hilfe der Hough-Transformation werden die ungefähren Positionen und Richtungen der acht Kantengeraden des Kalibrierungsmusters bestimmt.
4. Für jede der in Schritt 3 gefundenen Geraden:
    a) Es werden die Start und Endpunkte von zu der Gerade senkrechten Geraden mit einer bestimmten Breite und einem festen Abstand zueinander berechnet.
    b) Für jede dieser Senkrechten:
        i. Mittels bilinearer Interpolation werden die Intensitätswerte der auf der Senkrechten liegenden Punkte berechnet.
        ii. Innerhalb des so berechneten Intensitätsverlaufes wird die exakte Position des Wendepunktes bestimmt.

iii. Aus der Position und Richtung der Senkrechten und der Position des Wendepunktes innerhalb des Intensitätsverlaufes werden die Bildkoordinaten des Wendepunktes berechnet.

c) Aus den in Schritt 4b bestimmten Kantenpunkten wird eine Ausgleichsgerade berechnet.

5. Die in Schritt 4 berechneten Ausgleichsgeraden werden anhand ihrer Steigungswinkel als entweder horizontale oder vertikale Geraden kategorisiert.

6. Die jeweils vier horizontalen und vertikalen Geraden werden aufsteigend nach ihrem Abstand zum Ursprung sortiert.

7. Es werden die Schnittpunkte der horizontalen und vertikalen Geraden berechnet, in denen sich eine äußere mit einer äußeren bzw. eine innere mit einer inneren Gerade schneidet. Die so bestimmten Schnittpunkte sind die Bildkoordinaten der acht Referenzpunkte.

Schritt 1: Näherungsweise Bestimmung der Kantenposition

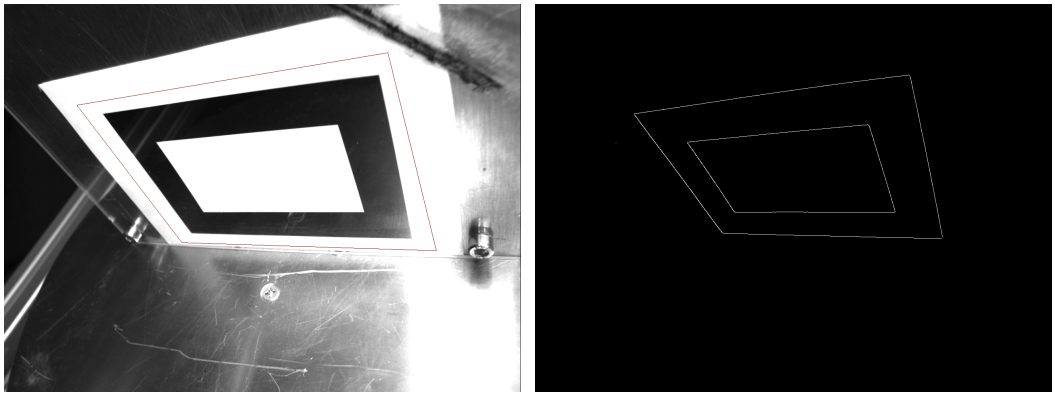
Für die Bestimmung der näherungsweisen Kantenpositionen wird die Hough-Transformation verwendet. Die hierfür benötigten Kanteninformationen werden mithilfe des Canny-Algorithmus berechnet. Das durch den Canny-Algorithmus erzeugte Kantenbild ist in Abbildung 2.4e auf S. 13 dargestellt. In diesem Bild lässt sich auch erkennen, dass hierbei noch zu viele Kanten, also solche, die nicht zu dem Kalibrierungsmuster gehören, erkannt werden. Da es nur sehr schwer möglich wäre solche Probleme zu vermeiden bzw. automatisch zu erkennen, wird dem Kalibrierungsprogramm die *„Region of interest“* (ROI) angegeben. Hierbei handelt es sich um ein Polygon, das den Bereich des Bildes umschließt, der bei der Kalibrierung berücksichtigt werden soll. Bereiche ausserhalb dieses Polygons werden in allen Schritten der Kalibrierung ignoriert (*„Clipping“*). Abbildung 4.3 auf S. 24 zeigt die ROI in dem Beispielbild und das Kantenbild nach dem Clipping.

In diesem Kantenbild können nun mittels Hough-Transformation (siehe Abschnitt 2.2.3 auf S.13) die signifikanten Geraden bestimmt werden. Die so gefundenen Geraden sind in Abbildung 4.4 auf S. 25 dargestellt.

Schritt 2: Bestimmung der exakten Kantenposition

Im nun folgenden Schritt werden die von der Hough-Transformation nur ungenau bestimmten Geraden mit einem speziell hierfür entwickelten Verfahren einer Korrektur unterzogen, um die exakte Position der Kante zu bestimmen. Hierzu werden zunächst in festen Abständen Senkrechten auf jeder der näherungsweise bestimmten Geraden errichtet[22]. Mittels bilinearer

[22] Sowohl der Abstand zwischen den Senkrechten als auch die Länge können dem Kalibrierungsprogramm als Parameter übergeben werden.



(a) Kalibrierungsbild mit eingezeichneter ROI **(b)** Kantenbild nach Clipping

Abbildung 4.3.: Kalibrierungsbild mit *Region of interest*

Interpolation kann der Intensitätsverlauf entlang dieser Senkrechten berechnet werden. Dieses Vorgehen wird auch als *„Edge-Sampling“* bezeichnet.

Innerhalb des dadurch berechneten Intensitätsverlaufes (*„Sample“*) kann nun die exakte Position der Kante bestimmt werden. Diese wird hierbei definiert als der Wendepunkt des Intensitätsverlaufes, also die Nullstelle der zweiten Ableitung, die mithilfe des LaPlace-Operators berechnet werden kann. Zwei Beispiele für solche Intensitätsverläufe und deren zweite Ableitungen sind in Abbildung 4.5 gegeben.

Um die genaue Position eines Kantenpunktes innerhalb eines Samples zu ermitteln, werden zunächst die Intensitätswerte sowie die zweite Ableitung eines Samples untersucht. Da der Intensitätsverlauf aufgrund von Rauschen in der Regel mehr als einen Wendepunkt aufweist, wird derjenige Nulldurchgang der zweiten Ableitung gesucht, der zwischen ihrem Maximal- und Minimalwert liegt. Die exakte Position des Nulldurchganges innerhalb des Samples wird mittels linearer Interpolation berechnet.

In wenigen ungünstigen Fällen kann es vorkommen, dass die Senkrechte eine andere Kante oder einen Eckpunkt des Kalibrierungsmusters schneidet. Um diese Fälle zu erkennen und damit Fehler bei der Kantenkorrektur zu verhindern, werden noch eine Reihe von Überprüfungen durchgeführt. In der folgenden Liste sind alle Bedingungen, die erfüllt sein müssen, um einen gefundenen Kantenpunkt zu verifizieren, aufgeführt.

- Die zweite Ableitung des Intensitätsverlaufes muss mindestens einen Nulldurchgang aufweisen.
- Maximum und Minimum der zweiten Ableitung[23] dürfen nicht zu weit auseinander liegen.

[23] Maximum und Minimum der zweiten Ableitung kennzeichnen den oberen und unteren Wendepunkt der ersten Ableitung und damit den Anfangs- und Endpunkt des Kantenverlaufes

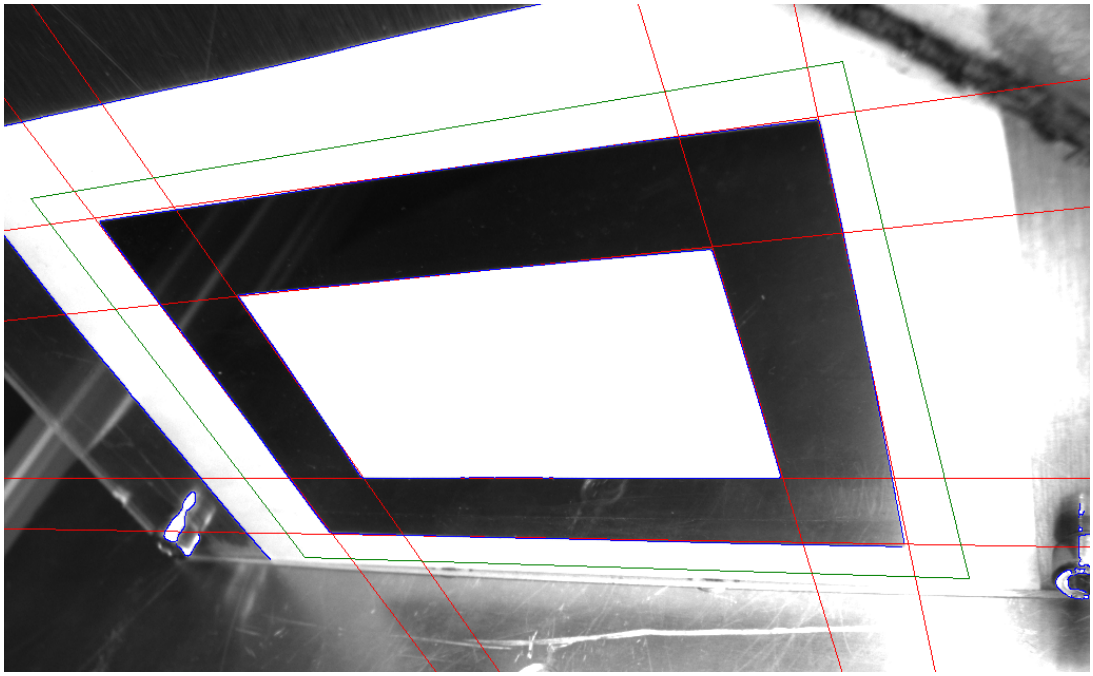


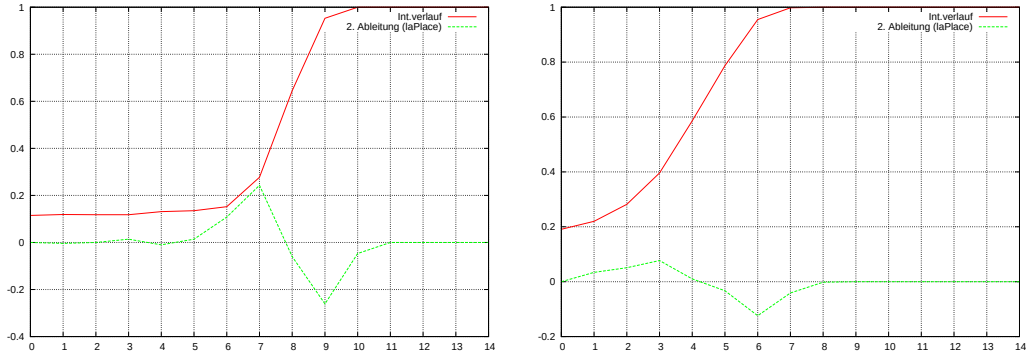
Abbildung 4.4.: Kalibrierungsbild mit näherungsweise bestimmten Kantengeraden: BLAU: durch den Canny-Algorithmus ermittelte Kantenpunkte; GRÜN: Region of interest; ROT: lokale Maxima des Hough-Raums nach der Hough-Transformation.

- Die Differenz von Minimal- und Maximalwert der zweiten Ableitung darf nicht zu klein sein.
- Die Differenz der Intensitätswerte an den Stellen des Minimums und Maximums der zweiten Ableitung muss ausreichend groß sein.
- Der Winkel zwischen der Richtung des Samples und der Gradientenrichtung (diese wurde bereits im ersten Schritt durch den Canny-Operator berechnet) darf nicht zu groß sein.

Die Vergleichswerte für diese Bedingungen können dem Kalibrierungsprogramm als Parameter übergeben werden, jedoch hat es sich gezeigt, dass die Voreinstellungen in den meisten Fällen zu sehr guten Ergebnissen führen.

Die Koordinaten des gefundenen Kantenpunktes können jetzt aus der Position und Richtung des Samples und der Kantenposition innerhalb dieses Samples berechnet werden.

Für jede der im ersten Schritt bestimmten Kantengeraden kann nun aus den n gefundenen Kantenpunkten $\vec{p}_i, \quad i = 1, \ldots, n$ eine Ausgleichsgerade berechnet werden. Basierend auf der Modellgleichung $y = a_1 x + a_0$ für die zu bestimmende Gerade und den bekannten Punkten



(a) Intensitätsverlauf einer scharfen Kante **(b)** Intensitätsverlauf einer schräg geschnittenen Kante

Abbildung 4.5.: Beispiele für Intensitätsverläufe: ROT: Intesitätsverlauf senkrecht zu der Kantengeraden; GRÜN: diskrete zweite Ableitung des Intensitätsverlaufes.

$\vec{p}_i = \begin{pmatrix} x_i & y_i \end{pmatrix}^T$ ergibt sich ein lineares Ausgleichsproblem mit

$$\begin{pmatrix} x_0 & 1 \\ x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_0 \end{pmatrix} \approx \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{4.1}$$

Die Lösung dieses Ausgleichsproblems im Sinne der kleinsten Fehlerquadrate wird mithilfe der QR-Zerlegung (siehe Abschnitt 2.1.2) berechnet. Somit ist eine optimale Näherung an die Parameter a_0 und a_1 der Geradengleichung der korrigierten Kantengerade bekannt.

Abbildung 4.6 auf S. 27 zeigt das Kalibrierungsbild in dem die näherungsweise bestimmten Geraden, die Edge-Samples, die gefundenen Kantenpunkte sowie die korrigierten Kantengeraden eingezeichnet sind.

Schritt 3: Sortieren der Geraden und Bestimmung der Schnittpunkte

Um nun die Schnittpunkte der Geraden und damit die Eckpunkte des Kalibrierungsmusters bestimmen zu können, müssen die korrigierten Geraden zunächst in horizontale und vertikale Geraden gruppiert und anschließend von oben nach unten bzw. von links nach rechts sortiert werden.

Um die Geraden als entweder horizontal oder vertikal zu kategorisieren ist es naheliegend, sie Anhand ihrer Steigung so zu gruppieren, dass Geraden mit ähnlicher Steigung in jeweils einer Gruppe zusammengefasst werden. Anstelle der Steigungen m_i der Geraden werden hierzu

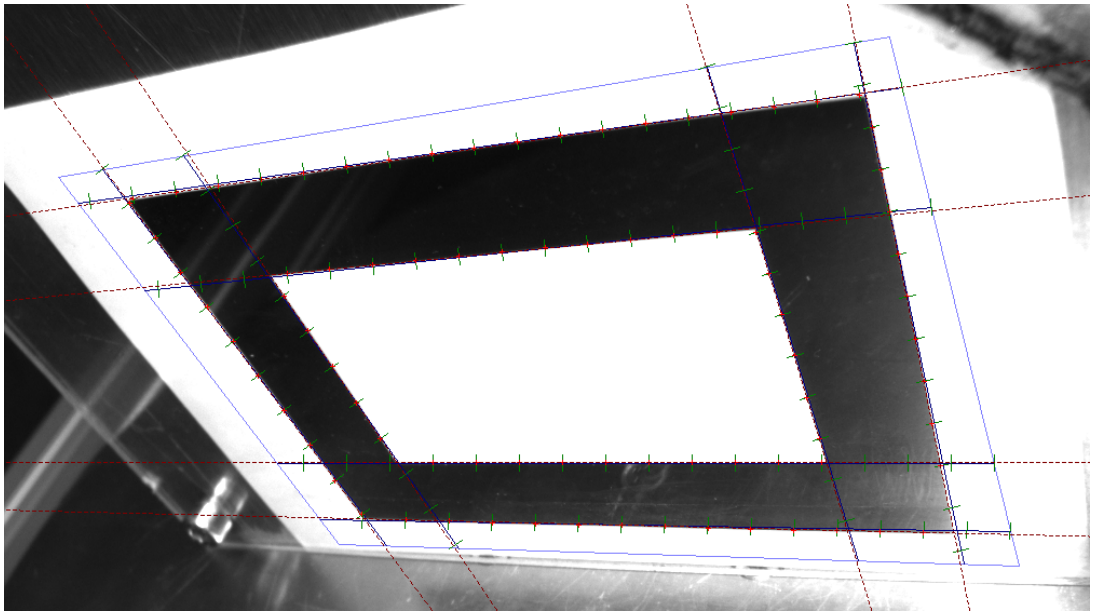


Abbildung 4.6.: Veranschaulichung des Kantenkorrekturverfahrens: BLAU: näherungsweise bestimmte Geraden; GRÜN Edge-Samples; ROT: Kantenpunkte; DUNKELROT/GESTRICHELT: korrigierte Geraden.

jedoch die Steigungswinkel $\alpha_i = \arctan(m_i), \quad \alpha_i \in [-\pi, \pi]$ verwendet, da sich mit ihnen die Ähnlichkeit zweier Steigungen unabhängig von ihrem Betrag[24] beschreiben lässt.

Es lässt sich zeigen, dass sich die acht Geraden unabhängig vom Blickwinkel der Kamera und damit unabhängig vom Grad der perspektivischen Verzerrung immer anhand des Mittelwertes der Beträge ihrer Steigungswinkel $\alpha_{avg} = \frac{1}{n}\sum_{i=0}^{n} |\alpha_i|$ in zwei Gruppen von je vier Geraden einordnen lassen. Eine Gerade wird als vertikale Gerade bezeichnet, falls $|\alpha_i| \geq \alpha_{avg}$, für horizontale Geraden gilt $|\alpha_i| < \alpha_{avg}$.

Bevor nun die Schnittpunkte berechnet werden können, müssen die Geraden noch von links nach rechts bzw. von oben nach unten sortiert werden. Für diese Sortierung wird der Abstand der Geraden zum Ursprung des Koordinatensystems als Vergleichskriterium verwendet.

Um abschließend die Koordinaten der Eckpunkte des Kalibrierungsmusters zu erhalten, müssen die Schnittpunkte der horizontalen mit den vertikalen Geraden berechnet werden. Von den insgesamt 16 Schnittpunkten werden jedoch nur die 8 Schnittpunkte benötigt, in denen sich eine der äußeren horizontalen Geraden mit einer der äußeren vertikalen Geraden bzw. eine der inneren horizontalen Geraden mit einer der inneren vertikalen Geraden schneidet.

[24] Die Steigungen zweier vertikaler Geraden mit Steigungswerten von $m_1 = 100$ und $m_2 = 1000$ ($\Delta m = 900$) sind sich genauso ähnlich wie die zweier horizontaler Geraden mit Steigungen $m_3 = \frac{1}{100}$ und $m_4 = \frac{1}{1000}$ ($\Delta m = 0.009$).

Diese acht Schnittpunkte müssen von links nach rechts und von oben nach unten durchnummeriert[25] werden, um den Abgleich mit den bekannten Koordinaten in der Laserebene zu ermöglichen. Dies wird durch die Reihenfolge bei der Berechnung der Schnittpunkte sichergestellt[26].

4.1.3. Berechnung der Transformationsmatrix

Wie bereits am Anfang dieses Kapitels auf S.19 beschrieben wurde, ist es das Ziel der Kalibrierung, die Transformationsmatrix einer perspektivischen Transformation zu bestimmen, mit der sich Koordinaten in der Bildebene auf Koordinaten in der Laserebene abbilden lassen. Wie diese Transformationsmatrix aus den nun vorliegenden Referenzpunktpaaren berechnet wird, soll im Folgenden gezeigt werden.

Die Gleichung für die Berechnung der Koordinaten eines Punktes $\vec{l}$ in der Laserebene aus den Koordinaten eines Punktes $\vec{c}$ in der Bildebene lautet

$$\vec{l^*} = \mathbf{H}\vec{c^*} \quad \Leftrightarrow \quad \begin{pmatrix} x_l^* \\ y_l^* \\ h_l^* \end{pmatrix} = \begin{pmatrix} h_{11} & \cdots & h_{13} \\ \vdots & \ddots & \vdots \\ h_{31} & \cdots & h_{33} \end{pmatrix} \begin{pmatrix} x_c \\ y_c \\ 1 \end{pmatrix} \quad \text{mit} \begin{cases} x_l = \frac{x_l^*}{h_l^*} \\ y_l = \frac{y_l^*}{h_l^*} \end{cases} \tag{4.2}$$

Die Anzahl der in dieser Gleichung auftretenden Unbekannten kann noch auf acht Unbekannte reduziert werden, indem das gesamte Gleichungssystem mit $\frac{1}{h_{33}}$ multipliziert wird. Dadurch erhalten wir eine neue Transformationsmatrix $\mathbf{B}$ mit $b_{ik} = \frac{h_{ik}}{h_{33}}$ und können obige Gleichung umschreiben in[27]

$$\vec{l^*} = \mathbf{B}\vec{c^*} \quad \Leftrightarrow \quad \begin{pmatrix} x_l^* \\ y_l^* \\ h_l^* \end{pmatrix} = \begin{pmatrix} b_{11} & \cdots & b_{13} \\ \vdots & \ddots & b_{23} \\ b_{31} & b_{32} & 1 \end{pmatrix} \begin{pmatrix} x_c \\ y_c \\ 1 \end{pmatrix} \quad \text{mit} \begin{cases} x_l = \frac{x_l^*}{h_l^*} \\ y_l = \frac{y_l^*}{h_l^*} \\ b_{ik} = \frac{h_{ik}}{h_{33}} \end{cases} \tag{4.3}$$

Die acht Werte für b_{ik} sind also die aus den Referenzpunktpaaren zu bestimmenden Unbekannten. Um diese mit der Methode der kleinsten Quadrate bestimmen zu können, wird ein lineares Gleichungssystem $\mathbf{M}\vec{x} = \vec{b}$ benötigt, in dem die Parameter b_{ik} nur noch in dem Vektor der Unbekannten $\vec{x}$ auftauchen.

---

[25] Die Nummerierung muss bezogen auf das Kalibrierungsmuster erfolgen, da die Koordinaten der Schnittpunkte in der Bildebene aufgrund der perspektivischen Verzerrung unter Umständen eine andere Reihenfolge ergeben könnten.

[26] Unter Auslassung der nicht benötigten Schnittpunkte werden die Schnittpunkte von oben nach unten und von links nach rechts nacheinander berechnet.

[27] Aufgrund der homogenen Gleichheit (vgl. Abschnitt 2.1.1 auf S.7) gilt $\frac{1}{h_{33}}\vec{l^*} \sim \vec{l^*}$. Beide Koordinaten beschreiben also denselben Punkt im euklidischen Raum. Daher kann die Multiplikation in diesem Fall vernachlässigt werden.

Um dies zu erreichen wird die obige Gleichung Komponentenweise aufgeschrieben und die Umwandlung der homogenen Koordinaten von $\vec{l}^*$ in die euklidischen Koordinaten von $\vec{l}$ mit $x_l = \frac{x_l^*}{h_l^*}$ und $y_l = \frac{y_l^*}{h_l^*}$ durchgeführt. Für die Komponenten von $\vec{l}$ ergibt sich hiermit:

$$\begin{aligned} x_l &= \frac{x_l^*}{h_l^*} = \frac{b_{11}x_c + b_{12}y_c + b_{13}}{b_{31}x_c + b_{32}y_c + 1} \\ \Leftrightarrow \quad x_l(b_{31}x_c + b_{32}y_c + 1) &= b_{11}x_c + b_{12}y_c + b_{13} \\ \Leftrightarrow \quad x_l &= b_{11}x_c + b_{12}y_c + b_{13} - x_c x_l b_{31} - x_l y_c b_{32} \end{aligned} \tag{4.4a}$$

sowie

$$\begin{aligned} y_l &= \frac{y_l^*}{h_l^*} = \frac{b_{21}x_c + b_{22}y_c + b_{23}}{b_{31}x_c + b_{32}y_c + 1} \\ \Leftrightarrow \quad y_l(b_{31}x_c + b_{32}y_c + 1) &= b_{21}x_c + b_{22}y_c + b_{23} \\ \Leftrightarrow \quad y_l &= b_{21}x_c + b_{22}y_c + b_{23} - x_c y_l b_{31} - y_l y_c b_{32} \end{aligned} \tag{4.4b}$$

Mit den beiden Gleichungen (4.4a) und (4.4b) lässt sich nun ein Gleichungssystem zur Bestimmung der acht Unbekannten b_{ik} aufstellen. Dabei entstehen aus jedem der n Referenzpunktpaare zwei Gleichungen. Daher werden zur Berechnung dieser Unbekannten $n \geq 4$ linear unabhängige Paare von zusammengehörigen Vektoren $\vec{c}_k = \begin{pmatrix} x_{c_k} & y_{c_k} \end{pmatrix}^T$ und $\vec{l}_k = \begin{pmatrix} x_{l_k} & y_{l_k} \end{pmatrix}^T$, mit $k = 1, \ldots, n$ benötigt, um eine eindeutige Lösung zu erhalten.

Dieses lineare Ausgleichsproblem zur Bestimmung der Unbekannten b_{ik} lässt sich in Matrixschreibweise nun folgendermaßen darstellen:

$$\mathbf{M}\vec{x} = \vec{b} \quad \Leftrightarrow \quad \begin{pmatrix} x_{c_1} & y_{c_1} & 1 & 0 & 0 & 0 & -x_{l_1}x_{c_1} & -x_{l_1}y_{c_1} \\ 0 & 0 & 0 & x_{c_1} & y_{c_1} & 1 & -y_{l_1}x_{c_1} & -y_{l_1}y_{c_1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{c_n} & y_{c_n} & 1 & 0 & 0 & 0 & -x_{l_n}x_{c_n} & -x_{l_n}y_{c_n} \\ 0 & 0 & 0 & x_{c_n} & y_{c_n} & 1 & -y_{l_n}x_{c_n} & -y_{l_n}y_{c_n} \end{pmatrix} \begin{pmatrix} b_{11} \\ b_{12} \\ b_{13} \\ b_{21} \\ b_{22} \\ b_{23} \\ b_{31} \\ b_{32} \end{pmatrix} \approx \begin{pmatrix} x_{l_1} \\ y_{l_1} \\ \vdots \\ x_{l_n} \\ y_{l_n} \end{pmatrix} \tag{4.5}$$

Für die Lösung dieses Gleichungssystems werden alle acht bekannten Referenzpunktpaare in diese Matrix eingesetzt, wodurch insgesamt 16 Gleichungen zur Bestimmung der 8 Unbekannten zur Verfügung stehen.

Da dieses Ausgleichsproblem oftmals schlecht konditioniert ist, und weniger rechenintensive Verfahren dabei an ihre Grenzen stoßen, wird die Lösung durch Berechnung der Pseudoinversen $\mathbf{M}^\dagger$ mithilfe der Singulärwertzerlegung ermittelt:

$$\vec{x} = \mathbf{M}^\dagger \vec{b} \quad \Leftrightarrow \quad \vec{x} = \mathbf{V}\Sigma^\dagger\mathbf{U}^T \qquad \text{mit } SVD(\mathbf{M}) = \mathbf{U}\Sigma\mathbf{V}^T \tag{4.6}$$

4.2. Objektvermessung

Nachdem die Transformationsmatrix durch die Kalibrierung berechnet wurde, kann nun die Objektvermessung durchgeführt werden. Hierzu müssen in einem Kamerabild die von dem Laser beleuchteten Punkte der Oberfläche erkannt werden und aus deren Koordinaten in der Bildebene $\vec{c}$ die entsprechenden Koordinaten $\vec{l}$ in der Laserebene berechnet werden. Diese Koordinaten werden als Ergebnis der Objektvermessung zurückgegeben.

4.2.1. Vorbereitung

Damit die Objektvermessung durchgeführt werden kann, muss zunächst das zu vermessende Objekt so auf den Drehteller gestellt werden, dass die Laserlinie auf das Objekt projiziert wird.

Um die spätere Verarbeitung der Bilder zu vereinfachen, wird die Messvorrichtung nun abgedunkelt und das Objektiv der Kamera so eingestellt, dass nur noch die von dem Laser beleuchteten Punkte in dem Bild zu scharf zu erkennen sind. Gegebenenfalls könnte hierzu auch ein monochromatischer Filter[28] eingesetzt werden. Das Ergebnis ist ein Bild, wie es in den Abbildungen 4.7b und 4.8 gezeigt ist.

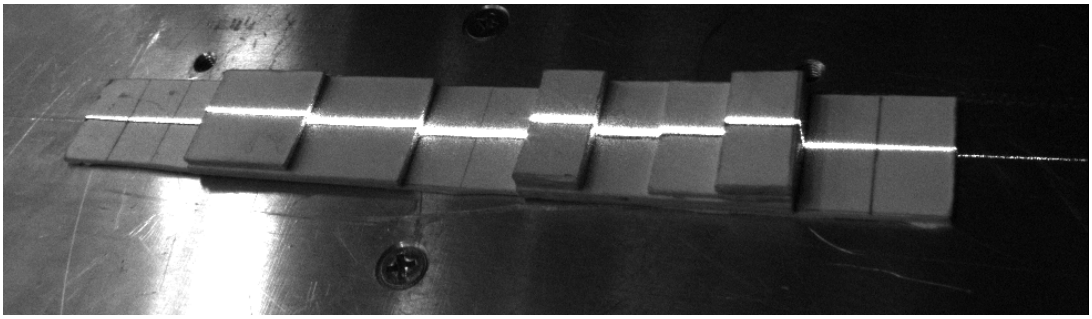
4.2.2. Vorverarbeitung des Kamerabildes

Um die durch den Laser beleuchteten Punkte im Kamerabild zu finden, wird aus dem Quellbild zunächst mithilfe des Canny-Algorithmus ein Kantenbild erzeugt.

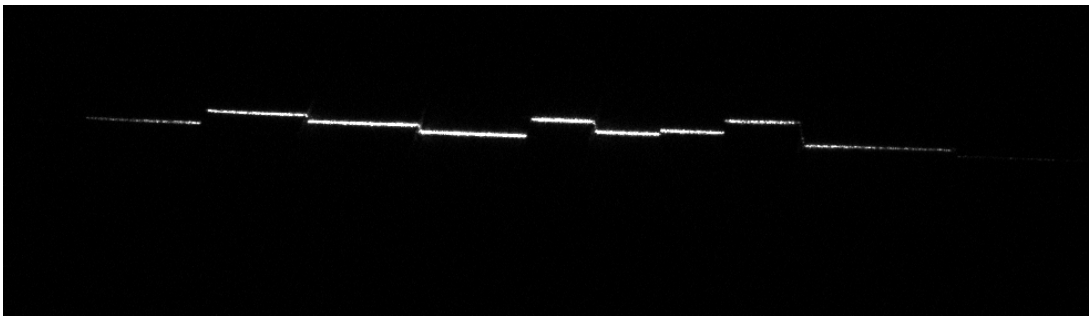
Da es bei ungünstigen Bedingungen je nach aktueller Ausrichtung und Oberflächenbeschaffenheit des Objektes zu Bildstörungen durch diffuse Reflektionen kommen kann, wird basierend auf diesem Kantenbild zunächst eine Vorverarbeitung bzw. Filterung des Bildes durchgeführt. Dabei wird für jedes einzelne der Pixel, die als Kantenpunkte gefunden wurden, überprüft, ob es sich dabei tatsächlich um einen Kantenpunkt der Laserlinie handelt oder ob der Punkt aus einer Bildstörung resultiert.

Die Grundidee des hierzu entwickelten Verfahrens ist, dass die Umgebung von jedem der gefundenen Kantenpunkte auf ihre Eigenschaften bezüglich der Linienhaftigkeit untersucht wird. Die Auswertbarkeit der abgebildeten Laserlinie hängt maßgeblich von der Breite ihrer Abbildung ab. Sie ist nur dann sinnvoll auswertbar, wenn sie eine bestimmte Breite von wenigen Pixeln nicht überschreitet. Die durch diffuse Reflektionen verursachten Störungen, wie sie beispielsweise in Abbildung 4.8 auf S. 32 gezeigt sind, lassen sich ebenfalls an größeren

[28] Monochromatische Filter sind Filter, die vor das Objektiv einer Kamera geschraubt werden können und nur einen sehr kleinen Bereich der Wellenlängen des sichtbaren Lichtes durchlassen.



(a) Testobjekt, bei normalem Umgebungslicht mit hoher Empfindlichkeit aufgenommen



(b) Testobjekt, in abgedunkeltem Scanner mit Messeinstellungen aufgenommen

Abbildung 4.7.: Beispielbilder Objektvermessung

Bereichen mit hoher Intensität erkennen. Es werden also diejenigen Pixel von der Weiterverarbeitung ausgeschlossen, die an der Kante eines zu großen Bereiches mit hoher Intensität liegen.

Um dies zu erreichen, wird die nähere Umgebung von jedem der gefundenen Kantenpunkte untersucht. Hierzu werden, ähnlich wie bei der Kalibrierung, zunächst die Intensitätswerte einer bestimmten Anzahl Pixel entlang der Gradientenrichtung mittels bilinearer Interpolation berechnet. Auf die so berechnete Bildzeile wird nun der in Abbildung 4.9 auf S. 32 dargestellte Filterkern angewendet. Das Ergebnis dieser Faltung wird jedoch nicht, wie bei der linearen Filterung üblich, in ein Zielbild geschrieben, sondern als Bewertungszahl für das aktuelle Pixel verwendet. Ist dieser Wert kleiner oder gleich Null, so wird das Pixel nicht als Kantenpunkt der Laserlinie gewertet und in dem Kantenbild auf Null gesetzt. Ist der Wert größer Null, so wird der Punkt beibehalten.

Das Ergebnis der Anwendung dieses Filters ist in Abbildung 4.10 auf S. 32 dargestellt.

4.2.3. Extraktion der Oberflächenpunkte aus dem Bild

Nachdem die Vorverarbeitung durchgeführt wurde, werden nun für alle nicht gefilterten Kantenpunkte die Mittelpunkte der Laserlinie bestimmt. Hierzu werden in den bereits im vorigen

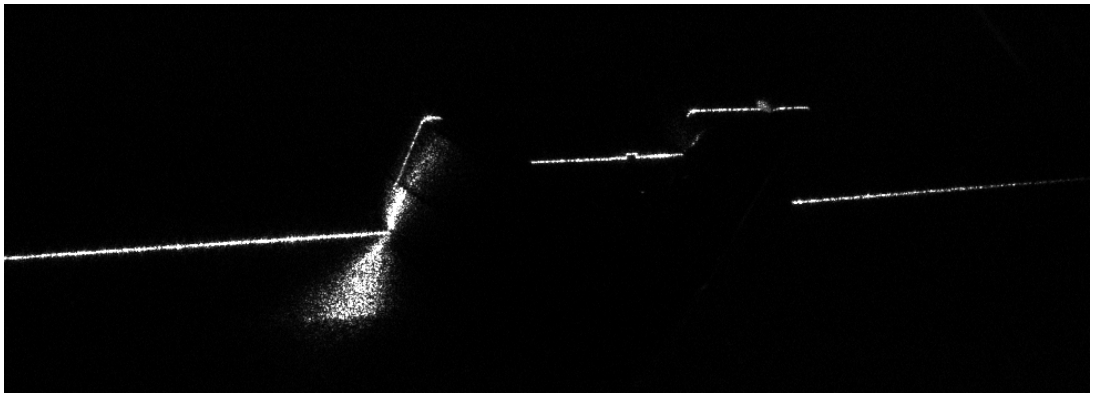


Abbildung 4.8.: Beispiel für diffuse Reflektionen

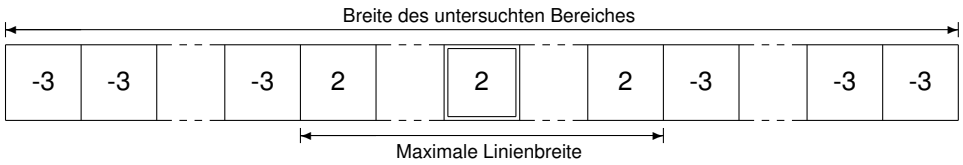


Abbildung 4.9.: Filtermaske zur Unterdrückung von diffusen Reflektionen

Schritt für jeden der Kantenpunkte berechneten Linienquerschnitten (siehe Abbildung 4.11 auf S. 33) die genauen Positionen des Maximums bestimmt. Diese lässt sich durch Bestimmung der Nullstelle der diskreten ersten Ableitung der Intensitätswerte berechnen. Die subpixelgenaue Position der Nullstelle wird per linearer Interpolation berechnet.

4.2.4. Berechnung der Koordinaten in der Laserebene

Im letzten Schritt müssen nun noch aus den im Kamerabild ermittelten Koordinaten $\vec{c}$ die entsprechenden Koordinaten $\vec{l}$ der Laserebene berechnet werden. Hierzu müssen die Koordinaten

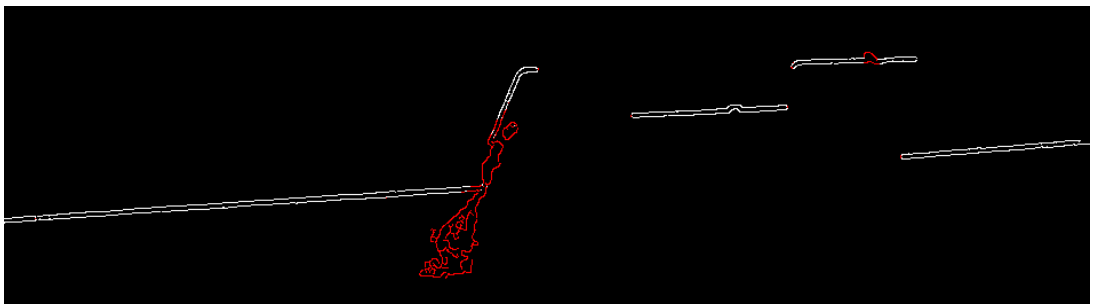


Abbildung 4.10.: Beispiel Reflektionsfilter: ROT: gefilterte Kantenpunkte; WEISS: beibehaltene Kantenpunkte

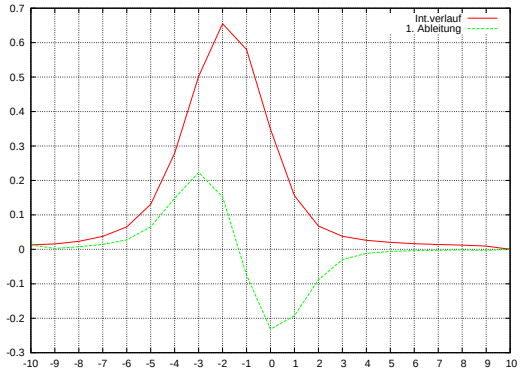


Abbildung 4.11.: Beispiel für einen Querschnitt durch die Laserlinie: ROT: Intensitätsverlauf; GRÜN: diskrete erste Ableitung der Intensitätswerte

lediglich mit der durch die Kalibrierung bestimmten Transformationsmatrix $\mathbf{B}$ multipliziert (vgl. Gleichung (4.3) in Abschnitt 4.1.3) und das Ergebnis in euklidische Koordinaten umgewandelt werden:

$$\vec{l^*} = \mathbf{B}\vec{c^*} \qquad \text{mit } \vec{c^*} = \begin{pmatrix} c_x & c_y & 1 \end{pmatrix}^T \tag{4.7}$$

5. Messergebnisse

In diesem Kapitel werden die Ergebnisse der testweise durchgeführten Messungen ausgewertet und Angaben über die Genauigkeit der Messungen gemacht.

Inhalt

5.1. Genauigkeit der Kalibrierung 34
5.2. Messgenauigkeit 35

5.1. Genauigkeit der Kalibrierung

Die maximal erzielbare Genauigkeit lässt sich näherungsweise als die Genauigkeit der Kalibrierung berechnen. Für jedes der n Paare von Referenzpunkten $(\vec{c}_i, \vec{l}_i)$ wird hierzu der Punkt $\vec{c}_i$ in der Bildebene auf einen Punkt $\vec{l}'_i = \mathbf{B}\vec{c}_i$ in der Laserebene transformiert. Der Abstand des so berechneten Punktes $\vec{l}'_i$ von dem tatsächlichen Punkt $\vec{l}_i$ ist die durch die Approximation der Transformationsmatrix verursachte Abweichung. Die mittlere Abweichung ergibt sich daraus mit

$$E_{avg} = \frac{1}{n} \sum_{i=0}^{n} |\vec{l}_i - \mathbf{B}\vec{c}_i| \tag{5.1}$$

In den durchgeführten Versuchen lagen die Werte der durchschnittlichen Genauigkeit im Bereich von $0.09mm$ bis $0.24mm$. Diese Genauigkeit ist von zahlreichen Faktoren abhängig. Den vermutlich entscheidenden Einfluss hat die Ausrichtung der Kamera relativ zu der Laserebene und damit die Stärke der auftretenden perspektivischen Verzerrungen. Hierüber lässt sich aber anhand der wenigen durchgeführten Messungen keine genaue Aussage treffen. Einen weiteren Einfluss haben auch die intrinsischen Parameter der Kamera[29], die in dieser Arbeit nicht berücksichtigt wurden, sowie die Wahl der Kalibrierungsparameter[30].

[29] Die intrinsischen Parameter der Kamera beschreiben die inneren Eigenschaften der Kamera und damit die Abbildung der Bildebene auf die Pixelebene. Werden diese Parameter berücksichtigt, kann die durch die Kamera verursachte Linsenverzeichnung im Bild ausgeglichen werden.

[30] Insbesondere hat der Abstand der Edge-Samples (vgl. Abschnitt 4.1.2 auf S.23) und damit die Anzahl der für die Approximation der Kantengeraden verwendeten Punkte einen Einfluss auf die maximale Genauigkeit.

5.2. Messgenauigkeit

Die Messgenauigkeit des Systems ist nicht nur von der Genauigkeit der Kalibrierung sondern auch von einigen anderen Faktoren abhängig. Zu diesen Faktoren gehört insbesondere die Genauigkeit, mit der die Koordinaten der Mittelpunkte der Laserlinie bestimmt werden.

Um diese Genauigkeiten zu untersuchen, wurde ein Messobjekt hergestellt (siehe Abbildung 4.7 auf S. 31), dessen Abmessungen und Eigenschaften hinreichend genau bekannt sind. Das aus insgesamt 1356 Punkten bestehende Ergebnis der Vermessung dieses Objektes ist in Abbildung 5.1 dargestellt.

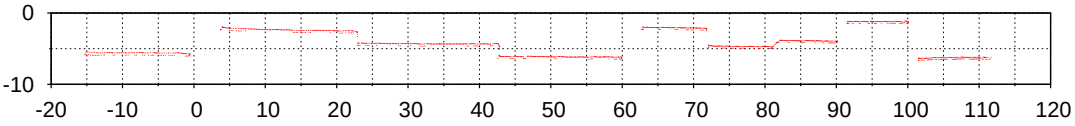


Abbildung 5.1.: Ergebnis der Vermessung des Testobjektes. Die Koordinaten sind in mm angegeben; Ursprung des Koordinatensystems ist der untere linke Eckpunkt des Kalibrierungsmusters (vgl. Abschnitt 4.1)

Um nun einige Kennzahlen für die Messgenauigkeit aus diesen Punkten zu berechnen, wurde zunächst die sogenannte Wiederholgenauigkeit der Messung der y-Koordinaten bestimmt. Hierzu wurde das Ergebnis in mehrere Bereiche eingeteilt, in denen die Höhe des Messobjektes jeweils konstant ist. Für jeden dieser Bereiche wurden nun die wichtigsten statistischen Kennzahlen für die y-Werte berechnet. Die Ergebnisse dieser Auswertung sind in Tabelle 5.1 zusammengefasst.

x	n	y_m	y_{min}	y_{max}	$\bar{y}_{arith}$	σ_y
$[-15,0]$	54	$0\ mm$	$-6.0326\ mm$	$-5.5028\ mm$	$-5.7090\ mm$	$0.1736\ mm$
$[5,20]$	138	$4\ mm$	$-2.7676\ mm$	$-2.1286\ mm$	$-2.3871\ mm$	$0.1356\ mm$
$[25,40]$	149	$2\ mm$	$-4.6542\ mm$	$-4.2318\ mm$	$-4.3375\ mm$	$0.0956\ mm$
$[45,60]$	167	$0\ mm$	$-6.4873\ mm$	$-6.0600\ mm$	$-6.1749\ mm$	$0.0956\ mm$
$[62,70]$	91	$4.5\ mm$	$-2.3689\ mm$	$-1.9398\ mm$	$-2.0933\ mm$	$0.1136\ mm$
$[75,80]$	58	$2\ mm$	$-5.0034\ mm$	$-4.6600\ mm$	$-4.7387\ mm$	$0.1103\ mm$
$[83,90]$	90	$3\ mm$	$-4.1963\ mm$	$-3.8547\ mm$	$-3.9499\ mm$	$0.1102\ mm$
$[92,100]$	110	$5\ mm$	$-1.4695\ mm$	$-1.1196\ mm$	$-1.2459\ mm$	$0.1075\ mm$
$[103,110]$	101	$0\ mm$	$-6.5662\ mm$	$-6.1599\ mm$	$-6.3276\ mm$	$0.1259\ mm$

Tabelle 5.1.: Messgenauigkeit in y-Richtung: x gibt den untersuchten Bereich des Ergebnisses auf der x-Achse an; n ist die in diesem Bereich befindliche Anzahl an Messpunkten; y_m gibt den Erwartungswert, bezogen auf die niedrigste Oberfläche des Objektes an; y_{min} und y_{max} geben den maximalen und minimalen y-Wert an; $\bar{y}_{arith}$ und σ_y geben den arithmetischen Mittelwert und die Standardabweichung an.

An diesen Daten lässt sich gut erkennen, dass mit dem realisierten Messsystem sehr genaue Messungen möglich sind. Vergleicht man zum Beispiel die recht ungenau gemessenen Erwartungswerte mit den dazugehörigen Mittelwerten, stellt man fest, dass die maximale Abweichung kaum größer als ein halber Millimeter ist. Dieser Wert entspricht in etwa der Genauigkeit, mit der das Messobjekt von Hand vermessen werden konnte.

Die von links nach rechts kleiner werdenden Mittelwerte in den Bereichen, in denen der Erwartungswert gleich Null ist, lassen darauf schließen, dass die untere Kante des Kalibrierungsmusters bei der Kalibrierung nicht exakt parallel zu der Oberfläche des Drehtellers war. Anhand der Werte der Standardabweichungen und der maximalen Abweichung vom Mittelwert ($\sim 0.3\ mm$) lässt sich gut erkennen, dass die Objektvermessung unter optimalen Bedingungen (genaue Ausrichtung des Kalibrierungsmusters und des Lasers sowie optimale Einstellungen von Laser und Kamera) in der Lage ist, eine sehr hohe Genauigkeit zu erreichen.

6. Implementierung

In diesem Kapitel soll nun auf einige Implementierungsdetails der im Rahmen dieser Arbeit erstellten Software, die verwendeten Software-Bibliotheken sowie die notwendigen Schritte zum Kompilieren und Verwenden der Software eingegangen werden. Beschreibungen der Programmierschnittstellen sowie einige Beispiele für die Verwendung der Software sind im Anhang A zu finden.

Inhalt

6.1. Struktur 37
6.2. Verwendung der Software 38
6.2.1. Durchführung einer Kalibrierung 38
6.2.2. Durchführung einer Objektvermessung 39
6.3. Verwendete Software-Bibliotheken 40
6.3.1. LTILib 40
6.3.2. Boost C++ Libraries 40
6.4. Kompilieren des Projektes 41
6.4.1. Linux 42
6.4.2. Windows 42

6.1. Struktur

Die gesamte Software wurde als ein über die Kommandozeile ausführbares Programm realisiert. Beim Aufruf dieses Programmes wird über einen Parameter gesteuert, ob eine Kalibrierung oder eine Objektvermessung durchgeführt werden soll. Eine direkte Anbindung an die verwendete Firewire-Kamera wurde in diesem Programm nicht realisiert, so dass die zu bearbeitenden Bilder vorher mit separater Software[31] aufgenommen werden müssen. Die Eingabedateien müssen im JPEG, PNG oder BMP-format vorliegen.

Die Software besteht aus einer Datei, die mit der main-Funktion den Einstiegspunkt des Programms enthält sowie zwei Klassen, die für die Kalibrierungsfunktion bzw. die Objektvermessung zuständig sind.

[31] Für die Aufnahme der Bilder wurde die mit der Kamera mitgelieferte Software *„IC Capture 2.0“* von The Imaging Source verwendet (siehe `http://www.imagingcontrol.com`).

Diese beiden Klassen sind so konzipiert, dass Sie sich ohne Schwierigkeiten in anderen Projekten weiterverwenden lassen. Der grundsätzliche Aufbau und die Benennung der Methoden in diesen Klassen orientiert sich an den in der LTILib verwendeten Konventionen.

Alle Aufgaben, die nicht zu der Kernfunktionalität der Kalibrierung bzw. der Objektvermessung gehören, wie zum Beispiel die Verarbeitung der Kommandozeilenargumente oder das Laden der Bilddateien, wurden in die main-Funktion ausgelagert.

6.2. Verwendung der Software

Der Name des erstellten Programmes ist `lts.exe` (bzw. `lts` unter Linux). Nach dem Kompilieren ist diese executable im Verzeichnis `bin/` zu finden. Die folgenden Beispiele beschreiben die Verwendung des Programmes unter Windows. Die Unterschiede zu der Verwendung unter Linux sind aber eher gering.

Das Programm verfügt über eine eingebaute Kommandozeilen-Hilfe, die Informationen zu den Parametern und deren Format enthält. Sie wird durch einen Aufruf von `lts.exe --help` angezeigt.

6.2.1. Durchführung einer Kalibrierung

Um eine Kalibrierung vorzunehmen, müssen zunächst einmal die Einstellungen des Messaufbaus wie in Kapitel 4 beschrieben vorgenommen werden. Ist dies erledigt, wird in folgenden Schritten verfahren, um die Kalibrierung durchzuführen:

1. Als erstes wird mit IC Capture ein Bild des Kalibrierungsmusters aufgenommen und auf der Festplatte gespeichert. Der Dateiname muss beim Aufruf der Kalibrierung mit dem Parameter `-f` bzw. `--inputFile` angegeben werden.

2. Es wird eine Datei erstellt, die die Koordinaten der Referenzpunkte enthält. Diese werden in der Reihenfolge von oben links nach unten rechts in der Form `(x1, y1) (x2, y2)...` in diese Datei geschrieben. Der Inhalt dieser Datei könnte in etwa so aussehen:

   ```
   (0, 70)                    (100, 70)
           (15, 55) (85, 55)
           (15, 15) (85, 15)
   (0,  0)                    (100,  0)
   ```

   Der Name dieser Datei muss über den Parameter `-p` bzw. `--refPointsFile` angegeben werden.

3. Nun muss in dem Kalibrierungsbild die Region of interest bestimmt werden. Diese wird in demselben Format wie die Referenzpunkte notiert. Im Unterschied zu den Referenzpunkten wird die ROI jedoch über den Kommandozeilenparameter `--roi` angegeben.

Der Aufruf des Programms sieht also folgendermaßen aus:

```
lts.exe -m calibration -f kalibrg.bmp -p refpoints.dat
       --roi="(360,270)(1050,330)(850,660)(270,690)" -M tr.mat
```

Der Parameter `-m calibration` bzw. `--mode=calibration` gibt hierbei den Betriebsmodus an. Wird dieser Parameter weggelassen, wird eine Objektvermessung durchgeführt. Der letzte Parameter `-M` bzw. `--matrixFile` gibt den Namen einer Datei an, in der die berechnete Transformationsmatrix gespeichert werden soll. Anstelle der Dateinamen können selbstverständlich auch beliebige Pfadangaben verwendet werden.

6.2.2. Durchführung einer Objektvermessung

In diesem Beispiel wird davon ausgegangen, dass eine Kalibrierung, wie in Abschnitt 6.2.1 beschrieben, bereits vorgenommen wurde.

Wie bei der Kalibrierung wird auch bei der Objektvermessung damit begonnen, die notwendigen Einstellungen des Messaufbaus wie in Kapitel 4 beschrieben vorzunehmen. Nun wird wieder ein Bild von der Kamera aufgenommen und gespeichert. Wie schon bei der Kalibrierung wird der Dateiname des Bildes mit dem Parameter `-f` angegeben. Die Vermessung dieses Bildes kann nun mit folgendem Aufruf durchgeführt werden:

```
lts.exe -m calibration -f messg.bmp -M tr.mat -o points.dat
```

Der Parameter `-M` gibt hierbei wieder den Namen der Datei an, in der die von der Kalibrierung berechnete Transformationsmatrix gespeichert wurde. Der letzte Parameter `-o` bzw. `--outputFile` gibt den Namen einer Datei an, in der die Ergebnisse der Objektvermessung gespeichert werden. Nachdem die Objektvermessung durchgeführt wurde, sieht der Inhalt der Datei `points.dat` in etwa folgendermaßen aus:

```
# result file. contained data is of format 'x y'
-15.383964 -5.856324
-15.204492 -5.505241
-15.196514 -5.840114
...
111.387268 -6.471078
111.532547 -6.457536
111.663780 -6.204763
```

6.3. Verwendete Software-Bibliotheken

Für die Implementierung der Software wurden zwei frei verfügbare Software-Bibliotheken verwendet, die benötigt werden, um das Programm zu kompilieren. Beide Bibliotheken stehen jeweils als Windows- und Linux-Versionen zur Verfügung.

6.3.1. LTILib

Die LTILib (Alvarado u. a., 2007) ist eine Software-Bibliothek, die speziell für die Anwendung in den Bereichen der Computer-Vision, der digitalen Bildverarbeitung und der numerischen Mathematik konzipiert worden ist. Sie stellt eine große Anzahl von Datenstrukturen und häufig verwendeten Algorithmen aus diesen Bereichen zur Verfügung.

Die in dieser Arbeit verwendeten Algorithmen und Datenstrukturen, wie zum Beispiel die Datenstrukturen zur Darstellung von Bildern, Matrizen und Vektoren oder die Algorithmen für Kantenerkennung, Hough-Transformation sowie die Algorithmen zur Lösung der linearen Ausgleichsprobleme wurden von der LTILib bereitgestellt.

Die verwendete Version (ltilib-1.9.15) kann von der Homepage des LTILib-Projektes unter `https://sourceforge.net/projects/ltilib/` heruntergeladen werden. Die sehr umfangreiche API-Dokumentation findet sich unter (Alvarado u. a., 2007).

6.3.2. Boost C++ Libraries

Die Boost C++ Libraries(boost.org, 2007) sind eine Sammlung von zahlreichen Bibliotheken, die einige sehr hilfreiche Ergänzungen zu der C++ Standard Library und der Standard Template Library (STL) anbieten. Ihr Ziel ist es, durch die Bereitstellung von performanten und professionell entworfenen Lösungen für Probleme, die in sehr vielen Anwendungsbereichen auftreten, die Produktivität bei der Software-Erstellung zu steigern.

Die Entwicklung dieser Libraries wurde von der „C++ Standards Comittee Library Working Group“ gestartet. Sie gelten heute als Quasi-Standard und bieten Referenz-Implementierungen für kommende Versionen der C++ Standard Libraries an.

In dieser Arbeit wurden die folgenden der boost-libraries eingesetzt:

program_options Diese Library ermöglicht es auf sehr einfache Art und Weise, die Kommandozeilenoptionen und Konfigurationsdateien eines Programmes zu verwalten. Hierbei müssen lediglich der Name, eine eventuelle Abkürzung und eine Beschreibung der Option für den Hilfetext definiert werden. Weiterhin wird es ermöglicht, einen Datentyp sowie eine Referenz auf eine Variable anzugeben, in welcher der eingegebene Wert gespeichert werden soll. Die gesamte Verarbeitung der Optionen, also Fehlerprüfung und Ausgabe von Fehlermeldungen, speichern der Werte in den angegebenen Variablen sowie die Erzeugung eines Hilfetextes aus den angegebenen Beschreibungen geschehen automatisch.

tokenizer Die `tokenizer`-Library stellt einen String-Tokenizer zur Verfügung, mit dem auf einfache Weise ein String anhand einer Liste von Separatoren in einzelne Token zerlegt werden kann, die dem Programm in Form eines Iterators zur Verfügung gestellt werden.

format Mit Hilfe dieser Library ist es möglich, Strings ähnlich wie mit den C-Funktionen `printf` und `sprintf` zu formatieren, bietet aber deutlich mehr Möglichkeiten für den Umgang mit den Ergebnissen.

lexical_cast Diese Library stellt eine gleichnamige Template-Funktion bereit, die einen übergebenen String in den als Template-Parameter angegebenen Typ umwandelt.

Die Boost Libraries sind in den Paket-Management Systemen der meisten Linux-Distributionen[32] verzeichnet, sollten sich also ohne weiteres mit den üblichen Mitteln installieren lassen. Ansonsten kann unter `http://sourceforge.net/projects/boost` der gesamte Quelltext der Boost Libraries heruntergeladen werden. Für Windows Benutzer steht unter `http://www.boost-consulting.com/products/free` ein Installer zum Download bereit.

6.4. Kompilieren des Projektes

Die im Rahmen dieser Arbeit erstellte Software wurde sowohl unter Windows XP Professional (Visual-C++ 8.0) als auch unter Linux (gcc-4.1.2) getestet. Abgesehen von den in Abschnitt 6.3 aufgeführten Software-Bibliotheken wurden lediglich Funktionen der C++ Standardbibliothek und der STL eingesetzt.

[32] Es existieren Pakete für debian, Ubuntu, SuSE, RedHat, fedora und gentoo und wahrscheinlich auch für weitere Linux und Unix-Varianten.

6.4.1. Linux

Unter Linux genügt nach Installation der benötigten Bibliotheken in der Regel ein Aufruf von `make` im Verzeichnis `src/`, um den Kompiliervorgang zu starten. Ist das von der LTILib installierte Skript `lti-config` nicht im Suchpfad für ausführbare Dateien (dies ist standardmäßig der Fall) muss der vollständige Pfad zu diesem Skript in dem Makefile eingetragen werden (`LTICONFIG=/path/to/lti-config`).

6.4.2. Windows

Unter Windows kann das Projekt mit Visual-C++ 2005 kompiliert werden. Hierzu muss zunächst die Solution-Datei `src/lts.sln` in Visual-C++ geöffnet werden. Bevor das Projekt nun erstellt werden kann, müssen noch die Installationspfade der LTILib und der boost-libraries konfiguriert werden. Diese Einstellungen können in den Property-Sheets ltilibConfig und boostConfig (`ltilibConfig.vsprops` bzw. `boostConfig.vsprops`) vorgenommen werden. Hierzu müssen die Property-Sheet im Property-Manager von Visual-C++ geöffnet werden und unter „User-Macros“ die Macros „boostDir“ bzw. „ltilibDir“ angepasst werden.

7. Fazit

Die Entwicklung eines vollständigen 3D-Laserscanners mit der dazugehörigen Software ist ein sehr großes Projekt. Mit dieser Arbeit wurde der Grundstein für einige darauf aufbauende Arbeiten gelegt, mit denen schließlich ein solches System realisiert werden kann.

In dieser Arbeit wurden die mathematischen und bildverarbeitungstechnischen Grundlagen der Lasertriangulation erläutert und es wurden Ansätze für die Kalibrierung dieses Systems und die zweidimensionale Vermessung einer Objektoberfläche beschrieben und implementiert. Desweiteren wurden Untersuchungen über die mit diesen Ansätzen realisierbaren Genauigkeiten angestellt.

Die hierbei erzielten Ergebnisse sind insgesamt vielversprechend und lassen hoffen, dass es möglich ist mit den gegebenen Mitteln einen 3D-Laserscanner mit einer relativ hohen Messgenauigkeit zu realisieren. Um jedoch in die Genauigkeitsbereiche vorzudringen, in denen sich die kommerziell eingesetzten 3D-Scanner huetzutage bewegen, ist noch einiges an Arbeit zu erledigen.

Diese Arbeit bietet noch einigen Spielraum für weitere Untersuchungen. So wäre es zum Beispiel sehr interessant zu analysieren, mit welchen Mitteln sich die Genauigkeit der Messungen noch weiter verbessern ließe. Wie groß ist zum Beispiel der Einfluss der genauen Ausrichtung von Laser und Kamera? Lassen sich mit anderen Kalibrierungsverfahren die Referenzpunkte noch genauer bestimmen? In welchem Maß lässt sich die Genauigkeit der Kalibrierung durch ein Ausgleichen der Linsenverzeichnungen verbessern?

Auch in Bezug auf das für die Erkennung der Laserlinie entwickelten Verfahrens gibt es sicherlich noch einige Verbesserungsmöglichkeiten. Es lässt sich hier beispielsweise die Frage stellen, wie den aufgrund variierender Oberflächeneingenschaften eines Messobjektes auftretenden Schwankungen der Qualität der Messergebnisse entgegengewirkt werden könnte. Lässt sich dies zum Beispiel durch ein Variieren der Lichtstärke des Lasers erreichen? Was für andere Verfahren ließen sich einsetzen um die Mittelpunkte der Laserlinie noch genauer zu bestimmen?

Bei der Auswertung der Messergebnisse ist deutlich geworden, das sich die Genauigkeit der Messungen nur dann richtig beurteilen lässt, wenn die Abmessungen und geometrischen Eigenschaften des Referenzkörpers weitaus genauer bekannt sind, als die zu erwartende Messgenauigkeit. Messungen mit einem solchen Referenzkörper können sicherlich zur Beantwortung einiger der zuvor gestellten Fragen beitragen.

Das in dieser Arbeit entwickelte Kalibrierungsverfahren ist auf ähnliche Weise in seiner Genauigkeit eingeschränkt, wenn es nicht mehr um Millimeter sondern um zehntel- oder gar hundertstel-Millimeter geht, da in diesen Genauigkeitsbereichen schon eine winzige Abweichung der Kalibriervorrichtung von ihrer Position oder ein kleiner Fehler bei der Ausrichtung des Lasers ausreicht, um die Positionsgenauigkeit der Refererenzpunkte schlechter als die gewünschte Messgenauigkeit werden zu lassen. Hier stellt sich also die Frage, ob es Kalibrierungsverfahren gibt, bei denen diese Probleme nicht oder nur in abgeschwächter Form auftreten.

Aufbauend auf den Ergebnissen dieser Arbeit ist noch einiges zu tun, um letztendlich vollständige 3D-Modelle von beliebigen Objekten erstellen zu können. Hierzu gehören die Weiterentwicklung des in dieser Arbeit realisierten Systems hinsichtlich seiner Genauigkeit wie auch die weiteren Schritte hin zur vollständigen Erfassung eines kompletten 3D-Modells.

Ausgehend von der Erfassung einer einzelnen Laserlinie könnte eine nachfolgende Arbeit das Ziel verfolgen, durch Drehung des Messobjektes eine große Anzahl von Oberflächenlinien zu erfassen und diese zu einem vollständigen 3D-Modell des Objektes zusammenzufügen.

Im folgenden könnten sich weiterführende Arbeiten mit der Nachbearbeitung der so entstehenden 3D-Modelle, also mit der Optimierung der Knotenanzahl der Gitternetze oder der Verknüpfung der Ergebnisse mehrerer Scans desselben Objektes beschäftigen.

A. Quelltexte

Inhalt

A.1. Dateien im Quelltextverzeichnis 45
A.2. Interfaces 46
A.3. Aufruf der Kalibrierung und Objektvermessung 48

A.1. Dateien im Quelltextverzeichnis

Im Folgenden zunächst eine Übersicht über die wichtigsten Dateien im Quelltext-Verzeichnis

src/Makefile Make-Steuerdatei für die Erstellung der ausführbaren Dateien unter Linux.

src/lts.sln Solution-Datei für Microsoft Visual-C++ 2005.

src/ltilibConfig.vsprops Visual-C++ Property-Sheet für die Konfiguration der ltilib

src/ltilibConfig.vsprops Visual-C++ Property-Sheet für die Konfiguration der boost-libraries

src/main.cpp Einstiegspunkt des Programms, enthält die main-Funktion sowie die Verarbeitung der Parameter und Konfigurationsdateien

src/calibration/Calibration.h Header-Datei für die Kalibrierungsfunktion, enthält die Klassendefinition der Klasse `lts::Calibration`

src/calibration/Calibration.cpp Implementierung der Klasse `lts::Calibration`

src/lts/Lts.h Header-Datei für die Vermessungsfunktion, enthält die Klassendefinition der Klasse `lts::Lts`

src/lts/Lts.cpp Implementierung der Klasse `lts::Lts`

A.2. Interfaces

In diesem Abschnitt sind die Programmierschnittstellen der Klassen `lts::Calibration` und `lts::Lts` in reduzierter Form dargestellt. Die vollständigen Schnittstellendefinitionen und eine deutlich ausführlichere Dokumentation findet sich in den entsprechenden Header-Dateien.

```
1  namespace lts {
2      class Calibration {
3          public:
4          // parameters-class, see Listing A.2
5          class parameters { /* ... */ };
6
7          // constructors
8          Calibration();
9          Calibration( const Calibration::parameters &p );
10
11         // getters and setters
12         void setParameters( const lts::Calibration::parameters &p );
13         void setRefPoints( const std::vector<lti::fpoint> &refPoints );
14         lti::dmatrix &getTransformationMatrix();
15
16         // apply-methods
17         void apply( const lti::image &srcImage );
18         void apply( const lti::image &srcImage,
19                 const std::vector<lti::fpoint> &refPoints );
20     };
21 };
```

Listing A.1: public-interface der Klasse `lts::Calibration`

```
1  namespace lts {
2      class Calibration {
3          public:
4          class parameters {
5              public:
6              // default- and copy-constructor
7              parameters();
8              parameters(const parameters &other);
9              parameters& copy(const parameters& other);
10
11             // operator for output to ostreams
12             friend std::ostream& operator <<( std::ostream &os,
13                     const Calibration::parameters &p );
14
15             // parameters (public attributes)
16             // configuration for the hough line-transform
17             int hltBaseValue;
```

```
18              int hltAccuracy;
19              float hltLmdStdDevFactor;
20              float hltLmdHysteresis;
21
22              // distance between two edge-samples
23              float edgeSampleDistance;
24
25              // width of an edge-sample
26              float edgeSampleWidth;
27
28              // minimal difference between minimum and maximum of the
29              // laPlacian within the edge-sample
30              float minLPValueDiff;
31
32              // maximum distance between indices of minimum and maximum of the laPlacian
33              int edgeMinMaxIdxDistance;
34
35              // definition of the region of interest
36              bool roiDefined;
37              lti::polygonPoints roi;
38          };
39          // ...
40      };
41  };
```

Listing A.2: Die Parameterklasse `lts::Calibration::parameters`

```
1  namespace lts {
2      class Lts {
3          public:
4          // parameters-class, see Listing A.4
5          class parameters { /* ... */ };
6
7          // constructors
8          Lts();
9          Lts( const Lts::parameters &p );
10         Lts( const Lts::parameters &p, const lti::dmatrix &trMatrix );
11
12         // accessors
13         void setParameters( const Lts::parameters &params );
14         void setTransformationMatrix( const lti::dmatrix &trMatrix );
15         void getPointList( std::vector<lti::fpoint> &res );
16
17         // apply-methods
18         void apply( lti::image &srcImage );
19         void apply( lti::image &srcImage, const lti::dmatrix &trMatrix );
20     };
```

```
21 };
```

Listing A.3: public-interface der Klasse `lts::Lts`

```
1  namespace lts {
2      class Lts {
3          public:
4          class parameters {
5              public:
6              // default- and copy-constructor
7              parameters();
8              parameters(const parameters &other);
9              parameters& copy(const parameters& other);
10
11             // operator for output to ostreams
12             friend std::ostream& operator <<( std::ostream &os,
13                     const Calibration::parameters &p );
14
15             // parameters (public attributes)
16             // cannyEdges parameters
17             float ceTresholdMin;
18             float ceTresholdMax;
19             float ceKernelSize;
20             float ceVariance;
21
22             // reflection-filter settings
23             bool rfEnabled;
24             float rfSize;
25             float rfMaxLineWidth;
26             float rfTreshold;
27             int rfOuterScore;
28             int rfInnerScore;
29         };
30         // ...
31     };
32 };
```

Listing A.4: Die Parameterklasse `lts::Lts::parameters`

A.3. Aufruf der Kalibrierung und Objektvermessung

Im Folgenden werden nun zwei kurze Code-Beispiele zur Verwendung der Klassen `lts::Calibration` und `lts::Lts` gegeben.

```
1 lts::Calibration::parameters clbParams;
2
```

```
3  // setup calibration parameters...
4
5  // load the source image
6  lti::loadImage loader;
7  lti::image srcImage;
8  bool isLoaded = loader.load( src_fname, srcImage );
9  if (!isLoaded) { /* handle error */ }
10
11 std::vector<lti::fpoint> refPoints;
12 // fill in reference points...
13
14 lts::Calibration clb( clbParams );
15
16 // run the calibration
17 clb.apply(srcImage, refPoints);
18
19 // retrieve the transformation-matrix
20 lti::dmatrix trMatrix = clb.getTransformationMatrix();
```

Listing A.5: Beispiel für einen Aufruf der Kalibrierung

```
1  lts::Lts::parameters ltsParams;
2
3  // setup parameters ...
4
5  // load the source image
6  lti::loadImage loader;
7  lti::image srcImage;
8  bool isLoaded = loader.load( src_fname, srcImage );
9  if (!isLoaded) { /* handle error */ }
10
11 lti::dmatrix trMatrix;
12 // load the transformation-matrix...
13
14 lts::Lts l( ltsParams, trMatrix );
15
16 // run the survey
17 l.apply(srcImage);
18
19 // retrieve the resulting points.
20 std::vector< lti::fpoint > result;
21 l.getPointList(result);
```

Listing A.6: Beispiel für einen Aufruf der Objektvermessung

B. Verwendete Hilfsmittel

B.1. Software-Entwicklung

B.1.1. Entwicklungswerkzeuge

Die Software die im Rahmen dieser Arbeit entstanden ist wurde vollständig in C++ implementiert. Als Entwicklungswerkzeuge kamen dabei Visual Studio Professional 2005 von Microsoft für die Entwicklung unter Windows und gcc[33] für die Entwicklung unter Linux zum Einsatz. Der Hauptteil der Entwicklung geschah unter Linux mit einem einfachen Texteditor (vim). Visual Studio kam nur zum Verifizieren der Windows-Kompatibilität sowie für das Debugging zum Einsatz.

B.1.2. Software-Bibliotheken

Als Software-Bibliothek wurde hauptsächlich die *„LTI-lib Computer Vision Library“*[34] des Lehrstuhls für technische Informatik der RWTH Aachen in der Version 1.9 eingesetzt. Diese Bibliothek stellt die Implementierungen der verwendeten mathemathischen Algorithmen sowie einen Großteil der verwendeten Bildverarbeitungsverfahren zur Verfügung.

Desweiteren kamen zur Verarbeitung von Kommandozeilenparametern und Konfigurationsdateien sowie für die Formatierung von Textausgaben die *„Boost C++ Libraries“* zum Einsatz.

B.2. Ausarbeitung

Die Ausarbeitung wurde vollständig in LaTeX2ε mit der TeX-Live Distribution der TeXUser-Group geschrieben. Die Illustrationen wurden mit OpenOffice.org Draw erstellt. Die Fotos wurden mit einer Digitalkamera aufgenommen und mit Gimp nachbearbeitet. Für die Erstellung der Funktionsplots wurden gnuplot und epstopdf eingesetzt. Die annotierten Abbildungen aus dem Kalibrierungsprozess und der Objektvermessung wurden mit der LTILib erstellt.

---

[33] gcc ist eine frei verfügbare Sammlung von Compilern für verschiedene Programmiersprachen sowie die dazugehörigen Standard-Libraries (siehe `http://www.gnu.org/software/gcc/gcc.html`).

[34] Die Dokumentation der LTILib findet sich unter `http://ltilib.sourceforge.net` (Alvarado u. a., 2007)

C. Inhalt der beiligenden CD

Die CD, die sich am Ende dieser Arbeit befindet, beinhaltet den vollständigen Quelltext der im Rahmen dieser Arbeit erstellten Software sowie alle Quelldateien dieser Ausarbeitung. Die Verzeichnisstruktur ist wie folgt aufgebaut:

doc/ enthält den vollständigen LaTeX-Quelltext dieses Dokuments, inklusive der Quelldateien der erstellten Illustrationen

examples/ enthält Bash-Skripte und Beispieldateien für die Verarbeitung der Testbilder im Verzeichnis **testimg**/

src/ enthält den Quelltext der implementierten Software sowie die für die Erstellung notwendigen Dateien. Ebenfalls in diesem Verzeichnis befinden sich das Makefile und die Projektdateien für Visual-Studio 2005.

testimg/ enthält eine Reihe von Bildern, die zum Testen der implementierten Software verwendet wurden.

util/ enthält einige Skripte, die für die Erstellung der Datenplots mit gnuplot verwendet wurden.

schuhfuss_bachelorarbeit.pdf dieses Dokument in digitaler Form.

Literaturverzeichnis

[Alvarado u. a. 2007] ALVARADO, Pablo ; DOERFLER, Peter ; CANZLER, Ulrich: *LTI-Lib Homepage*. Homepage des LTI-Lib-Projektes der RWTH Aachen. 2007. – URL `http://ltilib.sourceforge.net`

[Ansorge und Oberle 2000] ANSORGE, Rainer ; OBERLE, Hans J.: *Mathematik für Ingenieure*. Bd. 1: *Mathematik für Ingenieure, Band 1, 3. überarbeitete Auflage*. Berlin : Wiley-VCH, 2000. – ISBN 3-527-40309-4

[boost.org 2007] BOOST.ORG: *Boost Libraries and Documentation*. Dokumentationsseite des „Boost C++ Libraries"-Projektes. 2007. – URL `http://www.boost.org/libs/libraries.html`

[Gonzales und Woods 2002] GONZALES, Rafael C. ; WOODS, Richard E.: *Digital Image Processing, 2nd Edition*. New Jersey : Prentice-Hall, Inc., 2002. – ISBN 0-201-18075-8

[Gramlich 2004] GRAMLICH, Günter: *Anwendungen der linearen Algebra mit MATLAB*. Leipzig : Fachbuchverlag Leipzig, 2004. – ISBN 3-446-22655-9

[Hartley und Zisserman 2000] HARTLEY, R. I. ; ZISSERMAN, A.: *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000. – URL `http://www.robots.ox.ac.uk/~vgg/azbook.html`

[Jackèl 2005] JACKÈL, D.: *3D Computer Vision*. Erschienen als Teil des Skriptes zur Vorlesung 'Computer Vision 2' des Studiengangs Informatik an der Universität Rostock. 2005. – URL `http://wwwicg.informatik.uni-rostock.de/Lehre/CV2/script0304/06_3dcv.pdf`

[Jezouin u. a. 1988] JEZOUIN, J. L. ; SAINT-MARC, P. ; MEDIONI, G.: Building an Accurate Range Finder with Off the Shelf Components. In: *CVPR'88 (IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Ann Arbor, MI, June 5–9, 1988)*. Washington, DC. : Computer Society Press, Juni 1988, S. 195–200. – URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?isNumber=5032&prod=CNF&arnumber=196236&arSt=+195&ared=+200&arNumber=196236`

[Lichtblau und Weisstein 2004] LICHTBLAU, Daniel ; WEISSTEIN, Eric W.: *Condition Number*. Erschienen auf der Webseite mathworld.wolfram.com. 2004. – URL `http://mathworld.wolfram.com/ConditionNumber.html`

[Mackens und Voss 1993] MACKENS, Wolfgang ; VOSS, Heinrich: *Mathematik 1 für Studierende der Ingenieurwissenschaften*. Aachen : HECO-Verlag, 1993. – ISBN 3-930121-00-X

[Meisel 2006] MEISEL, Andreas: *Robot Vision*. Erschienen als Skript zur Vorlesung 'Robot Vision' der Informatikstudiengänge an der Hochschule für Angewandte Wissenschaften Hamburg. 2006. – URL https://www.informatik.haw-hamburg.de/cms/?id=432

[Meisel 2007] MEISEL, Andreas: *Anwendungen - 3D Bildverarbeitung*. Erschienen als Skript zur Vorlesung 'Anwendungen - 3D Bildverarbeitung' des Masterstudiengangs Informatik an der Hochschule für Angewandte Wissenschaften Hamburg. 2007. – URL https://www.informatik.haw-hamburg.de/cms/?id=744

[Niedermair und Niedermair 2006] NIEDERMAIR, Elke ; NIEDERMAIR, Michael: *LaTeX: Das Praxisbuch*. Poing : Franzis Verlag, 2006. – ISBN 3-7723-6930-8

[Press u. a. 1992] PRESS, William H. ; TEUKOLSKY, Saul A. ; VETTERLING, William T. ; FLANNERY, Brian P.: *Numerical Recipes in C, 2nd Edition*. Cambridge : Cambridge University Press, 1992. – ISBN 0-521-43108-5

[Rogers und Adams 1976] ROGERS, D. F. ; ADAMS, J. A.: *Mathematical Elements for Computer Graphics*. New York : McGraw-Hill, 1976. – ISBN 0-07-053530-2

[Stroustrup 2000] STROUSTRUP, Bjarne: *The C++ Programming Language, Special Edition*. Boston : Addison-Wesley, 2000. – ISBN 0201700735

[Weisstein 2006] WEISSTEIN, Eric W.: *Singular Value Decomposition*. Erschienen auf der Webseite mathworld.wolfram.com. 2006. – URL http://mathworld.wolfram.com/SingularValueDecomposition.html

[Weisstein 2007] WEISSTEIN, Eric W.: *QR Decomposition*. Erschienen auf der Webseite mathworld.wolfram.com. 2007. – URL http://mathworld.wolfram.com/QRDecomposition.html

[wikipedia.org 2007a] WIKIPEDIA.ORG: *Bilinear Interpolation*. Erschienen im Rahmen des Wikipedia-Projektes. 2007. – URL http://en.wikipedia.org/wiki/Bilinear_Interpolation

[wikipedia.org 2007b] WIKIPEDIA.ORG: *Condition Number*. Erschienen im Rahmen des Wikipedia-Projektes. 2007. – URL http://en.wikipedia.org/wiki/Condition_Number

[wikipedia.org 2007c] WIKIPEDIA.ORG: *Linear Least Squares*. Erschienen im Rahmen des Wikipedia-Projektes. 2007. – URL http://en.wikipedia.org/wiki/Linear_least_squares

[wikipedia.org 2007d] WIKIPEDIA.ORG: *QR Decomposition*. Erschienen im Rahmen des Wikipedia-Projektes. 2007. – URL http://en.wikipedia.org/wiki/QR_decomposition

[wikipedia.org 2007e] WIKIPEDIA.ORG: *Singular Value Decomposition*. Erschienen im Rahmen des Wikipedia-Projektes. 2007. – URL `http://en.wikipedia.org/wiki/SVD`

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 5. Oktober 2007 Martin Schuhfuß