Rheinische Friedrich-Wilhelms-Universität Bonn


Institut für Informatik III

# Autonome Exploration und Modellierung von 3D-Umgebungen

Andreas Nüchter

Angefertigt am Fraunhofer Institut für Autonome Intelligente Systeme


Erstgutachter: PD Dr. Joachim Hertzberg
Zweitgutachter: Prof. Dr. Armin B. Cremers

## Zusammenfassung

Autonome mobile Roboter müssen in der Lage sein, sicher durch ihre Umgebung zu navigieren, um anwendungsspezifische Aufgaben ausführen zu können. Gelingen kann dies nur durch den Einsatz von 3D-Sensoren und 3D-Karten. Daher ist die automatische und schnelle Modellierung der Umgebung eine wichtige Fragestellung in der Robotik. 3D-Laserscanner sind eine junge Technologie, die die Erfassung räumlicher Daten revolutioniert und Robotern das dreidimensionale Abtasten von Objekten möglich macht. Die vorliegende Arbeit untersucht und evaluiert die zur autonomen 3D-Kartenerstellung notwendigen Algorithmen mit Hilfe des AIS 3D-Laserscanners, der sich auf einer geeigneten Roboterplattform befindet. Das entwickelte System ermöglicht dabei das berührungslose Abtasten der *gesamten* Umgebung.

Der erste Teil der Arbeit beschäftigt sich mit der Aufgabe, 3D-Scans in einem globalen Koordinatensystem zu registrieren. Die von der Odometrie des Roboters geschätzte Pose (Position und Orientierung) wird dabei korrigiert. Verschiedene Variationen des iterativen Algorithmus der nächsten Punkte (ICP) kommen zum Einsatz. Im zweiten Teil geht es darum, eine möglichst optimale nächste Scanposition zu bestimmen, von der aus unbekanntes Terrain erforscht werden kann. Ein randomisierter Approximationsalgorithmus plant die neue Position des Scanners. Anschließend ist diese Position durch eine geeignete Motorregelung anzufahren, wobei Kollisionsvermeidung berücksichtigt wird. Schließlich werden die Ergebnisse in geeigneter Weise, unter anderem durch Gittermodelle, visualisiert.



**Schlagwörter:** 3D-Laserscanner, 3D-Modellierung, 3D-Kartenerstellung, Scanmatching, iterativer Algorithmus der nächsten Punkte, autonome Exploration, simultanes Lokalisations- und Kartierungsproblem, Approximation der nächsten optimalen Scanposition, Robotersteuerung, Oberflächenrepräsentation durch Gittermodelle.



## Abstract

Autonomous mobile robots must be able to navigate safely through their environment in order to fulfill user specific tasks. The only way to achieve this is using 3D sensors and 3D maps. Therefore an important question in robotics is automatic and fast modeling of environments. A revolutionary method for gaging environments are 3D laser range finders which enable robots to scan objects in three dimensions. The presented work examines and evaluates the algorithms needed for autonomous 3D map building using the AIS 3D laser range finder mounted on a suitable robot platform. The built system scans the *whole* environment in a non-contact way.

The first part deals with the task to register 3D scans in a common coordinate system. The odometry-based robot pose (position and orientation) is corrected by this process. Different variations of the iterative closest point (ICP) algorithm are used. In the second part the calculation of the next best view for exploration of the un-scanned area is presented. This planning of the sensor placement is done by a randomized approximation algorithm. A suitable controller drives the robot with obstacle avoidance to the calculated position. Finally the results are visualized using different methods, e.g. meshes.



**keywords:** 3D laser range finder, 3D modeling, 3D map building, scan matching, iterative closest point algorithm, autonomous exploration, simultaneous localization and map building problem, next best view approximation, robot control, surface representation through meshes.

# Danksagung

Herzlich danken möchte ich meinem Erstgutachter Herrn PD Dr. Joachim Hertzberg, der mir wichtige Einblicke in die interessanten Forschungsgebiete *künstliche Intelligenz* und *Robotik* gewährt hat. Mein Dank gilt ebenfalls Herrn Dr.-Ing. Hartmut Surmann. Seine große Bereitschaft, meine Fragen zu beantworten und fachliche Diskussionen zu führen, hat diese Arbeit durch wertvolle Hinweise und Ideen vorangetrieben. Von seinem Wissen über den verwendeten Roboter konnte ich sehr profitieren.

Ein großes Dankeschön auch an Jutta Busenbender, die den Text Korrektur gelesen hat.

Dank sagen möchte ich schließlich meinen Eltern. Sie haben mir dieses Studium ermöglicht und mich in jeder Hinsicht unterstützt.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Symbolverzeichnis

| | |
|---|---|
| $\mathbb{N}$ | Menge der natürlichen Zahlen |
| $\mathbb{C}$ | Menge der komplexen Zahlen |
| $\mathbb{R}$ | Menge der reellen Zahlen |
| | |
| $M, D, P$ | Mengen von Messdaten, Teilmenge von $\mathbb{R}^3$ |
| $\mathbf{d}$, $\mathbf{m}$ | Messpunkte, Elemente von $\mathbb{R}^3$ |
| $N_m$ | Anzahl der Messpunkte in $M$, $N_m = \|M\|$ |
| $N_d$ | Anzahl der Messpunkte in $D$, $N_d = \|D\|$ |
| $\mathbf{c}_m, \mathbf{c}_d$ | Schwerpunkte der Mengen $M$ und $D$ |
| $w_{i,j}$ | Gewichte $w_{i,j} \in \mathbb{R}$ |
| | |
| $z$ | komplexe Zahl |
| $z^*$ | konjugiert komplexe Zahl |
| $\dot{q}$ | Quaternion $\dot{q} = q_0 + \mathfrak{i}q_x + \mathfrak{j}q_y + \mathfrak{k}q_z$ mit $q_0, q_x, q_y, q_z \in \mathbb{R}$ |
| $\dot{q}^*$ | konjugiert komplexes Quaternion |
| $\mathbf{q}$ | Spaltenvektor der imaginären Anteile eines Quaternions, $\mathbf{q} = (q_x, q_y, q_z)^T$ |
| $\mathfrak{i}, \mathfrak{j}, \mathfrak{k}$ | Imaginäre Anteile einer komplexen Zahl bzw. eines Quaternions |
| $\mathbf{i}$ | Spaltenvektor imaginärer Einheiten $\mathbf{i} = (\mathfrak{i}, \mathfrak{j}, \mathfrak{k})^T$ |
| $\mathbf{Q}$ | Matrix eines Quaternions |
| | |
| $\mathbf{M}, \mathbf{N}, \mathbf{R}, \ldots$ | Matrizen reeller Zahlen, auch Rotationsmatrizen |
| $\mathbb{1}$ | Einheitsmatrix |
| $\mathbf{v}, \mathbf{n}, \ldots$ | Vektoren reeller Zahlen |
| $\mathbf{M}^T, \mathbf{v}^T$ | transponierte Matrix bzw. transponierter Vektor |
| $\mathbf{M}^{-1}$ | inverse Matrix |
| | |
| $x$, $y$ | Punkte einer Ebene $\in \mathbb{R}^2$ |
| $v$ | Eckpunkte eines Polygons $\in \mathbb{R}^2$ |
| $e$ | Kanten eines Polygons |
| $\mathcal{P}, \mathcal{S}, \mathcal{U}, \ldots$ | Polygone $\subset \mathbb{R}^2$, Mengen von $n$ Eckpunkten $v_1, v_2, \ldots, v_n$, und $n$ Kanten $v_1v_2, v_2v_3, \ldots, v_{n-1}v_n, v_nv_1$ |

| | |
|---|---|
| $\mathsf{c}, \mathsf{u}$ | Klauseln und Literale in booleschen Formeln |
| $\wedge, \vee, \bar{\mathsf{u}}$ | logisches *und*, *oder* und Negation |
| $(x, y, \theta)$ | Pose (Position und Orientierung) des Roboters |
| $x_G, y_G$ | lokale Koordinaten des Controllers nach G. Indiveri |
| $\theta, \phi, \alpha, u, e, c, \ldots$ | Orientierungen und Attribute für den Motorcontroller |
| $V$ | Lyapunovfunktion |
| $\dot{V}, \dot{e}, \dot{\phi}, \dot{\alpha}, \dot{\theta} \ldots$ | Ableitungen nach der Zeit |
| $\mathfrak{P}(X)$ | Potenzmenge von $X$, Menge aller Teilmengen von $X$. |
| P | Die Komplexitätsklasse P, $\mathrm{P} := \bigcup_{k\in\mathbb{N}} \bigcup_{c\in\mathbb{N}} \mathrm{DTIME}(cn^k)$ |
| NP | Die Komplexitätsklasse NP, $\mathrm{NP} := \bigcup_{k\in\mathbb{N}} \bigcup_{c\in\mathbb{N}} \mathrm{NTIME}(cn^k)$ |
| $\varrho(n)$ | Approximationsgüte |
| $\mathcal{O}(f)$ | O-Notation $\mathcal{O}(f) := \{g : \mathbb{N} \to \mathbb{N} \mid \exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n > n_0 :\ g(n) \leq cf(n)\}$ |
| ATAN2 | Der auf allen vier Quadranten definierte Arcus Tangens ist durch $\mathrm{ATAN2}(x,y) = \arctan\frac{y}{x} + \zeta\pi$, $\zeta = \begin{cases} 1 & \forall\ x < 0 \wedge y \geq 0 \\ 0 & \forall\ x \geq 0 \\ -1 & \forall\ x < 0 \wedge y < 0 \end{cases}$ gegeben. |
| $\cdot$ | Skalarprodukt |
| $\times$ | Kreuzprodukt |
| $i, j, k$ | Indizes |
| $\square$ | Ende eines Beweises oder Beweis klar |

# Kapitel 1

# Einleitung

Science Fiction Filme sind voll von ihnen: Laufende, sprechende, handelnde Roboter. Der Traum des Menschen, eine ihm ähnliche Maschine zu entwickeln, die denken, lernen und fühlen kann, spiegelt sich aber nicht nur in den humanoiden Robotern der Science Fiction Literatur wider: Große Fortschritte im Bereich des Forschungsgebiets *künstliche Intelligenz* ermöglichen es Computern zum Beispiel, mathematische Beweise zu führen oder besser Schach zu spielen als Menschen. Dennoch gehören Roboter noch nicht selbstverständlich zum Alltag dazu, und es stellt sich die Frage, warum dies so ist. Eine Ursache besteht in der Sensorik. Sie bereitet den Robotern Probleme. Zwar sind teure Speziallösungen vorhanden, es mangelt allerdings immer noch an preiswerten und robusten Sensoren zur Umgebungswahrnehmung und Selbstlokalisation. Häufig werden auf Robotern optische Sensoren eingesetzt, wie beispielsweise preisgünstige Kameras. Oftmals dient bei diesen Versuchen das Sehen der Menschen als Vorbild.

Neue wissenschaftliche Erkenntnisse aus dem Bereich der Wahrnehmungspsychologie haben dagegen ergeben, dass die optische Wahrnehmung des Menschen nur so gut funktioniert, weil er im Kindesalter die Umwelt erfahren und begriffen hat [57, 84]. Eine entscheidende Rolle beim Begreifen spielt dabei der haptische Sinn (Tasten und Fühlen) [57, 84]. Die Haptik ist die wichtigste Informationsquelle für das Weltverständnis eines Kindes. So können Kleinkinder Objekte visuell wieder erkennen, die sie zuvor ertastet haben. Erst Monate später gelingt es ihnen auch, Dinge wieder zu erkennen, die sie nur gesehen haben [73]. Auch Erwachsene machen intensiven Gebrauch vom Tastsinn, allerdings meistens unbewusst. So regen „Berühren verboten“-Schilder in Museen zum Anfassen an, oder „Frisch gestrichen“-Hinweise zum Nachschauen, ob die Farbe tatsächlich noch frisch ist [84].

In der vorliegenden Arbeit wird ein 3D-Laserscanner als Sensor für einen Roboter eingesetzt. Ein Laserscanner *tastet* die Oberfläche aktiv durch das Aussenden von Lichtstrahlen ab. Dabei wird nicht nur die Form der Umgebung, sondern auch ihre Oberflächeneigenschaften (Reflexion, Absorption) bestimmt.

Vorgestellt wird ein System, mit dem Umgebungen von Robotern erfasst und vermessen werden können. Ein Roboter soll so programmiert werden, dass er seine Umgebung exploriert und eine 3D-Karte von ihr erstellt. Dabei muss der Roboter nicht nur seine ihm unbekannte Umge-

bung erfassen, sondern auch ein konsistentes Geometriemodell von ihr erstellen. Er muss sicher navigieren und fahren können, sowie sich überlegen, wo interessante Positionen sind, die die Erforschung des noch unbekannten Terrains erlauben.

Bei der zu explorierenden Umgebung soll es sich um normale Gebäude handeln. Es werden keine Veränderungen in Form von Landmarken oder Strukturvereinfachungen vorgenommen. Als Roboter dient die Ariadne Roboterplattform. Dieses 80 cm × 60 cm × 80 cm große und 250 kg schwere, fahrerlose Transportsystem wird mit dem AIS 3D-Laserscanner und einem PIII-800 Notebook erweitert. Abbildung 1.1 zeigt das Robotersystem.


**Abbildung 1.1:** Die Ariadne Roboterplattform ausgestattet mit einem AIS 3D-Laserscanner und Notebook

## 1.1 Wissenschaftlicher Beitrag

2D-Laserscanner werden heutzutage auf vielen autonomen Robotern verwendet. Ihr zunehmender Einsatz ist darauf zurückzuführen, dass sie aus dem Bereich der Sicherheitstechnik stammen und immer preiswerter geworden sind. In [52, 74, 75, 76] wird eine Möglichkeit vorgestellt, die günstigen 2D-Laserscanner zu 3D-Sensoren zu erweitern. Die vorliegende Arbeit baut auf diese Forschungsergebnisse auf und beschreibt, wie ein 3D-Laserscanner auf Robotern eingesetzt werden kann.

Die Arbeit legt einen Schwerpunkt darauf, Verfahren zum Zusammenfügen von 3D-Scans zu testen und diese mit einem Planungsalgorithmus zu kombinieren, damit die Umgebung eines

Roboters vollständig autonom und dreidimensional erfasst werden kann. Für Roboter, die in Gebäuden operieren, wurde eine solche Verfahrenskombination noch nicht erforscht (vgl. dazu Kapitel 2).

## 1.2 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in 6 Kapitel:

**Kapitel 1** ist die vorliegende Einleitung.

**Kapitel 2** gibt einen Überblick über den aktuellen Stand der Forschung. Vorgestellt werden Laserscannersysteme zur 3D-Rekonstruktion. Am Ende des Kapitels wird der in dieser Arbeit verwendete AIS 3D-Laserscanner behandelt. Eine wichtige Rolle spielen die auf den Scanner zugeschnittenen Online-Algorithmen, die während des Scans bereits einen großen Teil der Datenverarbeitung erledigen.

**Kapitel 3** beschäftigt sich mit dem Zusammenfügen mehrerer 3D-Scans zu einer konsistenten Szene. Zuerst werden Verfahren behandelt, die zwei 3D-Scans vereinigen. Anschließend wird dies auf mehrere 3D-Scans erweitert. Die vorgestellten Algorithmen korrigieren gleichzeitig die geschätzte Pose (Position und Orientierung) des Roboters.

**Kapitel 4** heißt „Die optimale nächste Scanposition“. Sie muss berechnet werden, damit die Umgebung möglichst effizient erfasst wird. Es stellt sich heraus, dass das Finden der Position selbst dann ein schwieriges Problem darstellt, wenn der Roboter bereits eine Karte seiner Umgebung besitzt. Ein Approximationsalgorithmus wird entwickelt, der eine möglichst optimale nächste Scanposition für das Erstellen einer 3D-Umgebungskarte berechnet.

Der zweite Teil des Kapitels beschreibt, wie der Roboter die berechnete optimale nächste Scanposition anfährt. Weiterhin erläutert er den verwendeten Motorcontroller im Detail.

**Kapitel 5** stellt die Ergebnisse und Erkenntnisse zusammen. Dabei werden alle Teilergebnisse der vorangegangenen Kapitel verbunden. Auch geht dieser Teil auf die Visualisierung der Messdaten ein.

**Kapitel 6** ist die Zusammenfassung der Arbeit. Sie zeigt offene Probleme und gibt einen Ausblick auf künftige Arbeiten.

# Kapitel 2

# 3D-Rekonstruktion mittels Laserscanner — Stand der Technik

Dieses Kapitel beschreibt aktuelle Projekte, die sich mit dem dreidimensionalen Erfassen von Objekten beschäftigen. Der letzte Abschnitt stellt den FhG-AIS 3D-Laserscanner vor, wie er zur Aufnahme der 3D-Umgebung in dieser Diplomarbeit verwendet wird.

## 2.1 Tiefenbilder zur Rekonstruktion von Umgebungen

**RESOLV.** Virtuelle Umgebungen im Computer zu erschaffen, war das Ziel der Forscher im RESOLV (REconstruction using Scanned Laser and Video) Projekt von 1995 – 2000 [82]. Ein Laserscanner (Marke Riegl [26]) wird über eine Drehvorrichtung sowohl horizontal als auch vertikal bewegt, so dass er die 3D-Stuktur der Umgebung erfasst. Der auf diese Weise entstandene 3D-Laserscanner wird auf einer mobilen Plattform montiert. Zusätzlich ist die Höhe des Scanners veränderlich [69]. Auf Abbildung 2.1 sind diese Scannersysteme zu sehen, wobei das rechte Bild einen autonomen Roboter zeigt.

Zur Rekonstruktion der Umgebung werden ein Scanmatching Algorithmus und ein Planungsalgorithmus für den optimalen nächsten Scanpunkt verwendet. Die 3D-Daten werden mit Intensitätswerten und Photos einer Kamera zusammengeführt. Der Modellrekonstruktionsprozess findet teilweise nach der Datenaufnahme statt [68, 69].

**CAMERA.** Ein aktuelles EU Projekt heißt CAMERA (CAd Modeling of Built Environments from Range Analysis) [53]. Die automatische Akquisition von CAD-Modellen für den Einsatz in der Architektur steht hier im Vordergrund. Die Forscher konzentrieren sich auf Scanmatching, Schätzung der Sensorpositionen, Rekonstruktion aus 3D-Daten und Intensitätswerten, Perzeptionsplannung, Segmentierung auf Basis von Kantenextraktion und die Generierung von Gittermodellen [53, 63, 69].

**Abbildung 2.1:** Mobile 3D-Messsysteme des RESOLV Projektes.

**AVENUE.** An der Columbia Universität in New York findet das Projekt AVENUE statt [22]. AVENUE hat das Ziel, den Prozess der Modellierung von städtischen Geländen zu automatisieren. Die Bestandteile des Projektes sind ein 3D-Modellierungssystem, das auf einem Laserscanner und einer Kamera zur Texturgewinnung basiert, eine Planungskomponente und ein mobiler Roboter (vgl. Abbildung 2.2). Mit diesem System lassen sich virtuelle Szenen von Straßenzügen im Computer kreieren, was unter anderem für den Städtebau sehr bedeutsam ist [3, 22, 31].

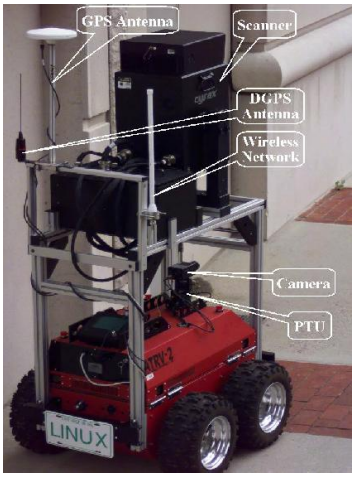


**Abbildung 2.2:** Mobiler Roboter des AVENUE Projektes.

Das Projekt beschäftigt sich ausschließlich mit städtischen Umgebungen; die Roboter operieren also außerhalb von Gebäuden. Der nun folgende Abschnitt handelt von der Erfassung ganzer Landkarten.

**3D-Landkartenerstellung.** Forscher an der Carnegie Mellon University befassen sich mit der automatischen Generierung von 3D-Landkarten [67]. Sehr präzise und hochwertige 3D-Laserscaner werden als Sensoren auf (autonome) Mini-Vans montiert. Zusätzlich überfliegen autonome Hubschrauber das Terrain, die mit 2D-Scanner ausgestattet sind und eine Oberflächenlinie abtasten. Aus den Daten der Hubschrauber und den am Boden aufgenommenen Messwerten entstehen die 3D-Landkarten. Besonders schwierig ist dabei die Verarbeitung der Hubschrauberdaten, weil Flugdaten wie Bewegung und Position sehr ungenau sind. Die Flugdaten werden durch die Verwendung von Mapping Algorithmen korrigiert, die ohne anfängliche Positionsvorgabe arbeiten [36, 42, 43, 67].

## 2.2 3D-Scannen durch Roboterbewegung

Thrun et al. beschreiben ein Robotersystem, das mit zwei 2D-Laserscannern in der Lage ist, die 3D-Umgebung des Roboters zu erfassen [56]. Einer der beiden 2D-Laserscanner tastet die Umgebung vor dem Roboter durch horizontales Scannen ab. Der andere blickt nach oben, scannt also vertikal. Abbildung 2.3 zeigt einen mobilen Roboter der Universität Freiburg, der mit zwei 2D-Laserscannern ausgestattet ist. Die 3D-Daten werden über die Bewegung des Roboters generiert; so erfordert ein 360° Scan eine Drehung des Roboters um 180°. Mit Hilfe dieses Verfahrens lassen sich Gebäudeflure sehr gut abscannen [33, 78].

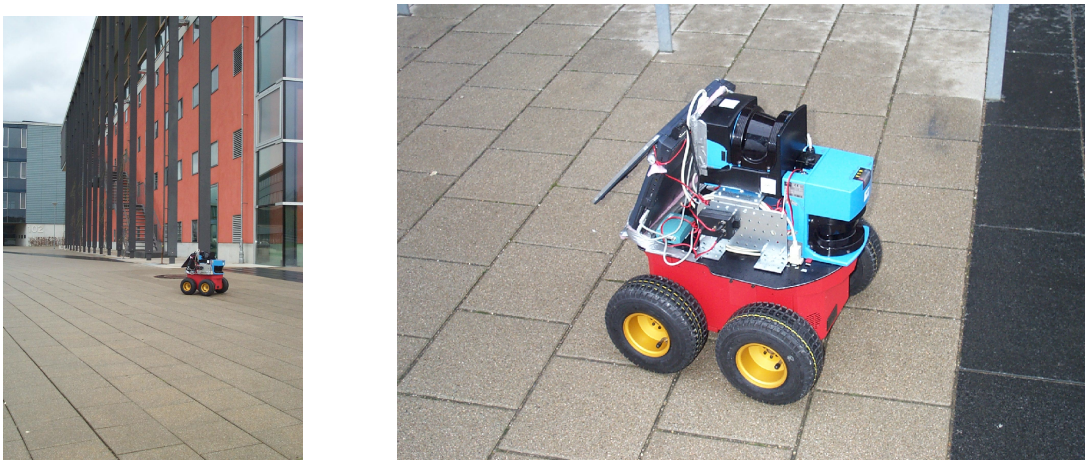

**Abbildung 2.3:** Der mobile Roboter Herbert.

Bei diesem Verfahren ist es extrem wichtig, die Roboterposition genau zu kennen, da sie entscheidend für die Qualität der 3D-Daten ist. Die Lokalisation des Roboters erfolgt mit Hilfe des

horizontal scannenden Lasers. Algorithmen zur Positionsschätzung spielen eine große Rolle in den Arbeiten von Thrun et al. [78].

Die Idee, einen 2D-Laserscanner vertikal zu verwenden, wurde von Banos et al. [30], Zhao et al. [86] und Früh et al. [24, 25] aufgegriffen. Letztere nutzen ein System, das aus zwei SICK-Laserscannern besteht, um Straßenzüge abzutasten. Dabei werden die Laserscanner auf einem Pickup-Truck, der durch San Francisco fährt, montiert [24, 25]. Fassaden von Gebäuden lassen sich damit problemlos erfassen; Seitenansichten sind problematisch, da sie senkrecht zur Fahrtrichtung des Automobils stehen. Diesen Nachteil beheben Zhao et al. indem sie zwei vertikale Scanner 45° zur Fahrtrichtung benutzen [86].

## 2.3 Rekonstruktion von 3D-Modellen

Viele Arbeiten beschäftigen sich mit 3D-Tiefenbildern, um Modelle von Objekten im Computer zu rekonstruieren. Im Digital Michelangelo Projekt geht es beispielsweise darum, große Statuen einzuscannen und diese in möglichst realen Darstellungen auf Bildschirmen anzuzeigen [50]. In der Regel ist das Ziel allerdings, Modelle von kleinen Objekten (Höhe $< 50$cm) zu rekonstruieren.

Um solche Modelle im Computer zu rekonstruieren, bewegt sich ein 3D-Sensor um das Objekt [4, 7, 60, 61] herum, oder ein Laserstreifen wird auf das Objekt projiziert [17], wobei dann Triangulationen Tiefenbilder erzeugen. Auch Systeme mit rotierender Drehscheibe, auf der die zu scannenden Objekte Platz finden, sind im Einsatz. Die Tiefenbilder werden nach dem Scannvorgang in Gittermodelle umgewandelt und zu einem Modell zusammengefügt [17, 19, 81]. Diese Modelle lassen sich in CAD Anwendungen (engl.: *computer aided design*) weiternutzen, im Bereich Virtual Reality oder in einem 3D-Fax einsetzen. 3D-Fax nennt man die Möglichkeit, Modelle zu scannen, diese elektronisch weiterzuleiten und am Zielort mit Hilfe von CAM (engl.: *computer aided manufacturing*) zu rekonstruieren [80]. Den Objekt-Modellierungsprozess vollständig autonom zu gestalten, ist ebenfalls ein Ziel aktueller Forschung [41, 44].

## 2.4 Der AIS 3D-Laserscanner

In der vorliegenden Arbeit wird ein schneller, mit großer Genauigkeit arbeitender, zuverlässiger und preiswerter 3D-Laserscanner, der speziell für autonome mobile Roboter entwickelt wurde, als Sensor zum Erfassen der Roboterumgebung verwendet. Die folgende Darstellung des Sensors ist an [52] angelehnt. Ausführlichere Beschreibungen des 3D-Scanners finden sich in [74, 75, 76].

### 2.4.1 Der Aufbau des 3D-Laserscanners

Ein Standard 2D-Laserscanner mit einem Gewicht von 4.5kg und einer Größe von 35cm $\times$ 24cm $\times$ 24cm erhält einen zusätzlichen Freiheitsgrad dadurch, dass er durch eine Drehvorrichtung und einen Standardservomotor um eine horizontale Achse rotiert wird (vgl. Abbildung 2.4). Die Ansteuerung des Servomotors erfolgt über die parallele Schnittstelle direkt vom Hostcomputer

aus. Die Messdaten werden über ein serielles Interface an den Host übertragen. Durch den mechanischen Aufbau sind die Daten des 2D-Laserscanners gegen den Uhrzeigersinn geordnet und die kontinuierliche Drehbewegung bewirkt ebenfalls eine Ordnung der einzelnen Scan-Ebenen.

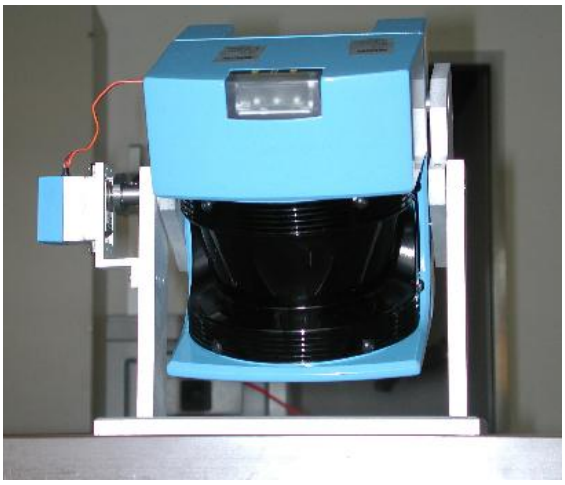

**Abbildung 2.4:** Der 3D-Laserscanner.

Die maximale Scanauflösung für einen kompletten 3D-Scan eines Raumwinkels von 180° (horizontal) × 120° (vertikal) beträgt 194400 Punkte, die in 16.2 Sekunden gescannt werden können. 3D-Scans niedriger Auflösungen, beispielsweise 16200 Punkte, können in 1.35 Sekunden aufgenommen werden. Dabei ist die Genauigkeit abhängig vom verwendeten 2D-Laserscanner (Schmersal: 5cm, SICK: 1cm) sowie von der Genauigkeit des Servomotors und der Servoansteuerung.

### 2.4.2 Software Module des AIS 3D-Laserscanners

#### RealTime-Linux und Online-Algorithmen zur Linien- und Flächenerkennung

Der Servo wird direkt vom Echtzeitbetriebssystem RealTime-Linux angesteuert. Ein Servomotor erwartet alle 20ms einen TTL-Impuls. Dabei bestimmt die Länge des Impulses die Position des Servos (1ms = Links, 1.5ms = Mitte, 2ms = Rechts). Dies ist eine sehr zeitkritische Aufgabe, da bereits eine Abweichung von 10$\mu$s zu einem Fehler von 1° führt. RealTime Linux hat eine durchschnittliche Latenz von 5$\mu$s (PII-333). Somit kann die Rotation bis auf ein halbes Grad genau realisiert werden.

Während des Scans werden verschiedene Online-Algorithmen zur Linien- und Flächenerkennung eingesetzt. Der erste Schritt der Messdatenverarbeitung ist die Linienerkennung (vgl. Abbildungen 2.5, 2.6 und 3.7), die auf jedem 2D-Schnitt einzeln ausgeführt wird. Dabei kommen wahlweise zwei Algorithmen zum Einsatz: Ein einfacher Matching-Algorithmus oder die Hough-Transformation. Der Matching-Algorithmus betrachtet die Punkte in der durch den Scanprozess bestimmten Reihenfolge. Die Messpunkte $a_0, a_1, \ldots, a_n$ sind gegen den Uhrzeigersinn geordnet.

Wenn die Punkte $a_i, \ldots, a_j$ bereits auf einer Linie liegen, muss folgende Bedingung erfüllt sein, damit auch der Punkt $a_{j+1}$ auf der Linie liegt:

$$\frac{||a_i, a_{j+1}||}{\sum_{t=i}^{j} ||a_t, a_{t+1}||} < \varepsilon(j).$$

Dieser Test kann sehr schnell durchgeführt werden, die Qualität der Linien ist jedoch wegen des Rauschens der Messdaten nicht optimal. Um die negativen Effekte der Messabweichungen zu verringern, wird der Linienerkenner nicht direkt auf die Messdaten, sondern auf so genannte REDUZIERTE PUNKTE angewendet. Dicht beieinander liegende Messpunkte werden gemittelt und zu einem Punkt zusammengefasst. Dies dünnt die Werte aus und die Anzahl reduziert sich (vgl. Abbildung 2.5). Die Hough-Transformation ($\mathcal{O}(cn)$, $n$ = Anzahl der Scan-Punkte, $c$ abhängig von der maximalen Entfernung) liefert qualitativ bessere Linien, ist jedoch nur begrenzt als Online-Algorithmus einsetzbar, da die Konstante $c$ sehr groß wird. Eine ausführliche Darstellung der Hough-Transformation ist in [75] zu finden.

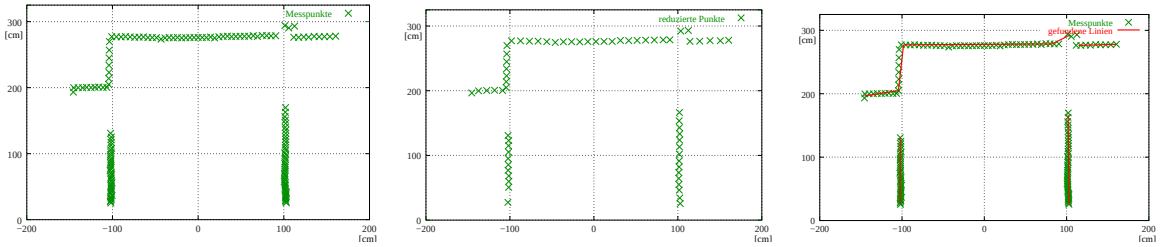


**Abbildung 2.5:** Messpunkte, REDUZIERTE PUNKTE und erkannte Linien in einem Scanschnitt.

Anschließend findet die 3. Dimension Berücksichtigung. Aus den detektierten Linien werden sukzessive Flächen gebaut. Dabei wird eine Linie, bzw. die oberste Linie einer bereits vorhandenen Fläche, mit einer neu erkannten Linie vereinigt, falls folgende Kriterien erfüllt sind:

- Die Endpunkte der neuen Linien müssen in einer Epsilon-Umgebung der Endpunkte von den bereits vorhandenen Linien liegen.
- Der Winkel zwischen den beiden Linien sollte einen gewissen Schwellenwert nicht überschreiten.
- Die neu erkannte Linie muss in der Ebene der bereits vorhandenen Fläche liegen.

Dieses sukzessive Vergrößern angewendet auf jede Scan-Ebene der Flächen ist als Online-Algorithmus implementiert, wird also während des Scanprozesses ausgeführt.

### Polygon-Generierung

Die im letzten Abschnitt vorgestellte Flächendetektion liefert wegen des Rauschens innerhalb der Messdaten viele Flächen, die eventuell überlappen. Die Aufgabe der Polygon-Generierung ist

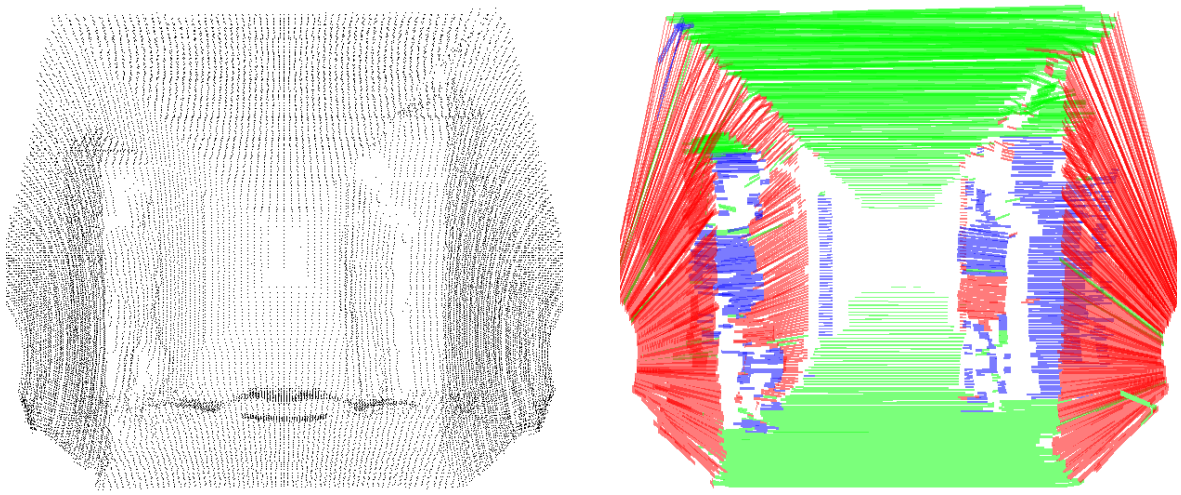

**Abbildung 2.6:** Messpunkte und detektierte Linien.

es, die aus Linien aufgebauten Flächen in Polygone umzuwandeln und überlappende Polygone zu vereinigen. Die Polygonerzeugung erfolgt in 2 Schritten: Zuerst detektiert ein Linienerkenner auf den Endpunkten aller Linien einer Fläche 3D-Kanten. Dadurch wird die aus Linien bestehende Fläche in ein Polygon umgewandelt. Anschließend bildet Vattis Polygon-Clipping-Algorithmus [83] die Vereinigung von Polygonen, die auf einer Ebene liegen (vgl. Abbildung 2.7).

### 3D-Objektsegmentierung

Algorithmen zur Objektsegmentierung dienen zur Erstellung von 3D-Karten der gescannten Szene. Ein sequenzieller Algorithmus vereinigt Punktwolken, Linien und Flächen/Polygone zu Objekten, die durch eine Bounding-Box dargestellt werden. Dieser Vorgang erfolgt in zwei Schritten:

- In einem Vorverarbeitungsschritt werden alle Flächen, deren Größe einen Schwellenwert übersteigt, direkt als Objekte markiert und eine Bounding-Box erstellt.
- Dann findet eine Iteration über die zuvor gefundenen Elemente statt. Es wird geprüft, ob für jedes Objekt ein weiteres „nahe genug“ liegt. Bei der Anwendung für autonome mobile Roboter bemisst sich „nahe genug“ entsprechend der Robotermaße: Bounding-Boxen von Objekten, die so nahe beieinander stehen, dass der Roboter nicht zwischen ihnen hindurch fahren kann, werden vereinigt.

Dieser Algorithmus teilt die Szene so auf, dass Akkumulationen von Punkten, Linien und/oder Flächen zu Objekten zusammengefasst und von einer Bounding-Box umschlossen werden. Die

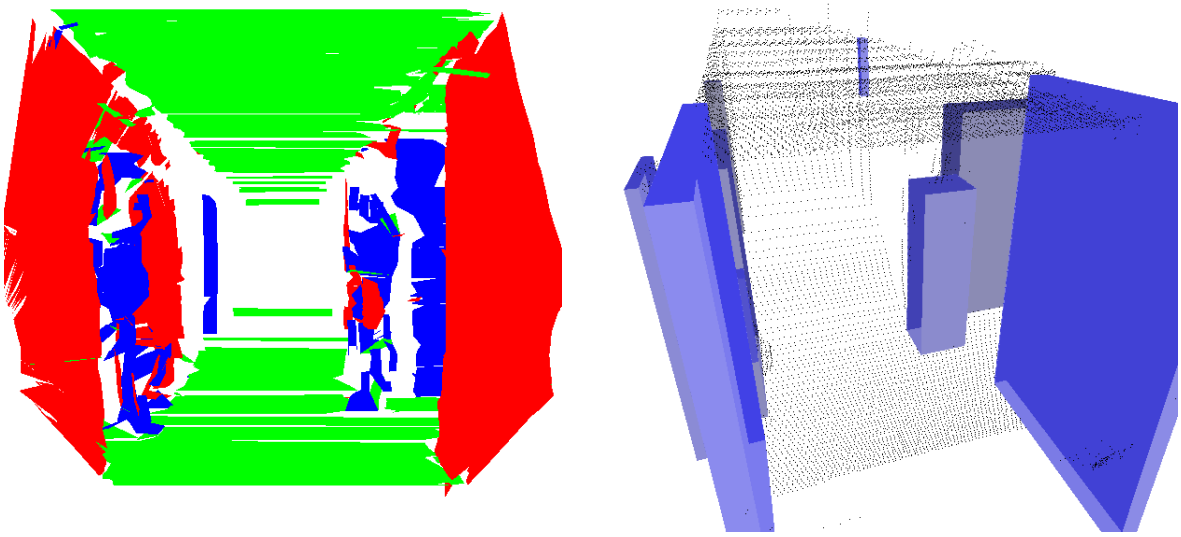

**Abbildung 2.7:** Erkannte Polygone und Objekte.

Bounding-Box stellt nicht nur einen wichtigen Schritt zur Objekterkennung dar, sondern ermöglicht auch das sichere Umfahren dreidimensionaler Hindernisse.

### Ergebnisse der Algorithmen und weitere Eigenschaften des 3D-Laserscanners

Zur Visualisierung der 3D-Daten wird die Grafikbibliothek OpenGL verwendet. Die Abbildungen 2.6 und 2.7 zeigen den Flur C2-Erdgeschoß des FhG-AIS Gebäudes, in dem zwei Personen stehen. Abbildung 2.6 (links) gibt die 46080 Messpunkte wieder, die in 7.6 Sekunden aufgenommen wurden, und als Ergebnis der Linienerkennung (rechts) die 1900 Linien. Diese Linien resultieren in 89 Polygonen (vgl. Abbildung 2.7 (links)), die wiederum aus 422 Dreiecken aufgebaut sind. Der 3D-Objektsegmentierungsalgorithmus detektiert 12 Objekte, dargestellt in Abbildung 2.7 (rechts).

Der für den 3D-Scanner verwendete 2D-Laserscanner liefert in einem speziellen Betriebsmodus Intensitätswerte. Um die Entfernung eines Punktes zu bestimmen, wird ein Laserimpuls ausgesendet und die Zeit gemessen, die vergeht, bis das reflektierte Licht wieder am Laserscanner eintrifft. Gleichzeitig kann aber auch die Intensität des zurückkehrenden Lichts gemessen werden. Der gemessene Wert hängt sowohl von der Entfernung als auch vom Material der gescannten Oberfläche ab, da dort ein Teil des Lichtimpulses absorbiert wird [54]. Der linke Teil der Abbildung 2.8 zeigt die Intensitätswerte; dabei sind nur die Intensitätswerte aufgetragen, die Drehung des Scanners wurde vernachlässigt und das Bild ist verzerrt. Auf dem rechten Teil der Abbildung 2.8 sind vergrößerte und mit den Intensitätswerten eingefärbte 3D-Punkte zu sehen.

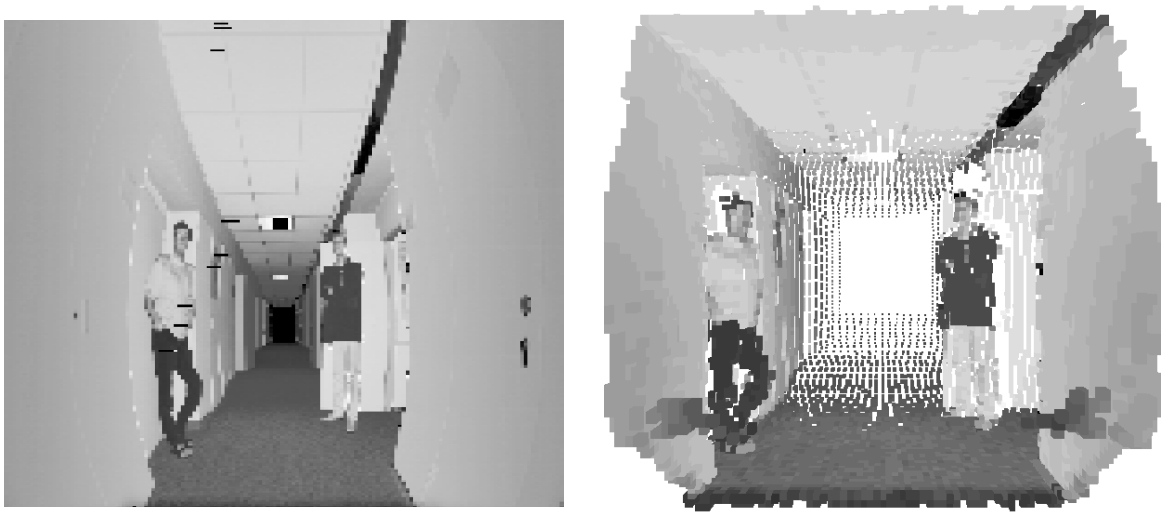

**Abbildung 2.8:** Gemessene Intensitätswerte.

# Kapitel 3

# Scanmatching

Dieses Kapitel beschreibt das Zusammenfügen mehrerer 3D-Scans zu einer konsistenten Szene. Die vorgestellten Algorithmen korrigieren gleichzeitig die geschätzte Pose (Position und Orientierung) des Roboters.

## 3.1 Problemdefinition

Das vollständige Erfassen von komplexen Objekten und Szenen erfordert das Scannen von mehreren Roboterpositionen aus. Nach dem Scanvorgang werden die aufgenommenen 3D-Scans so aneinandergefügt, dass sie die Objekte und die Szene richtig repräsentieren. Das Aneinanderfügen von Scans heißt Registrieren.

Ist die Position des Scanners und damit jene des Roboters genau bekannt, können die 3D-Scans auf der Grundlage dieser Position registriert werden. Leider ist die Selbstlokalisation des Roboters mit einem Fehler behaftet. Daher stellt sich die Frage: *Wie ist die Lage der 3D-Scans zueinander?* Das Zusammenfügen der 3D-Scans darf deshalb nicht nur auf der Roboterposition basieren, sondern muss auch auf der Grundlage der 3D-Scans selbst geschehen. Letzterer Vorgang heißt Scanmatching. Für das Matching von 3D-Scans, die sich überlappen, wurden in den letzten Jahren verschiedene Verfahren entwickelt und in der Literatur vorgestellt [10, 65, 71]. Sie können folgendermaßen unterteilt werden [71]:

**Matching als Optimierungsproblem.** Das Registrieren als Optimierungsproblem bedeutet, eine Kostenfunktion für die Qualität eines Matchings einzuführen. Die Registrierung der 3D-Scans erfolgt über eine Suche nach derjenigen Transformation, die die Kostenfunktion minimiert. Unterschiedliche Kostenfunktionen und Transformationssuchstrategien wurden bereits erforscht [10, 51, 62, 71, 85].

**Merkmalbasiertes Matching.** Dieses Verfahren basiert auf der Suche nach verschiedenen Merkmalen in zwei zu registrierenden 3D-Scans. Aus der Korrespondenz zwischen gleichen Merkmalen kann anschließend die Lage der Scans bestimmt werden. Diese Technik benötigt

kein Vorwissen über die Transformation [65], ist aber wegen der Merkmalextraktion rechenaufwendig [71].

Neben diesen beiden Ansätzen existieren weiterhin noch so genannte hybride Verfahren, die Kombinationen der beiden Methoden sind [65]. Abschnitt 3.3 geht auf den ersten Ansatz ein. Die Ideen des merkmalbasierten Matchings werden am Beispiel des kantenbasierten Matchings in Abschnitt 3.4 hinzugezogen. Beide Verfahren berechnen Drehungen und Verschiebungen der Datenpunkte, so dass die Datensätze möglichst perfekt zueinander passen und konsistent sind. Das folgende Kapitel beschreibt die mathematische Darstellung solcher Drehungen (Rotationen) und Verschiebungen (Translationen).

## 3.2 Darstellung von Rotationen und Translationen

Eine Rotation im Raum ist eine Abbildung, die Punkten $\mathbf{p} \in \mathbb{R}^3$ neue Koordinaten $\mathbf{p'} \in \mathbb{R}^3$ zuweist. Dabei werden alle Punkte um ein festes Drehzentrum und um einen konstanten Winkel gedreht. Es bleiben Längen erhalten: Zwei Punkte haben vor und nach der Rotation den gleichen Abstand. Auch die Winkel zwischen jeweils drei Punkten verändern sich nicht. Des Weiteren handelt es sich bei der Rotation um eine lineare Abbildung ($\mathbb{R}^3 \rightarrow \mathbb{R}^3$). Sie kann durch Euler-Winkel, Gibb-Vektoren, Caley-Klein-Parameter, Pauli-Spin Matrizen, Achsen und Winkel, Hamiltons Quaternionen und orthonormale Matrizen dargestellt werden. In der Robotik werden hauptsächlich Euler Winkel, Quaternionen und Rotationsmatrizen eingesetzt [40]. Jede Rotationsmatrix ist eine $3 \times 3$ Orthonormalmatrix und es gilt: $\mathbf{R}\mathbf{R}^T = \mathbb{1}$ und $\det(\mathbf{R}) = 1$. Diese Matrix multipliziert mit den Punktvektoren ergibt die gedrehten Punkte ($\mathbf{p'} = \mathbf{R}\mathbf{p}$).

Die Translation im Raum weist ebenfalls Punkten $\mathbf{p} \in \mathbb{R}^3$ neue Koordinaten $\mathbf{p'} \in \mathbb{R}^3$ zu. Es werden alle Punkte in eine vorgegebene Richtung um einen konstanten Betrag verschoben, Längen und Winkel bleiben dabei erhalten. Die Translation ist durch einen Vektor $\mathbf{t} \in \mathbb{R}^3$ darstellbar, der zu den Punktvektoren addiert wird ($\mathbf{p'} = \mathbf{p} + \mathbf{t}$).

### 3.2.1 Rotation und Euler Winkel

Eine $3 \times 3$ Rotationsmatrix $\mathbf{R}$ kann durch drei Euler Winkel $\theta_x, \theta_y, \theta_z$, die einer Drehung um die $x, y$ und $z$-Achse entsprechen, ausgedrückt werden [20]. Die orthonormale Matrix $\mathbf{R}$ wird durch

$$\mathbf{R} = \mathbf{R}_x \mathbf{R}_y \mathbf{R}_z \tag{3.1}$$

berechnet, wobei

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{pmatrix}$$

ist und $\theta_x$ den Winkel der Drehung um die $x$-Achse bezeichnet. Die Matrizen $\mathbf{R}_y, \mathbf{R}_z$ sind analog definiert:

$$\mathbf{R}_y = \begin{pmatrix} \cos\theta_y & 0 & -\sin\theta_y \\ 0 & 1 & 0 \\ \sin\theta_y & 0 & \cos\theta_y \end{pmatrix}, \quad \mathbf{R}_z = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Damit ergibt sich die Rotationsmatrix als

$$\mathbf{R} = \begin{pmatrix} \cos\theta_y \cos\theta_z & -\cos\theta_y \sin\theta_z & -\sin\theta_y \\ -\sin\theta_x \sin\theta_y \cos\theta_z + \cos\theta_x \sin\theta_z & \sin\theta_x \sin\theta_y \sin\theta_z + \cos\theta_x \cos\theta_z & -\sin\theta_x \cos\theta_y \\ \cos\theta_x \sin\theta_y \cos\theta_z + \sin\theta_x \sin\theta_z & -\cos\theta_x \sin\theta_y \sin\theta_z + \sin\theta_x \cos\theta_z & \cos\theta_x \cos\theta_y \end{pmatrix}.$$

Bei der Darstellung einer Rotationsmatrix mittels Euler-Winkel ist zu beachten, dass die Reihenfolge der Multiplikationen in (3.1) eine entscheidende Rolle spielt: Das Ergebnis einer Drehung hängt im Allgemeinen davon ab, um welchen Euler Winkel zuerst rotiert wird.

### 3.2.2 Das Einheitsquaternion

Neben der Repräsentation einer Rotation mit Hilfe der Euler Winkel wird das so genannte Einheitsquaternion (engl.: *unit quaternion*) eingesetzt [20]. Das Quaternion beschreibt eine Rotation um einen Vektor, der durch den Ursprung des Koordinatensystems geht. In Abbildung 3.1 sind ein Einheitsvektor $\mathbf{n}$ und ein Winkel $\theta$ dargestellt, die die Rotation der beiden Koordinatensysteme beschreiben. Das blaue Koordinatensystem ist das Ergebnis der Drehung des schwarzen Systems um einen Winkel $\theta$.

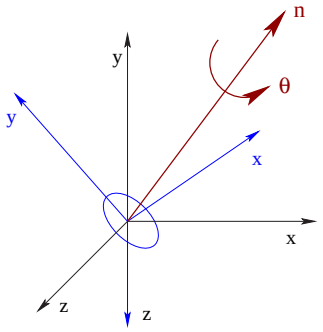


**Abbildung 3.1:** Quaternion zur Beschreibung einer Rotation

Das Quaternion geht auf Hamilton zurück und kann mathematisch als eine komplexe Zahl mit drei verschiedenen imaginären Anteilen behandelt werden (siehe Anhang A.1) [1, 34]:

$$\dot{q} = q_0 + \mathfrak{i}q_x + \mathfrak{j}q_y + \mathfrak{k}q_z \quad \text{mit } q_0, q_x, q_y, q_z \in \mathbb{R}.$$

Bei gegebenem Einheitsvektor $\mathbf{n} = (n_x, n_y, n_z)^T$ und Rotationswinkel $\theta_n$ lässt sich das Einheits-

quaternion berechnen durch [10, 20]:

$$q_0 = \cos\frac{\theta_n}{2} \tag{3.2}$$

$$q_x = n_x \sin\frac{\theta_n}{2} \tag{3.3}$$

$$q_y = n_y \sin\frac{\theta_n}{2} \tag{3.4}$$

$$q_z = n_z \sin\frac{\theta_n}{2}. \tag{3.5}$$

Die Rotationsmatrix berechnet sich aus einem Einheitsquaternion $\dot{q}$ wie folgt:

$$\mathbf{R} = \begin{pmatrix} (q_0^2 + q_x^2 - q_y^2 - q_z^2) & 2(q_x q_y + q_z q_0) & 2(q_x q_z + q_y q_0) \\ 2(q_x q_y + q_z q_0) & (q_0^2 - q_x^2 + q_y^2 - q_z^2) & 2(q_y q_z - q_x q_0) \\ 2(q_z q_x - q_y q_0) & 2(q_z q_y + q_x q_0) & (q_0^2 - q_x^2 - q_y^2 + q_z^2) \end{pmatrix}. \tag{3.6}$$

Dass eine Rotation in dieser Form dargestellt werden kann, geht aus den Beweisen in Anhang A.1.1 und A.1.2 hervor. Die Verwendung des Einheitsquaternions zur Darstellung von Rotationen garantiert, dass die entsprechende Rotationsmatrix orthonormal ist.

## 3.3 Matching als Optimierungsproblem

### 3.3.1 Der iterative Algorithmus der nächsten Punkte

Gegeben sei eine Menge von 3D-Punkten $M = \{\mathbf{m}_i \mid \mathbf{m}_i \in \mathbb{R}^3,\ i = 1, \ldots, N_m\}$. Für einen 3D-Scan mit der Datenmenge $D = \{\mathbf{d}_i \mid \mathbf{d}_i \in \mathbb{R}^3,\ i = 1, \ldots, N_d\}$ sind eine Rotation $\mathbf{R}$ sowie eine Translation $\mathbf{t}$ gesucht, die beide Mengen korrekt ineinander abbilden. Dabei muss die Fehlerfunktion

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j} \left|\left|\mathbf{m}_i - (\mathbf{R}\mathbf{d}_j + \mathbf{t})\right|\right|^2 \tag{3.7}$$

minimiert werden. $w_{i,j}$ nimmt hierbei den Wert 1 an, falls die Messpunkte $\mathbf{m}_i \in M$, $\mathbf{d}_j \in D$ den gleichen Punkt darstellen. Die Minimierung von (3.7) muss mit der Maximierung von

$$\sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j} \tag{3.8}$$

einhergehen. Trifft dies nicht zu, ist die triviale Lösung $w_{i,j} = 0 \quad \forall i, j$ zulässig, was der Idee des Scanmatchings widerspricht.

Der iterative Algorithmus der nächsten Punkte (engl.: *iterative closest points (ICP)*) wurde etwa zeitgleich von Besl/McKay [10], Zhang [85] und Cheng/Medioni [13] 1991 entwickelt. Der Algorithmus – im Folgenden mit ICP bezeichnet – stellt einen allgemeinen Rahmen für die Registrierung von 3D-Tiefenbildern dar. Die fundamentalen Schritte des Algorithmus sind:

1. Für jeden Punkt $\mathbf{d}_i \in D$ berechne bzw. suche den am nächsten gelegenen Punkt in $M$. Es werden die für (3.7) benötigten $w_{i,j}$ bestimmt.
2. Berechne aus der in Schritt 1 bestimmten Korrespondenz die Transformation $\mathbf{R}$ und $\mathbf{t}$, die die Fehlerfunktion $E(\mathbf{R}, \mathbf{t})$ (3.7) minimiert.
3. Wende die in Schritt 2 gefundene Transformation auf die Menge $D$ an.
4. Berechne die Differenz des durchschnittlichen quadratischen Fehlers. Falls diese Differenz kleiner als ein Schwellenwert $\varepsilon$ ist, terminiere. Ansonsten gehe zu Schritt 1.

Es kann nachgewiesen werden, dass der obige Algorithmus konvergiert und ein lokales Minimum findet [10].

**Satz 1 (Konvergenz-Theorem).** Der iterative Algorithmus der nächsten Punkte konvergiert monoton zu einem lokalen Minimum, wenn die Fehlerfunktion $E(\mathbf{R}, \mathbf{t})$ (3.7) gegeben ist [10].

**Beweis:** Der Beweis wird mit vollständiger Induktion geführt. Für alle Iterationen $k$ des Algorithmus, insbesondere für den Induktionsanfang $k = 1$, gelten folgende Aussagen: Gegeben seien zwei Mengen $M$ und $D_k$. Der erste Schritt des obigen Algorithmus liefert die Paare $w_{k,i,j}$, die den Fehler $F_k$ aufweisen:

$$F_k = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{k,i,j} \left|\left|\mathbf{m}_i - \mathbf{d}_{k,j}\right|\right|^2 . \tag{3.9}$$

Schritt 2 des Algorithmus sucht die Transformation $(\mathbf{R}_k, \mathbf{t}_k)$, die, angewendet auf die Datenpunkte $\mathbf{d}_{k,j}$, $F_k$ minimiert. Dadurch wird (3.9) zu

$$E_k = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{k,i,j} \left|\left|\mathbf{m}_i - \mathbf{R}_k \mathbf{d}_{k,j} + \mathbf{t}_k\right|\right|^2 .$$

Offensichtlich gilt immer $F_k \geq E_k$. Angenommen dies wäre nicht der Fall und es gälte $F_k < E_k$, dann würde die Identitätstransformation ($\mathbf{R} = \mathbb{1}$ und $\mathbf{t} = \mathbf{0}$) in einem kleineren Fehler $E_k$ resultieren. Als nächstes wendet man die gefundene Transformation auf $D_k$ an, wodurch die Menge $D_{k+1}$ entsteht. Nun werden die Paare $w_{k+1,i,j}$ berechnet und es ergibt sich folgender neuer Fehler:

$$F_{k+1} = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{k+1,i,j} \left|\left|\mathbf{m}_i - \mathbf{d}_{k+1,j}\right|\right|^2 .$$

Es gilt $E_k \geq F_{k+1}$, weil entweder neue Paare mit kleineren Abständen $\left|\left|\mathbf{m}_i - \mathbf{d}_{k+1,j}\right|\right|^2$ entstehen oder die vorige Paarung wiederholt wird.

Also ist:

$$F_k \geq E_k \geq F_{k+1} \geq 0 \quad \forall k \geq 1. \tag{3.10}$$

Aus dem Satz der Konvergenz einer beschränkten monotonen Folge ergibt sich die Konvergenz zu einem Minimum [23]. □

**Performanz des iterativen Algorithmus der nächsten Punkte.** Der erste Schritt des ICP-Algorithmus hat eine Komplexität von $\mathcal{O}(N_m N_d)$ bei einer *brute force* Suche der nächsten Punkte. Wie Kapitel 3.3.3 zeigen wird, liegt die Berechnung der Transformation aus den Punktpaaren in $\mathcal{O}(N_d)$. Der dritte Schritt, die Anwendung der gefundenen Transformation, benötigt nochmals $\mathcal{O}(N_d)$ Rechenzeit.

Die Gesamtkomplexität ergibt sich als das Maximum der genannten Komplexitäten und ist also $\mathcal{O}(N_m N_d)$.

### 3.3.2 Bestimmung der Punktpaare

Der erste Schritt des ICP-Algorithmus verfolgt das Ziel, Punktpaare zu ermitteln. Für jeden Punkt der Menge $D$ muss der nächste Punkt in der Menge $M$ bestimmt werden. Abbildung 3.2 zeigt zwei Scans des Flurs im FhG-AIS Gebäude C2, wobei der zweite Scan 2 Meter „hinter“ dem ersten liegt. Streng genommen liegt der zweite Scan im ersten. Die Scanpunkte des ersten Scans sind schwarz, die des zweiten blau dargestellt. Zu jedem Punkt des zweiten Scans wurde der nächstgelegene im ersten Scan bestimmt und die Verbindung der beiden Punkte durch eine rote Linie gekennzeichnet. Die größere Punktdichte im Überlappungsbereich lässt sich dadurch erklären, dass der zweite Scan im ersten liegt. Auch müssen sich aus diesem Grund mehrere Punkte des zweiten Scans einen Punkt des ersten Scans als nächsten Punkt teilen, wie die vergrößerten Ausschnitte in der Abbildung zeigen.

Die Performanzanalyse des ICP-Algorithmus hat gezeigt, dass seine Geschwindigkeit von der Suche nach den Punktpaaren dominiert wird. Dies ist der zeitaufwendigste Schritt des ICP-Algorithmus. Daher setzen *alle* Optimierungsversuche dort an. Zwei sich nicht ausschließende Strategien können hierbei verfolgt werden:

**Reduktion der Punkte.** Es werden nur Teilmengen der ursprünglichen Punktmengen betrachtet. Sowohl das im Folgenden vorgestellte Verfahren (Scanmatching mit REDUZIERTEN PUNKTEN) als auch das kantenbasierte Matching (Kapitel 3.4) wenden diese Strategie an.

**Beschleunigung des Datenzugriffs.** Verschiedene Datenstrukturen, wie beispielsweise mehrdimensionale Binärbäume oder Buckets, beschleunigen den Zugriff auf die gesuchten nächsten Punkte.

#### Scanmatching mit REDUZIERTEN PUNKTEN

In Kapitel 2.4.2 wurden die REDUZIERTEN PUNKTE für den Matching-Algorithmus zur Linienerkennung vorgestellt. In jeder 2D-Scan-Ebene resultieren hohe Messpunktdichten durch Mittelwertbildung in einem einzigen Wert, mit dem weitergearbeitet werden kann. Die so entstandenen

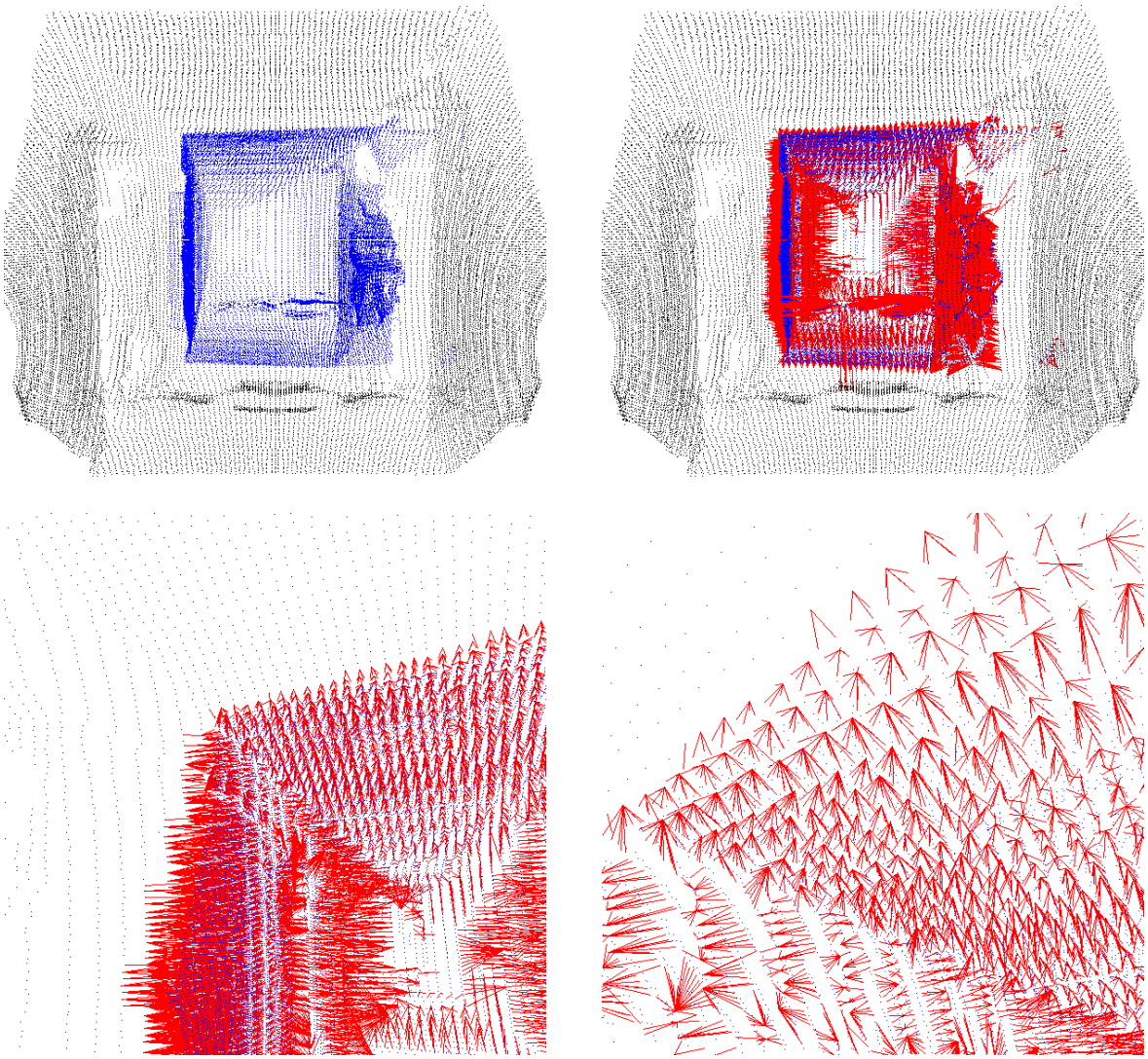

**Abbildung 3.2:** Die nächsten Punkte zweier Scans. Links oben: Zwei Scans, zweiter Scan 2 Meter in $z$-Richtung aufgenommen. Rechts oben: Die nächsten Punkte sind durch eine rote Linie verbunden. Unten: Vergrößerte Ausschnitte.

REDUZIERTEN PUNKTE werden zusätzlich vertikal ausgedünnt, indem nur die gemittelten Punkte aller $t$ Schnitt-Ebenen (z.B. $t = 3$) weiterverwendet werden. Abbildung 3.3 zeigt einen solchen ausgedünnten Scan, bei dem die REDUZIERTEN PUNKTE vergrößert dargestellt sind.

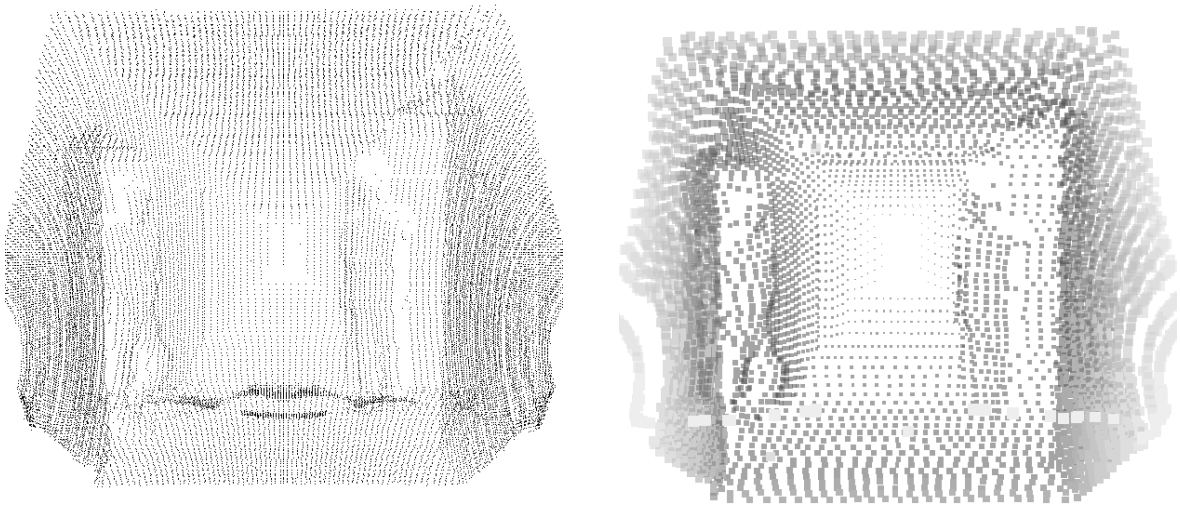

**Abbildung 3.3:** Messpunkte vs. REDUZIERTE PUNKTE.

Dieses Verfahren verringert die Punkte für den ICP-Algorithmus etwa im Verhältnis 1:10. Die in Abbildung 3.3 dargestellten 46080 Messpunkte werden auf 5047 REDUZIERTEN PUNKTEN abgebildet. Die Verwendung von REDUZIERTEN PUNKTEN für den ICP-Algorithmus liefert einen beachtlichen Geschwindigkeitsgewinn.

### Mehrdimensionale Binärbäume

Binärbäume sind ein wichtiges Konzept in der Informatik [14]. Die Speicherung von Daten erfolgt in den Blättern von binären Bäumen, wobei die Schlüssel so gewählt werden, dass jeweils eine Halbierung des Datenraums auftritt. Dies erlaubt einen Zugriff auf die Daten in $\log(n)$-Zeit [14]. Mehrdimensionale Binärbäume ($k$D-Bäume, hier $k = 3$) verallgemeinern dieses Konzept der Halbierung des Datenraumes auf höhere Dimensionen. Die Idee dabei ist, Datenpunkte in einem Binärbaum so zu speichern, dass es während der Suche im Baum möglich ist, ganze Unterbäume abzuschneiden, die den nächsten Punkt nicht enthalten können.

**Aufbau des 3D-Baumes.** Gegeben sei eine Menge $P$ von 3D-Datenpunkten. Die erste Aufgabe besteht darin, den 3D-Baum zu konstruieren. Dabei soll $P$ in zwei ungefähr gleich große Regionen aufgespalten werden. Als Separierung dient eine Ebene, die parallel zu der $(x, y, 0)$, $(0, y, z)$ oder $(x, 0, z)$-Ebene verläuft und die durch den Datenmittelpunkt geht. Datenpunkte links neben der Ebene werden im linken Teilbaum gespeichert, während Punkte rechts neben der

Ebene im zweiten Teilbaum abgelegt werden. Da etwa gleich große Bereiche entstehen sollen, alterniert in der Regel die Lage der Ebene. In Anhang B.1.1 befindet sich der oben beschriebene Algorithmus im gekürzten Quellcode.

Das Verfahren teilt nicht in jedem Fall die Daten auf zwei gleich große Teilbäume auf. Dazu müsste man nicht den Datenmittelpunkt, sondern den Punkt finden, der bei gegebener Trennebene die Daten gleichmäßig zur Verteilung bringt. Ein solch zeitaufwendiger Rechenschritt wird nicht durchgeführt.

**Suche im 3D-Baum.** Nachdem der Baum aufgebaut wurde, müssen nun Funktionen zur Verfügung gestellt werden, um die nächsten Punkte zu finden. Dies führt zum Konzept der Reichweitensuche (engl.: *range search*). Jeder Knoten des Baumes repräsentiert eine Region durch einen Quader. Die Aufgabe besteht darin, zu einem 3D-Punkt den nächstgelegenen Punkt zu finden, dessen Abstand den Wert $d$ nicht überschreitet. Dazu wird um den gegebenen Punkt herum eine Suchregion der Größe $d \times d \times d$ aufgebaut. Falls die Suchregion vollständig innerhalb des Quaders eines Teilbaumes liegt, setzt sich die Suche nur dort fort. Dagegen werden beide Teilbäume durchsucht, falls die Suchregion auf beiden Seiten der Trennebene liegt. In Anhang B.1.2 sind Ausschnitte des Programmcodes des beschriebenen Algorithmus zu finden.

**Performanz von 3D-Bäumen.** Wenn die zu speichernden Datenpunkte annähernd gleichmäßig verteilt sind, erfolgt der Aufbau eines 3D-Baumes im Mittel in $\mathcal{O}(3n \log n)$. Die Suchzeit im Baum ist direkt proportional zu den Knoten, die während der Suche besucht worden sind. Die Suche erfolgt in $\mathcal{O}(n^{2/3})$ [85]. Der 3D-Baum benötigt $\mathcal{O}(3n)$ Speicher [85].

### 3.3.3 Transformationsschätzung

Der zweite Schritt des ICP-Algorithmus bestimmt bei gegebenen Punktpaaren eine Transformation, die die Fehlerfunktion $E(\mathbf{R}, \mathbf{t})$ (3.7) minimiert. Dieser Vorgang lässt sich mit Hilfe der folgenden physikalischen Analogie veranschaulichen: Man stelle sich vor, dass zwischen den Punktpaaren – wie in Abbildung 3.2 rot dargestellt – Federn gespannt sind. Wird dieses dynamische System sich selbst überlassen, ziehen die Federn den zweiten Scan (bei geeigneter Dämpfung) in eine Position, die ein Minimum an potenzieller Energie aufweist [54]. Die Rotation und Translation, die zu diesem Energieminimum führen, entsprechen denjenigen, die die Fehlerfunktion $E(\mathbf{R}, \mathbf{t})$ (3.7) minimieren.

Um das Minimum von $E(\mathbf{R}, \mathbf{t})$ zu finden, existieren verschiedene Strategien, die sich in direkte und indirekte Verfahren einteilen lassen. Zu den indirekten Verfahren zählt beispielsweise die Simulation des oben beschriebenen physikalischen Systems [19, 72]. Auch informierte Suchverfahren (z.B. Gradientenabstieg oder simuliertes Abkühlen) werden verwendet [11]. Diesen Verfahren stehen Methoden gegenüber, die die Transformation analytisch direkt aus den Punktpaaren berechnen. Vier Verfahren zur direkten Transformationsbestimmung sind zurzeit in Verwendung [51]:

- Die Transformationsschätzung mittels der Singulärwertzerlegung einer Matrix,

- Die Transformationsschätzung mit Hilfe von Orthogonal-Matrizen,
- Die Transformationsschätzung unter Verwendung des Einheitsquaternions,
- Die Transformationsschätzung mit Dualzahl-Quaternion.

Lorusso et. al. haben diese Verfahren untersucht und miteinander verglichen [51]. Sie sind alle in etwa gleich schnell ($\mathcal{O}$(Anzahl der Punktpaare), mit ähnlichen Konstanten), besitzen ähnliche Genauigkeiten (die Rechnergenauigkeit) und Stabilitäten gegenüber verrauschten Daten. Die implementierte Transformationsschätzung verwendet die Methode des Einheitsquaternions.

### Berechnung der Transformation mittels Einheitsquaternion

Zunächst wird die Berechnung der Rotation $\mathbf{R}$ von der Berechnung der Translation $\mathbf{t}$ getrennt, um die Rotation separat zu bestimmen. Die Translation findet man anschließend ausgehend von der Rotation. Eine solche Entkopplung ist möglich, weil für eine Lösung $(\hat{\mathbf{R}}, \hat{\mathbf{t}})$ von (3.7) gilt, dass $M = \{\mathbf{m}_i\}_{1,\ldots,N_m}$ und $\hat{D} = \{\hat{\mathbf{R}}\mathbf{d}_j - \hat{\mathbf{t}}\}_{1,\ldots,N_d}$ den gleichen Schwerpunkt haben.

Der erste Schritt berechnet zwei Punktmengen $M'$ und $D'$ von den Originalpunktmengen $M$ und $D$, indem von jedem Punkt der Schwerpunkt der Punktmengen subtrahiert wird. Es ist

$$\mathbf{c}_m = \frac{1}{N_m}\sum_{i=1}^{N_m}\mathbf{m}_i, \tag{3.11}$$

$$\mathbf{c}_d = \frac{1}{N_d}\sum_{j=1}^{N_d}\mathbf{d}_j \tag{3.12}$$

und

$$M' = \{\mathbf{m}'_i = \mathbf{m}_i - \mathbf{c}_m\}_{1,\ldots,N_m}, \tag{3.13}$$

$$D' = \{\mathbf{d}'_j = \mathbf{d}_j - \mathbf{c}_d\}_{1,\ldots,N_m}. \tag{3.14}$$

Die Gleichung (3.7) vereinfacht sich zu

$$E(\mathbf{R},\mathbf{t}) = \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\left|\left|\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_j\right|\right|^2. \tag{3.15}$$

Die Herleitung für obiges Resultat befindet sich in Anhang A.2.1. Nachdem eine Lösung für (3.15) bestimmt ist, ergibt sich die Translation als

$$\mathbf{t} = \mathbf{c}_m - \mathbf{R}\mathbf{c}_d.$$

Die gesuchte Rotation $\mathbf{R}$ findet man mit Hilfe der $3 \times 3$ Kovarianz-Matrix $\mathbf{M}$, die sich aus den Punkten von $M'$ und $D'$ bildet:

$$\mathbf{M} = \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\ \mathbf{m}'_i{\mathbf{d}'_j}^T.$$

Diese Matrix enthält *alle* notwendigen Informationen, um **R** unter Benutzung der Methode der kleinsten Quadrate zu berechnen. Die einzelnen Elemente von **M** bestimmen sich wie folgt:

$$\mathbf{M} = \begin{pmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{pmatrix}$$

mit

$$S_{xx} = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j}\ m'_{ix} d'_{jx}, \qquad S_{xy} = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j}\ m'_{ix} d'_{jy}, \qquad \dots \quad .$$

Eine symmetrische $4 \times 4$ Matrix **N** wird aus **M** berechnet:

$$\mathbf{N} = \begin{pmatrix} (S_{xx}+S_{yy}+S_{zz}) & (S_{yz}+S_{zy}) & (S_{zx}+S_{xz}) & (S_{xy}+S_{yx}) \\ (S_{yz}+S_{zy}) & (S_{xx}-S_{yy}-S_{zz}) & (S_{xy}+S_{yx}) & (S_{zx}+S_{xz}) \\ (S_{zx}+S_{xz}) & (S_{xy}+S_{yx}) & (-S_{xx}+S_{yy}-S_{zz}) & (S_{yz}+S_{zy}) \\ (S_{xy}+S_{yx}) & (S_{yz}+S_{zy}) & (S_{zx}+S_{xz}) & (-S_{xx}-S_{yy}+S_{zz}) \end{pmatrix}. \quad (3.16)$$

Die Eigenvektoren und die zugehörigen Eigenwerte von **N** müssen berechnet werden. Horn zeigt, dass das Einheitsquaternion $\dot{q} = (q_0, q_x, q_y, q_z)$ der gesuchten Rotation dem Eigenvektor des größten positiven Eigenwertes von **N** entspricht [40]. Der Beweis für diese Tatsache findet sich in Anhang A.2.2. Die Nullstellen des charakteristischen Polynoms der Matrix **N** sind die gesuchten Eigenwerte. Da das Polynom den Grad 4 hat, lassen sich die Eigenwerte direkt berechnen (Methode von Ferrari). Die gesuchte orthonormale Rotationsmatrix berechnet man anschließend aus dem Einheitsquaternion $\dot{q}$ mit (3.6).

### 3.3.4 Ergebnisse des Scanmatchings als Optimierungsproblem

Die vorgestellten Verfahren zur Bildung von Punktpaaren werden in zahlreichen Experimenten untersucht, wobei dazu auch Verfahren kombiniert werden. Die Experimente finden im GMD-Robobench, einem erweiterten Bürotrakt statt (FhG-AIS Gebäude C2 Erdgeschoss). Über einen langen Flur sind dort Büros miteinander verbunden. Der Roboter steht im Flur und nimmt einen 3D-Scan von $180 \times 256$ Punkten auf. Anschließend fährt er 2 Meter den Flur entlang und nimmt einen weiteren 3D-Scan auf. Das Scanmatching wird mit dem ICP-Algorithmus solange durchgeführt, bis die Änderung des mittleren Fehlers, also des Fehlers je Punktpaar, kleiner als 0.0001 Zentimeter ist. Eine so kleine Veränderung der Fehlerfunktion deutet auf ein lokales Minimum oder Plateau hin. Der Roboter bestimmt zunächst seine Position mittels der Odometrie. Diese Schätzung ergibt nach der 2-Meter-Fahrt die Werte $(x, y, \theta) = (27.10,\ 233.20,\ 6.64)$, die als Startwert für den ICP dienen. Der ICP-Algorithmus bildet nur Punktpaare, falls der Abstand zwischen den nächsten Punkten kleiner als 25cm ist, d.h.

$$w_{i,j} = \begin{cases} 1 & \text{wenn der nächste Punkt von } \mathbf{m}_i\ \mathbf{d}_j \text{ ist und der Abstand dieser Punkte} \\ & \text{weniger als 25cm beträgt,} \\ 0 & \text{sonst.} \end{cases}$$

Mit einem solchen harten Schwellenwert kann es sein, dass sich die Anzahl der gebildeten Punktpaare ($\sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j}$ (3.8)) verringert. Versuche bestätigen aber, dass durch diese Einschränkung das gesuchte Minimum schneller gefunden wird und man bessere Ergebnisse erhält. (Die Ungleichungskette (3.10) im Konvergenzbeweis ist in diesem Fall nicht erfüllt; Experimente zeigen aber, dass der Algorithmus trotzdem konvergiert.)

Abbildung 3.4 zeigt exemplarisch, wie mit dem ICP Algorithmus zwei 3D-Scans ineinander geschoben, also registriert werden. Weitere Abbildungen und Animationen befinden sich auf der CD

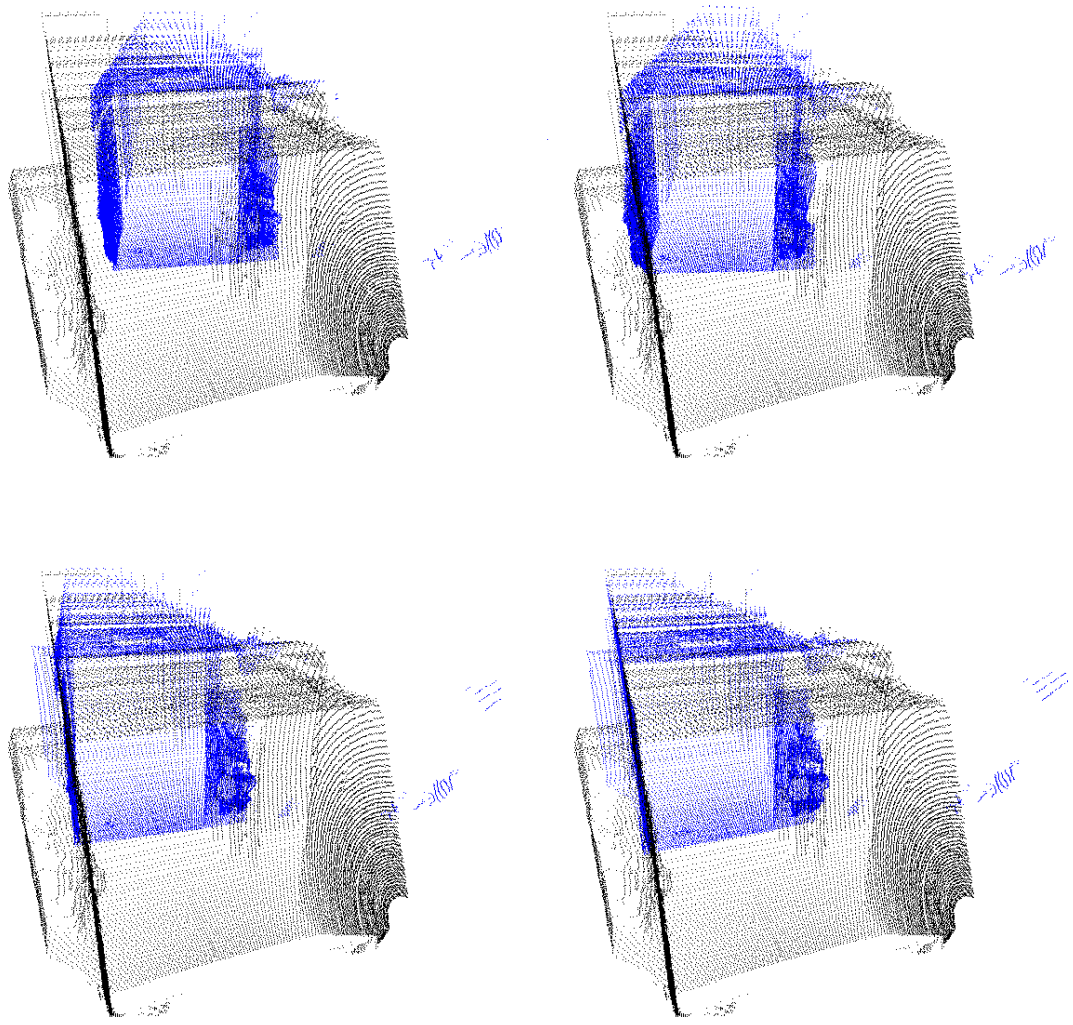

**Abbildung 3.4:** Visualisierung der Ergebnisse des ICP-Algorithmus. Die anfängliche Lage zweier 3D-Scans und die nach der ersten, dritten und 10. Iteration sind dargestellt. In jedem Iterationsschritt wird eine Rotation $\mathbf{R}$ und Translation $\mathbf{t}$ errechnet und auf den zweiten 3D-Scan angewendet.

Tabelle 3.1 stellt die Ergebnisse der Verfahren zur Punktpaarbildung gegenüber. Es wird die zur Bildung der Punktpaare benutzte Methode mit der dafür benötigten Zeit angegeben. Die Zeiten

wurden auf einem Pentium III mit 600MHz gemessen. Zusätzlich sind die Anzahl der Iterationen, die nötig waren um das Konvergenzkriterium zu erfüllen, der mittlere Fehler $E(\mathbf{R}, \mathbf{t})$ (3.7) sowie die Endposition des Roboters angegeben.

| Methode | Zeit | Iterationen | Fehler | Endkoordinaten |
|---|---|---|---|---|
| I – alle Punkte & brute force Suche | 3h 47m | 27 | 8.24 | (4.53, 221.00, -0.06) |
| II – RED. PUNKTE & brute force Suche | 3m 6s | 25 | 10.91 | (5.33, 220.55, -0.05) |
| III – alle Punkte & $k$D–Baum | 6s | 27 | 8.24 | (4.53, 221.00, -0.06) |
| IV – REDUZIERTE PUNKTE & $k$D–Baum | <2s | 25 | 10.91 | (5.33, 220.55, -0.05) |

**Tabelle 3.1:** Vergleich der Methoden zur Bildung von Punktpaaren; Ausgangsposition des ICP-Algorithmus ist die auf Odometrie-Daten beruhende Roboterposition $(x, y, \theta) = (27.10,\ 233.20,\ 6.64)$.

Abbildung 3.5 veranschaulicht die Entwicklung des mittleren Fehlers und die Anzahl der gepaarten Punkte in den ersten 20 Iterationen des Algorithmus. Die linke Abbildung zeigt die Entwicklung, falls alle Punkte Berücksichtigung finden, während in der rechten Abbildung nur die reduzierten Punkte einbezogen werden. Zu erkennen ist, dass beide Verfahren etwa gleich schnell konvergieren (in zirka 5 Schritten).

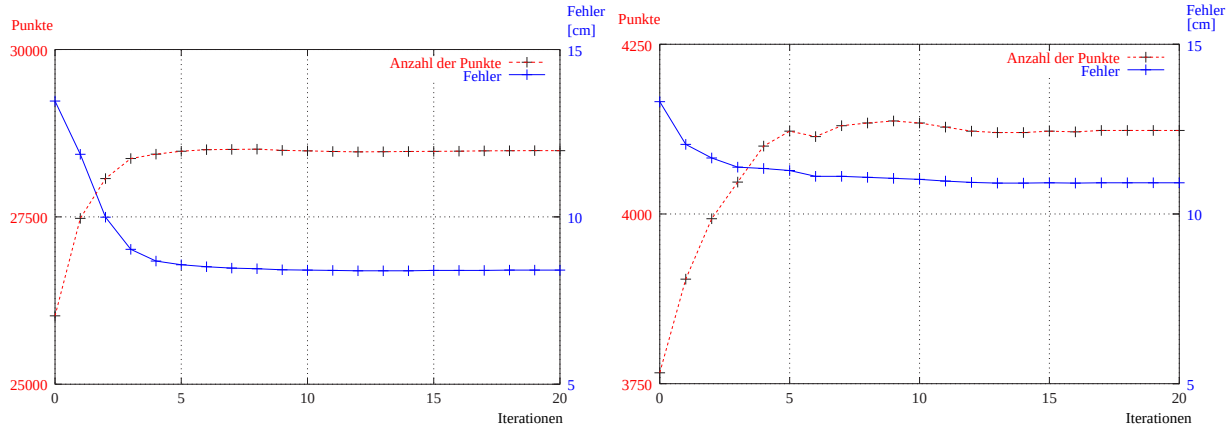


**Abbildung 3.5:** Die Entwicklung des mittleren Fehlers und die Anzahl der Punktpaare während des ICP-Algorithmus. Links: Methode I bzw. III. Rechts: Methode II bzw. IV.

Für die bisher präsentierten Ergebnisse wird wie beschrieben ein harter Schwellenwert verwendet und alle Punktpaare werden gleich gewichtet. Der Parameter $w_{i,j}$ ermöglicht allerdings auch eine Gewichtung der Punktpaare; die vorgestellte Transformationsbestimmung (Kapitel 3.3.3) wird dadurch nicht beeinflusst [40]. Neben der konstanten Verteilung der Gewichte erscheinen folgende Strategien logisch:

- Die Zuweisung von geringeren Gewichten zu Paaren mit größerem Punkt-Punkt Abstand. Dies ist vergleichbar mit der harten Schwellenwertmethode, allerdings werden Diskontinuitäten vermieden. Folgende Gewichte sind für den nächsten Test gewählt:

$$w_{i,j} = \begin{cases} 1 - \frac{||\mathbf{m}_i - \mathbf{d}_j||}{d_{max}} & \text{falls } \mathbf{d}_j \text{ der nächste Punkt zu } \mathbf{m}_i \text{ ist} \\ 0 & \text{sonst.} \end{cases}$$

  $d_{max}$ wurde wieder auf 25cm festgelegt.

- Große Gewichte werden Paaren zugewiesen, deren Remissionswert sich kaum unterscheidet. Sind nämlich die Intensitätswerte der Punkte annähernd gleich, ist es wahrscheinlicher, dass es sich um den gleichen Punkt handelt. $I(\mathbf{p})$ bezeichnet die Intensität eines Datenpunktes $\mathbf{p}$ . Folgende Formel wird verwendet:

$$w_{i,j} = \begin{cases} = & 1 - |I(\mathbf{m}_i) - I(\mathbf{d}_j)| & \text{falls } \mathbf{d}_j \text{ der nächste Punkt zu } \mathbf{m}_i \text{ ist,} \\ = & 0 & \text{sonst.} \end{cases}$$

| Methode | Zeit | Iterationen | Fehler | Endkoordinaten |
|---|---|---|---|---|
| IVa – RED. PUNKTE & dynamische Gewichte | <2s | 25 | 9.0 | (5.10, 221.38, -0.31) |
| IVb – RED. PUNKTE & Intensität | <2s | 27 | 10.61 | (6.14, 220.43, 0.51) |

**Tabelle 3.2:** Vergleich der Methoden zur Bildung von Punktpaaren (Forts.); Ausgangsposition des ICP-Algorithmus ist die auf Odometrie-Daten beruhende Roboterposition $(x, y, \theta) = (27.10,\ 233.20,\ 6.64)$; Zeiten bei brute-force Suche.

Die Benutzung von dynamischen Gewichten und die Verteilung der Gewichte mittels Intensitätswerten lassen sich auch mit $k$d-Bäumen kombinieren, wobei die Programmlaufzeit dann ebenfalls zirka 2 Sekunden beträgt (vgl. Tabelle 3.2). Abbildung 3.6 zeigt die Entwicklung des mittleren Fehlers und die Anzahl der benutzten Punktpaare für diese beiden Strategien. Wie man sieht, wird das Fehlerminimum wiederum während der ersten Iterationen erreicht.

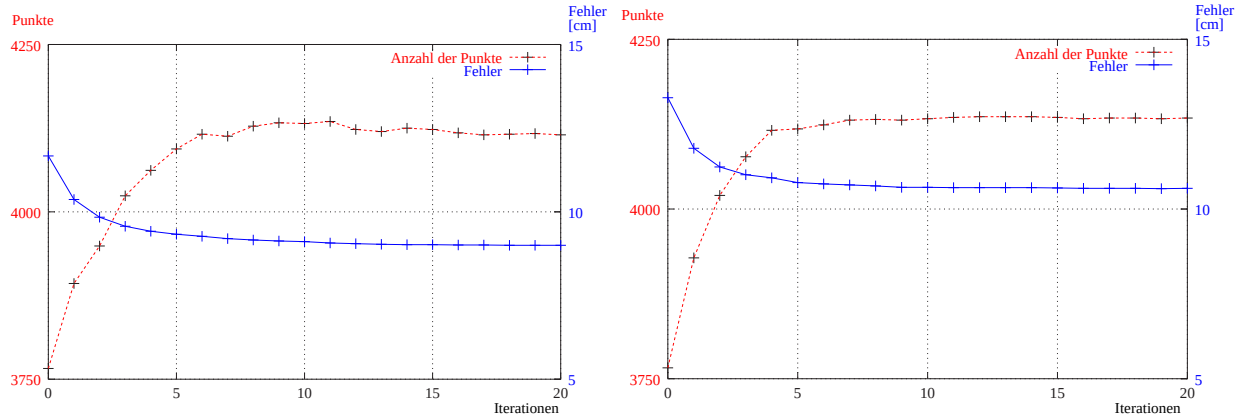


**Abbildung 3.6:** Die Entwicklung des mittleren Fehlers und die Anzahl der Punktpaare während des ICP-Algorithmus (Forts.). Links: Methode IVa. Rechts: Methode IVb.

Die angegebenen Endkoordinaten des Roboters können nicht mit einem Referenzwert verglichen werden, da die genaue Position des Roboters nicht bekannt ist. Der angegebene Fehler ist der Fehler $E(\mathbf{R}, \mathbf{t})$ (3.7) und nicht jener der Roboterposition.

### 3.3.5 Probabilistisches Scanmatching

Hähnel et al. schlagen ein probabilistisches Scanmatching vor [32] und betrachten dabei das korrekte Zusammenfügen von zwei 3D-Scans als Optimierungsproblem. Im Gegensatz zum ICP minimieren sie keine Fehlerfunktion, sondern maximieren die bedingte Wahrscheinlichkeit für einen 3D-Scan $S$ bei gegebener Roboterpose $l$ und einem zuvor aufgenommenen 3D-Scan $S'$:

$$\max\ P(S|S', l).$$

Um das Maximum zu finden, wird die Pose des Roboters solange verschoben, bis sich die Wahrscheinlichkeit nicht mehr verändert. Die anzuwendende Transformation berechnen Hähnel et al. nicht in geschlossener Form. Durch das Gradientenverfahren können sie nicht garantieren, dass das globale Maximum erreicht wird [32].

Durch die Einführung von Wahrscheinlichkeiten in das Scanmatching lassen die Fehlerwahrscheinlichkeiten des Scanner gut modellieren [32].

## 3.4 Kantenbasiertes Scanmatching

Sappa et al. schlagen ein merkmalbasiertes Scanmatching auf der Grundlage des ICP-Algorithmus vor [65]. Dabei sind die extrahierten Merkmale die Kanten in einem Scan. Um diese Kanten zu finden, zerlegen Sappa et al. einen 3D-Scan in horizontale und vertikale Schnitte und suchen dort nach Diskontinuitäten. Dabei unterscheiden sie so genannte Faltenkanten (engl.: *crease edge*) und Sprungkanten (engl.: *jump edge*). Faltenkanten entstehen durch die Änderung der Orientierung einer Fläche; die Ursache der Sprungkanten liegt im großen Abstand benachbarter Scanpunkte. Die Punkte in einem Schnitt werden durch quadratische Funktionen approximiert, wodurch Sappa et al. die zwei Kantentypen erkennen. Das Zerlegen eines 3D-Scans in horizontale und vertikale Schnitte garantiert, dass alle Kantenpunkte gefunden werden. Der ICP-Algorithmus registriert anschließend die Kantenpunkte, d.h. es kommt nur eine Teilmenge der vorhandenen Datenpunkte zum Einsatz.

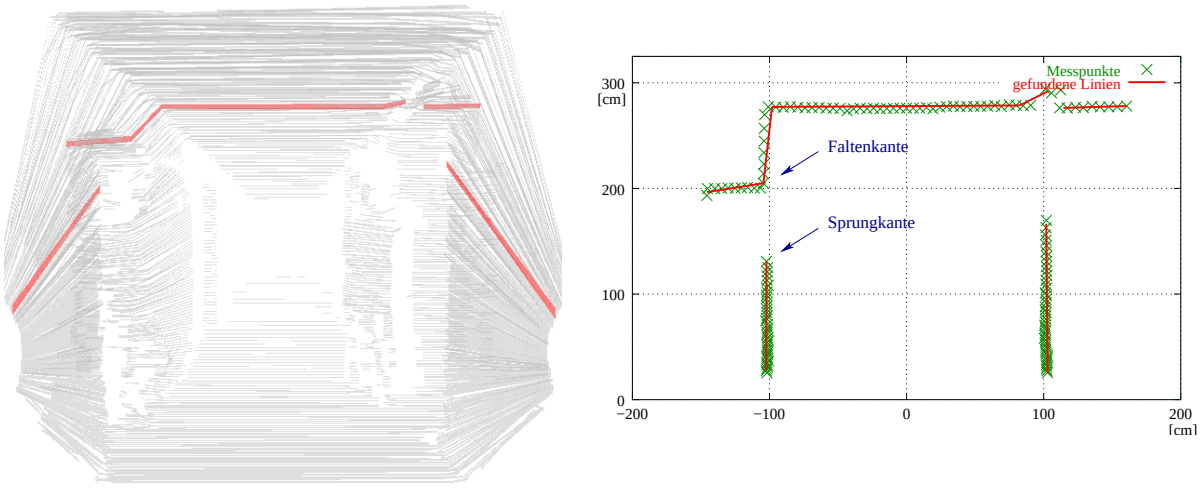


**Abbildung 3.7:** Liniendarstellung eines 3D-Scans und Kantenextraktion.

Der in dieser Arbeit eingesetzte Laserscanner (Abbildung 2.4) scannt jeweils eine Ebene, in der Linien erkannt werden. Die Endpunkte der Linien gehören entweder zu den Faltenkanten oder zu den Sprungkanten (vgl. Abbildung 3.7). Durch die Drehbewegung werden fast alle Kantenpunkte gefunden, eine zusätzliche Zerlegung der Scanpunkte in vertikale Schnitte erscheint nicht nötig. Die Abbildung 3.8 zeigt die auf der Basis von Linienerkennung extrahierten Kantenpunkte und

stellt die Ergebnisse der beiden in Kapitel 2.4.2 vorgestellten Linienerkenner gegenüber. Es wird deutlich, dass die mit Hough-Transformation erzeugten Kantenpunkte weniger stark verrauscht sind. Dies lässt sich auf die qualitativ besseren Linien der Hough-Transformation zurückführen.

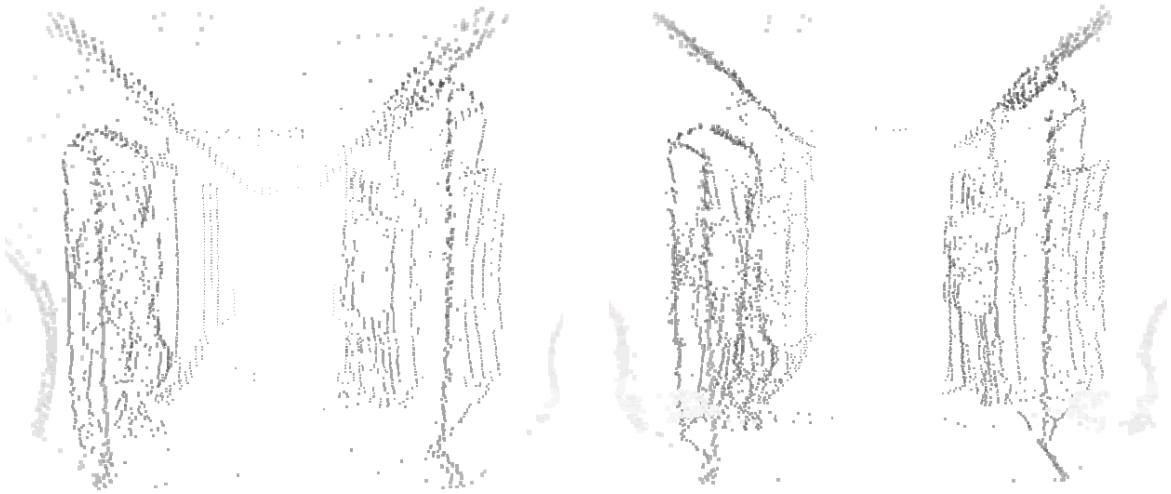

**Abbildung 3.8:** Kantenpunkte: Links: Matching-Algorithmus. Rechts: Hough-Transformation.

Es liegt nahe, den ICP-Algorithmus auf diese Punkte anzuwenden. Die 46080 Punkte des Beispielscans enthalten 3704 Kantenpunkte. Damit hat dieses Verfahren eine etwas bessere Performanz als die Verwendung der REDUZIERTEN PUNKTE (Datenreduktion etwa 1:12). Tabelle 3.3 und Abbildung 3.9 zeigen die Ergebnisse. Die angegebene Zeit setzt sich aus der für den ICP benötigten Zeit und jener, die für die Linienerkennung gebraucht wird, zusammen. Es dominiert der Aufwand für die Linienerkennung (Matching Algorithmus: 4s, Hough-Transformation: 2min 45s). Der hohe Aufwand bei der Hough-Transformation erklärt sich dadurch, dass sich der Bereich, auf dem Linien erkannt werden müssen, schlecht einschränken lässt, da von möglichst vielen Kantenpunkten der Scans Punktpaare gebildet werden sollen. Es ist eine Linienerkennung in einem Radius von 10 Metern nötig, um die Scans zu matchen, während die Hough-Transformation als online Algorithmus nur Punkte im Abstand von 5 Metern betrachtet [74, 75].

| Methode | Zeit | Iterationen | Fehler | Endkoordinaten |
|---|---|---|---|---|
| V – Kantenpunkte (Matching Alg.) | 6s (<2s) | 20 | 12.37 | (6.41, 227.92, -0.41) |
| VI – Kantenpunkte (Hough Transf.) | 2m 48s (3s) | 50 | 13.20 | (0.06, 221.81, -2.03) |

**Tabelle 3.3:** Vergleich der Methoden zur Bildung von Kantenpunktpaaren, Zeiten bei brute-force Suche und vollständiger Datenverarbeitung sowie Anteil für das Scanmatching (in Klammern). Ausgangsposition des ICP-Algorithmus ist die auf Odometrie-Daten beruhende Roboterposition $(x, y, \theta) = (27.10,\ 233.20,\ 6.64)$.

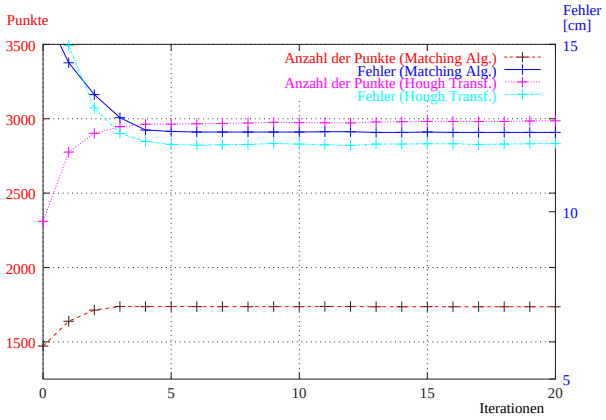


**Abbildung 3.9:** Die Entwicklung des mittleren Fehlers und die Anzahl der Kantenpunktpaare während des ICP-Algorithmus

## 3.5 Matching mehrerer 3D-Scans

Die vorangehenden Teile dieses Kapitels haben sich mit dem Matching von zwei 3D-Scans beschäftigt. Erst eine Registrierung vieler Scans erlaubt aber oftmals das vollständige Erfassen von Szenen. Daher wird im Folgenden das Matching mehrerer 3D-Scans behandelt. Die Registrierung von $n$ Punktmengen zu einem konsistenten Modell ist aktuelles Forschungsgebiet, in dem bereits einige Lösungsansätze präsentiert wurden [8, 11, 16, 19, 58, 72]:

**Paarweises Matching.** Auf zwei nacheinander aufgenommene 3D-Scans wird jeweils der iterative Algorithmus der nächsten Punkte (ICP) angewendet. Der zweite Scan benutzt zur Registrierung die Daten des ersten Scans, der dritte Scan die Daten des zweiten Scans, u.s.w.

Diese einfache Methode liefert eine Registrierung aller Scans unter der Voraussetzung, dass aufeinander folgende 3D-Scans genügend überlappen. Damit auch große Positions- oder Orientierungsänderungen des Roboters berücksichtigt werden können, wird in der implementierten Fassung dieses Algorithmus in einem Vorverarbeitungsschritt derjenige 3D-Scan mit dem größten Überlappungsbereich zu dem zu registrierenden Scan ermittelt. Der so bestimmte Scan dient dann als Basis für den zu matchenden Scan.

Bei einem solchen paarweisen Matching kumulieren Fehler. Da das Scanmatching vorangegangener Scans niemals perfekt sein kann, wird dieser Fehler an die nachfolgenden Scans weitergegeben. Anschließend entstehende Registrationsfehler addieren sich zu den vorangegangenen.

**Inkrementelles Matching.** Chen und Medioni schlagen vor, zuerst zwei Scans zu registrieren und diese anschließend zu einer einzigen Datenmenge (Metascan) zusammenzufassen [13]. Der nun folgende Scan wird mit diesem Metascan registriert, wodurch ein neuer Metascan entsteht. Bei dieser inkrementellen Matching Methode summieren sich ebenfalls Fehler.

Ein weiteres potenzielles Problem des inkrementellen Matchings ist in Abbildung 3.10 illustriert. Mehrere Scans sind zu einem Metascan zusammengefügt, es entsteht eine Schale begrenzter Dicke (Abbildung 3.10 (a)). Bei der Registrierung eines weiteren Scans wäre es wünschenswert, diesen in der Mitte der bereits vorhandenen Scans zu platzieren (Abbildung 3.10(b)). Je nach Implementation ist es wahrscheinlicher, dass der neue Scan sich an die äußere Schale anlagert. Dadurch verbreitert sich die Dicke der Schale von Scan zu Scan [58].

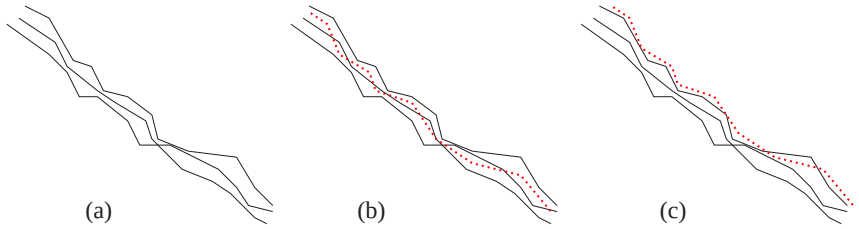


**Abbildung 3.10:** Ein Problem des inkrementellen Matchings. (a) Ausschnitt eines Metascans. (b) Beste Registrierung eines neuen Scans. (c) Wahrscheinliche Registrierung des neuen Scans. Abbildung entnommen aus [58].

Die beiden vorangegangenen Verfahren haben einen entscheidenden Nachteil: Wenn zusätzliche Scans hinzugefügt werden, besteht die Möglichkeit, dass diese neuen Scans Informationen mitbringen, die das bisherige Matching verbessern [58]. Dies wird erst in dem nachfolgend vorgestellten Verfahren berücksichtigt, das zusätzlich den entstehenden Registrationsfehler gleichmäßig über alle Scans verteilt.

**Simultanes Matching.** Benjemaa und Schmitt benutzen für das Matching vieler 3D-Scans eine iterative Methode [8]. Dabei spielen die Begriffe Masterscan und Nachbar-Scan eine entscheidende Rolle. Ein Masterscan ist ein 3D-Scan, der das globale Koordinatensystem definiert. Die Koordinatentransformationen zur Registrierung aller anderen 3D-Scans beziehen sich also immer auf diesen. Die Lage des Masterscans wird nicht verändert. Die Nachbar-Scans eines 3D-Scans sind diejenigen, die mit dem 3D-Scan Überlappungsbereiche aufweisen. Die Verwendung einer Menge von Nachbar-Scans statt der sämtlicher Scans ist wesentlich effizienter, da die Anzahl der zu untersuchenden Punkte eingeschränkt wird.

Benjemaa und Schmitt verwenden folgenden sequenziellen Algorithmus, um das Problem des Matchings mehrerer 3D-Scans zu lösen [8]:

Solange ein 3D-Scan seine Lage verändert, führe folgende Schritte aus:

1. (a) Berechne bzw. suche für jeden Punkt des aktuellen Scans den am nächsten gelegenen Punkt aus der Menge der Nachbar-Scans.
   (b) Berechne aus den Punktpaarungen die beste Transformation für den aktuellen Scan und wende diese an, falls es sich nicht um den Masterscan handelt.
2. Falls ein weiterer Scan vorhanden ist, benutze diesen als aktuellen Scan und wiederhole Schritt 1.

Das Verfahren iteriert vielfach über alle Scans und konvergiert dabei nur langsam, verteilt aber den Fehler gleichmäßig [8, 58].

Eggert et al. [19] und Stoddart/Hilton [72] benutzen eine ähnliche Methode. Sie paaren jeden Punkt eines Scans mit dem nächsten Punkt. Dabei suchen sie in allen weiteren Scans. Anschließend berechnen sie die Transformation simultan; ihr Algorithmus simuliert ein virtuelles (und vereinfachtes) dynamisches System aus gespannten Federn (vgl. Kapitel 3.3.3). Sie arbeiten nicht mit der besten Transformation weiter, um somit Fehler zu minimieren, deren Ursachen in falschen Punktpaaren liegen [19, 72].

Das implementierte simultane Matching ist vergleichbar mit dem Algorithmus von Benjemaa und Schmitt, basiert aber zusätzlich auf Ideen von Pulli [58]. Das Verfahren registriert die Scans in der Reihenfolge, in der sie aufgenommen wurden, fügt also einer Menge von Scans jeweils einen weiteren hinzu. Die wesentlichen Schritte des Algorithmus sind:

1. In einem Vorverarbeitungsschritt benutze paarweises Matching.
2. Initialisiere eine Liste mit dem zu registrierenden Scan.
3. Solange die Liste nicht leer ist, führe folgende Schritte aus:
   (a) Der aktuelle Scan ist das vorderste Element der Liste und dieser Scan wird aus der Liste entfernt.
   (b) Falls der aktuelle Scan nicht der Masterscan ist, bestimme die Nachbarn des Scans und registriere diesen aktuellen Scan anhand der Menge der Nachbar-Scans.
   (c) Falls der aktuelle Scan seine Lage ändert, füge die Nachbar-Scans der Liste hinzu.

Dieser Algorithmus befindet sich im Pseudocode in Anhang B.2. Der Vorverarbeitungsschritt (Schritt 1) des Algorithmus beschleunigt das Konvergieren des Verfahrens wesentlich.

### 3.5.1 Ergebnisse des Matchings mehrerer 3D-Scans

**Experimente im Flur (Robobench)**

Die drei Algorithmen werden im Flur des FhG-AIS Gebäudes getestet. Dabei ist das Szenario analog des Tests für zwei Scans (vgl. Kapitel 3.3.4), d.h. die Testbedingungen und Konstanten für Abbruchkriterium und maximalen Punktabstand sind exakt gleich gewählt. Der Roboter fährt in einem langen Flur. Nach jeweils 2 Metern (Odometriewert) stoppt er und nimmt einen 3D-Scan von 180 × 256 Punkten auf. Am Ende des Flurs dreht der Roboter und fährt zurück. Dabei werden ebenfalls 3D-Scans aufgenommen.

Das Scanmatching betrachtet nur die REDUZIERTEN PUNKTE und verwendet einen $k$D-Baum (Methode IV). Tabelle 3.4 stellt die Ergebnisse von paarweisem Matching, inkrementellen Matchings und simultanen Matchings gegenüber und vergleicht sie mit der über die Odometrie bestimmten Roboterposition. Es werden die Positionen nach 3, 13 und 20maligem Scannen angegeben. Nach Erreichen der 13. Scan-Position dreht der Roboter und fährt in Richtung seines Ausgangspunktes, da das Ende des Flurs erreicht ist. Es ist ersichtlich, dass die Positionen stark von einander abweichen.

| Position nach Odometrie | paarweises Matching (Methode IV) | inkrementelles Matching (Methode IV) | simultanes Matching (Methode IV) |
|---|---|---|---|
| (21, 446, 8.9) | (10, 416, 3.1) | (10, 416, 3.1) | (10, 414, 3.4) |
| (116, 2548, 4.7) | (89, 2559, 5.3) | (96, 2570, 5.8) | (99, 2573, 5.7) |
| (-109, 843, 191.9) | (30, 806, 186) | (29, 881, 186.8) | (26, 815, 186.8) |

**Tabelle 3.4:** Vergleich der bestimmten Roboterpositionen $(x, y, \theta)$. Position nach 3, 13 und 20 maligem Scannen und Fahren des Roboters auf dem Testgelände. Nach dem 13. 3D-Scan dreht der Roboter am Ende des Flurs.

Tabelle 3.5 zeigt die benötigten Zeiten für die gesamte Datenverarbeitung und den Anteil des Scanmatchings daran. Damit wird der Nachteil des simultanen Matchings sichtbar. Die benötigte Zeit liegt um eine Größenordnung höher als die der anderen Verfahren. Aus diesem Grund lässt sich das Verfahren nicht oder nur bedingt auf Robotern einsetzen.

| Odometrie | paarweises Matching (Methode IV) | inkrementelles Matching (Methode IV) | simultanes Matching (Methode IV) |
|---|---|---|---|
| 80s (0s) | 2m 7s (47s) | 2m 19s (59s) | 19m 4s (17m 44s) |

**Tabelle 3.5:** Zeit für Datenverarbeitung mit Anteil für das Scanmatching (in Klammern) der 3 Matchingstrategien.

Die Frage, welche der drei Matchingstrategien für mehrere Scans am besten ist, versucht Abbildung 3.11 zu beantworten. Sie zeigt den gescannten Flur, also die $20 \times 180 \times 256 = 921600$ Messpunkte aus der Vogelperspektive. Da die Messpunkte teilweise sehr dicht liegen, ist der gescannte Bereich als schwarze Fläche zu sehen.

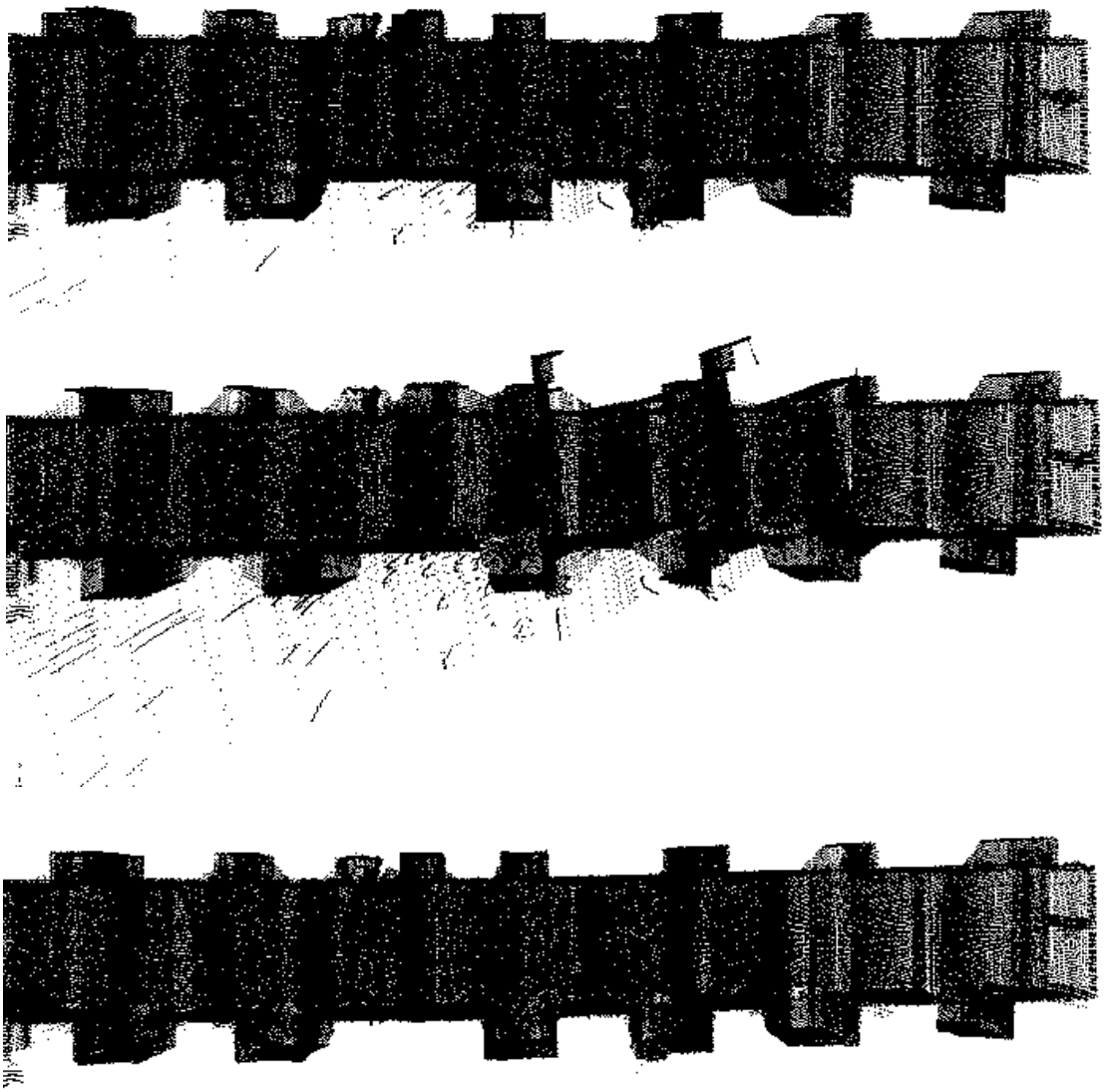

**Abbildung 3.11:** Vergleich der Matchingstrategien für das Scanmatching mehrerer 3D-Scans. Oben: Paarweises Scanmatching. Mitte: Inkrementelles Scanmatching. Unten: Simultanes Scanmatching.

In Abbildung 3.11 weist das inkrementelle Scanmatching (mittleres Bild) die meisten Artefakte auf und manche Kanten sind verrauscht. Das simultane Matching (unteres Bild) scheint die besten Ergebnisse zu liefern. Der folgende Abschnitt belegt eindeutig diese Vermutung. Dazu wird das Schloss Birlinghoven (Abbildung 3.12) von außen gescannt und vier 3D-Scans sollen zu einer konsistenten Szene zusammengefügt werden.

### Experimente am Schloss Birlinghoven

**Abbildung 3.12:** Schloss Birlinghoven

Die Scans des Schlosses (Abbildung 3.13) entstehen jeweils im Abstand von 10 Metern. Dieser Abstand wird lediglich per Schrittmaß geschätzt, ist also sehr ungenau. Die Algorithmen bekommen als Eingabe exakt die gleichen Daten. Die Gewichte der Punktpaare werden wie folgt gewählt:

$$w_{i,j} = \begin{cases} 1 & \text{wenn der nächste Punkt von } \mathbf{m}_i \ \mathbf{d}_j \text{ ist und der Abstand dieser Punkte} \\ & \text{weniger als 40cm beträgt,} \\ 0 & \text{sonst.} \end{cases}$$

Wenn die Veränderung des relativen Fehlers kleiner als 0.0001 Zentimeter ist, wird das Verfahren gestoppt.

Abbildung 3.14 zeigt die Ergebnisse der Registrierung der vier Scans aus Abbildung 3.13. Das inkrementelle Matching funktioniert in diesem Beispiel am schlechtesten.

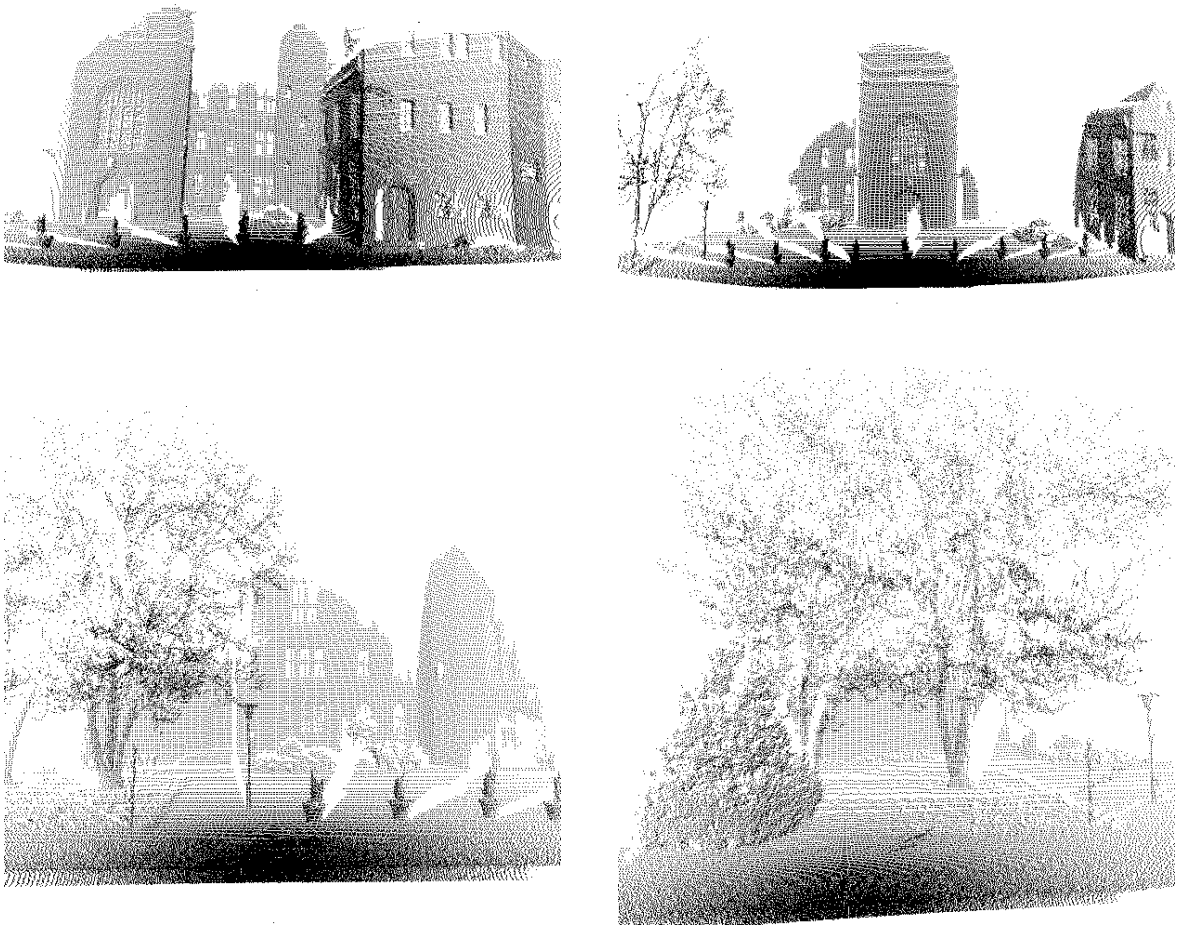

**Abbildung 3.13:** Vier 3D-Scans von Schloss Birlinghoven mit je 107520 Messpunkten.

**Abbildung 3.14:** Ergebnisse des Matchings mehrerer 3D-Scans. Links oben: Paarweises Matching. Rechts oben: Inkrementelles Matching. Unten: Simultanes Matching.

Offensichtlich registriert das simultane Matching am besten. So ist zum Beispiel dieses Verfahren in der Lage, die Blumenkübel (vgl. Foto in Abbildung 3.12) im Vordergrund richtig zusammenzufügen. Der entstehende Fehler wird gleichmäßig über alle Scans verteilt. Dadurch ergibt sich eine konsistente Szene.

### Das beste Scanmatchingverfahren

Der letzte Abschnitt hat eindeutig das simultane Scanmatching als die beste Methode für das Matching mehrerer 3D-Scans identifiziert. Wegen der hohen Laufzeit des simultanen Matchings von fast einer Minute pro aufgenommenem 3D-Scan, ist das paarweise Matching dem simultanen Matching zur Korrektur der Roboterpose vorzuziehen. Bei Verwendung des paarweisen Matchings werden zirka 2 Sekunden für das Scanmatching benötigt.

Für die obige Bestimmung des besten Verfahrens für das Matching mehrerer 3D-Scans wurde die Methode IV verwendet, also REDUZIERTE PUNKTE in einem $k$D-Baum. Um das beste Scanmatchingverfahren für die Anwendung auf dem Roboter zu ermitteln, muss untersucht werden, wie die in den Abschnitten 3.3.4 und 3.4 dieses Kapitels vorgestellten anderen Methoden das Matching mehrerer 3D-Scans beeinflussen. Dazu werden die Matchingmethoden III (alle Messpunkte und $k$D-Baum), IVa (REDUZIERTE PUNKTE mit dynamischen Gewichten und $k$D-Baum), IVb (REDUZIERE PUNKTE mit Intensitätswerten und $k$D-Baum), V (Kantenpunkte aus Matching-Algorithmus und $k$D-Baum) und VI (Kantenpunkte aus Hough-Transformation und $k$D-Baum) unter Benutzung paarweisen Matchings auf den 3D-Scans des Flurs ausgeführt. Methode I und II (brute force Suche) kommen wegen ihrer hohen Laufzeit nicht in Betracht.

| simultanes Matching (Methode IV) | paarweises Matching (Methode III) | paarweises Matching (Methode IVa) | paarweises Matching (Methode IVb) |
|---|---|---|---|
| (10, 414, 3.4) | (9, 416 , 3.1) | (10, 413, 3.1) | (10, 412, 3.1) |
| (99, 2573, 5.7) | (106, 2556, 5.9) | (80, 2508, 4.5) | (67, 2529, 3.4) |
| (26, 815, 186.8) | (32, 904, 187.6) | (28, 797, 182.6) | (31, 793, 186.0) |

| paarweises Matching (Methode V) | paarweises Matching (Methode VI) |
|---|---|
| (-1, 439, 16) | (5, 412, 3.9)) |
| (108, 2569, 6.3) | (67, 2529, 3.4) |
| (-113, 842, 191.1) | (-118, 836, -193.4) |

**Tabelle 3.6:** Vergleich der bestimmten Roboterpositionen $(x, y, \theta)$. Position nach 3, 13 und 20maligem Scannen und Fahren des Roboters auf dem Testgelände. Nach dem 13. 3D-Scan dreht der Roboter am Ende des Flurs.

Tabelle 3.6 stellt die Ergebnisse für den Flur dar. Diese werden von den Abbildungen 3.15 und 3.16 verdeutlicht. Es ist die gescannte Flur-Szene zu sehen. Ein Vergleich der Abbildung 3.11 oben (Methode VI) mit Abbildung 3.15 oben (Methode III) ergibt, dass die Verwendung aller Messpunkte keinen qualitativen Vorteil bringt. Das Benutzen von dynamischen Gewichten oder Intensitätswerten (vgl. Abbildung 3.15 mitte und unten mit Abbildung 3.11 oben) produziert in etwa gleich gute Ergebnisse wie die Verwendung eines harten Schwellenwertes. Abbildung 3.16

zeigt, dass die Verwendung von Kantenpunkten ungeeignet für das Scanmatching ist. Dieses Ergebnis lässt sich aber etwas verbessern, indem andere Konstanten für das Abbruchkriterium und den maximalen Punktabstand verwendet werden.

| simultanes Matching (Methode IV) | paarweises Matching (Methode III) | paarweises Matching (Methode IVa) | paarweises Matching (Methode IVb) |
|---|---|---|---|
| 19m 4s (17m 44s) | 4m 5s (2m 1s) | 2m 1s (41s) | 2m 3s (43s) |

| paarweises Matching (Methode V) | paarweises Matching (Methode VI) |
|---|---|
| 2m 25s (45s) | 54m (40s) |

**Tabelle 3.7:** Zeit für Datenverarbeitung mit Anteil für das Scanmatching bei unterschiedlichen Methoden zur Bildung von Punktpaaren.

**Fazit:** In den durchgeführten Experimenten bestätigt sich die in der Literatur geäußerte Vermutung, dass das simultane Matching für die Registrierung mehrerer Scans am besten funktioniert [58]. Leider ist das Verfahren sehr rechenaufwendig, so dass auf dem Roboter paarweises Matching eingesetzt werden muss. Bei den Strategien zur Bildung von Punktpaaren liefert Methode IV, also die Verwendung REDUZIERTER PUNKTE und des $k$D-Baums gute Ergebnisse und ist sehr schnell. Die Gesamtgenauigkeit des Scanmatchings lässt sich durch das Abbruchkriterium und den maximal zulässigen Punktabstand steuern. Grob gilt hierbei: Sind diese Konstanten klein, ist das Matching genauer. Zu kleine Konstanten führen dagegen aber oftmals zu sehr schlechten Ergebnissen.

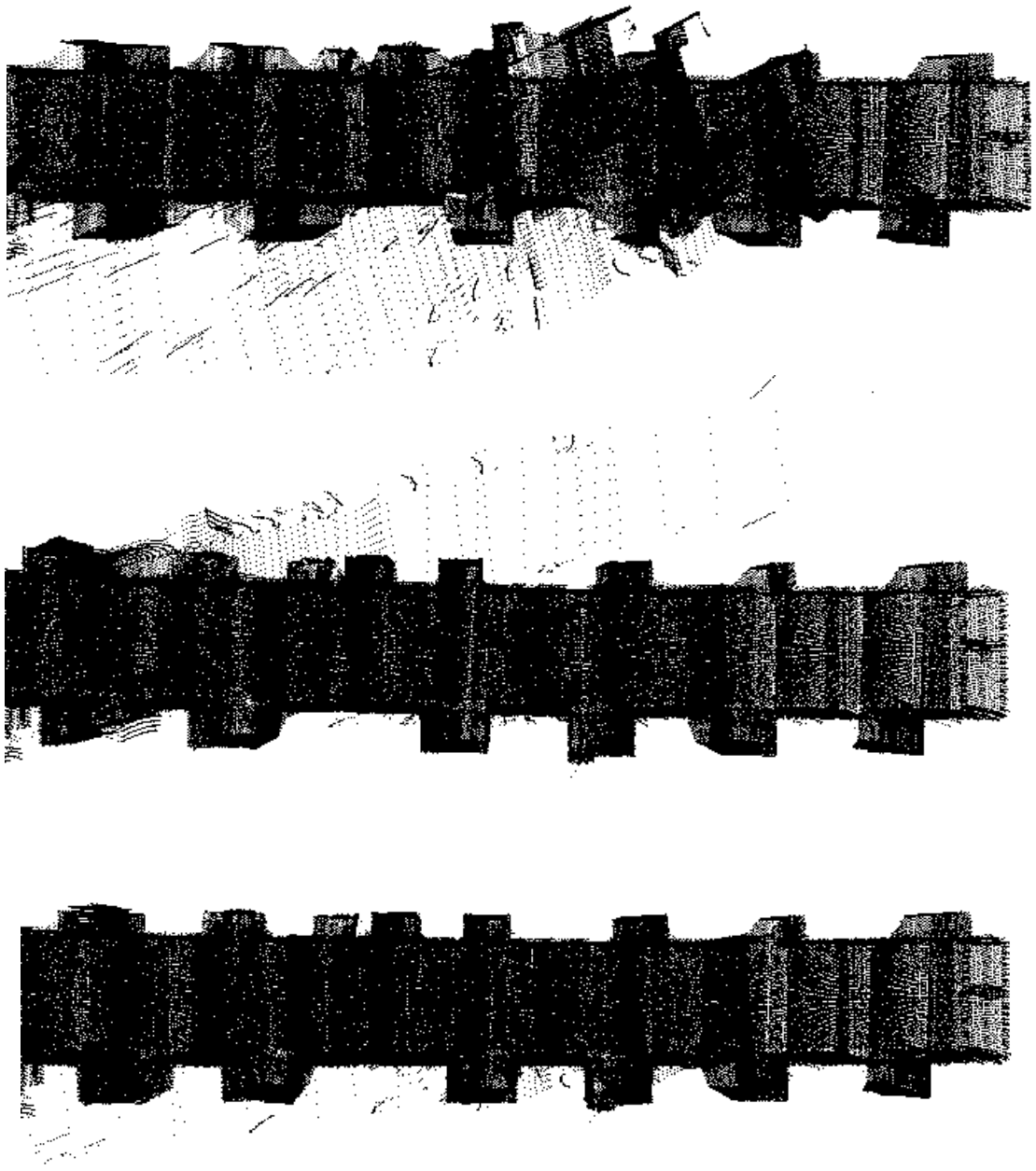

**Abbildung 3.15:** Vergleich der Matchingstrategien für das Scanmatching mehrerer 3D-Scans. Oben: Paarweises Scanmatching mit allen Messpunkten (Methode III). Mitte: Paarweises Matching mit dynamischen Gewichten der Punktpaare (Methode VIa). Unten: Paarweises Matching unter Berücksichtigung der Intensitätswerte (Methode VIb).

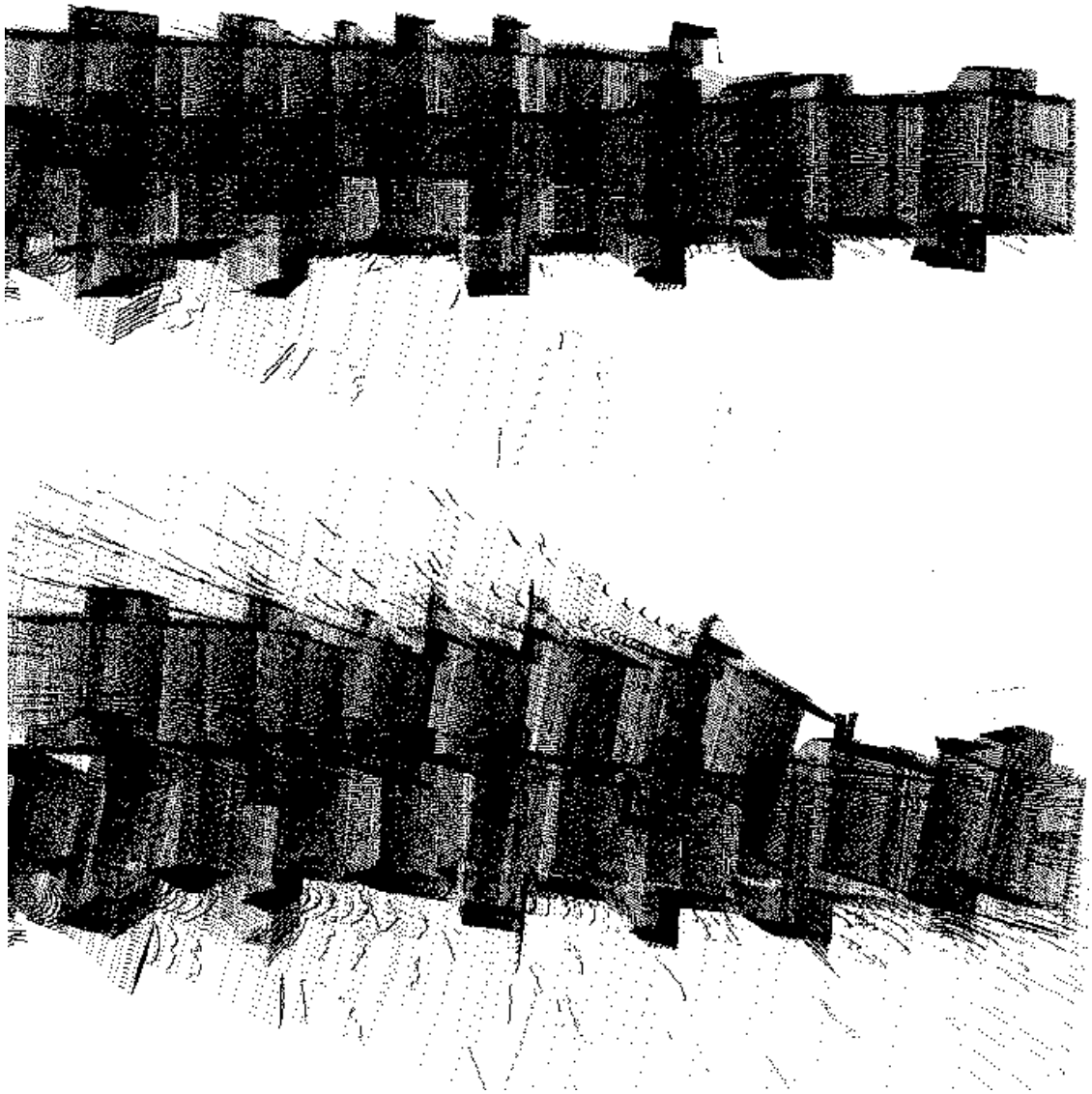

**Abbildung 3.16:** Vergleich der Matchingstrategien für das Scanmatching mehrerer 3D-Scans. Oben: Paarweises Scanmatching mit Kantenpunkten des Matching-Algorithmus (Methode V). Unten: Paarweises Scanmatching mit Kantenpunkten der Hough-Transformation (Methode VI).

# Kapitel 4

# Die optimale nächste Scanposition

Der Aufbau kompletter digitaler 3D-Modelle erfordert nicht nur das Zusammenfügen von 3D-Scans, sondern auch die Berechnung der optimalen nächsten Scanposition. Diese sollte einerseits möglichst viel Information über die Umgebung liefern und andererseits für den Roboter einfach anzufahren sein. Die dazu nötigen Algorithmen erläutert das nun folgende Kapitel.

## 4.1 Problemdefinition

Automatische Modellkonstruktion ist eine fundamentale Aufgabe in der Robotik [29]. Die Problemstellung lässt sich einfach formulieren: Wird ein Roboter in einer ihm unbekannten Umgebung ausgesetzt, muss er die Umgebung mit seinen Sensoren erfassen, um anschließend ein vollständiges digitales Modell von ihr erzeugen zu können. Ein solches Umgebungsmodell ermöglicht es einem automatischen System beispielsweise über die Umgebung zu kommunizieren oder Aktionen darin zu reproduzieren. In der vorliegenden Arbeit entsteht dieses Modell durch das Zusammenfügen unterschiedlicher Tiefenbilder, die der auf dem Roboter montierte 3D-Laserscanner von verschiedenen Positionen aus aufnimmt. Folglich stellt sich nach jedem Scan immer wieder die Frage: *Wohin muss der Roboter fahren, um eine möglichst optimale Position für den nächsten 3D-Scan zu erreichen?* Die Antwort auf das Explorationsproblem geben so genannte Nächste-beste-Sicht (engl.: *next best view*) Algorithmen. Sie ermitteln die optimale nächste Scanposition. Die Optimalität wird dabei anhand definierter Kriterien bestimmt (vgl. Kapitel 4.3 Gleichung (4.1)).

Traditionelle Nächste-beste-Sicht-Algorithmen sind nicht besonders gut für die Robotik geeignet [29]. Sie kombinieren zuerst Teilmodelle und schließen entstehende Lücken durch neue Aufnahmen. Dabei wird vorausgesetzt, dass sich ein Sensor völlig frei um das Objekt bewegen kann [7, 17, 60, 61]. In der Robotik hat der Sensor nur eine begrenzte Anzahl von Freiheitsgraden und befindet sich innerhalb der konvexen Hülle der Szene. Zusätzlich müssen mögliche Kollisionen vermieden werden. Aufgrund von physikalischen Einschränkungen, wie zum Beispiel Schlupf und Reibung, ist die Positionsbestimmung eines Roboters schwierig. Sie können allerdings durch effektive Registrationsalgorithmen (vgl. Kapitel 3) ausgeglichen werden. Man spricht in diesem

Zusammenhang vom simultanen Lokalisations- und Kartierungsproblem (engl.: *simultaneous localization and map building problem* kurz: *SLAM*).

Es wird nun darum gehen, einen Algorithmus für die Nächste-beste-Sicht – im Folgenden mit NBV bezeichnet – herzuleiten. Als Ausgangspunkt dient ein Algorithmus für Roboter, die den Grundriss ihrer Umgebung kennen (Abschnitte 4.2 und 4.2.1). Anschließend erfolgen Erweiterungen für das Explorationsproblem und für den 3D-Fall (Abschnitt 4.3). Schließlich beschäftigt sich Abschnitt 4.4 mit dem Erreichen des berechneten neuen Scanpunktes.

## 4.2 Das Kunstausstellungsproblem

Ein mit einer 2D-Karte seiner Umgebung ausgestatteter Roboter soll so gesteuert werden, dass er die gesamte 2D-Umgebung sieht. Gesucht ist also eine Menge von Positionen, von denen die gesamte Karte überblickt werden kann. Fährt der Roboter diese nacheinander an und führt dort Sensoroperationen aus, ist dadurch die Karte verifiziert. Das Berechnen der notwendigen Positionen heißt Kunstausstellungsproblem (engl.: *art gallery problem*). Die Analogie liegt auf der Hand: Sei die gegebene 2D-Karte der Grundriss einer Kunstausstellung. Dann entsprechen die Positionen den Orten, wo Wachpersonen postiert werden sollten, um alle Teile der Ausstellung einsehen und beschützen zu können[1].

Die 2D-Karte wird als Polygon $\mathcal{P}$ repräsentiert, das als eine Menge von $n$ Eckpunkten $v_1, v_2, \ldots, v_n$, und $n$ Kanten $v_1v_2, v_2v_3, \ldots, v_{n-1}v_n, v_nv_1$ (Kantenzug) definiert ist. Es handelt sich also um eine geschlossene, endliche, verbundene Region einer Ebene, die vom genannten Kantenzug umschlossen ist. Ein so definiertes Polygon besitzt keine Löcher. Alle nun folgenden Aussagen beziehen sich auf diesen Fall. Am Ende des Kapitels werden die Resultate für Polygone vorgestellt, die durch Kantenzüge begrenzt sind. Man sagt, ein Punkt $x \in \mathbb{R}^2$ sieht Punkt $y \in \mathbb{R}^2$, wenn das Liniensegment $xy$ vollständig in $\mathcal{P}$ liegt ($xy \subset \mathcal{P}$).

Für das Kunstausstellungsproblem gibt folgender Satz eine Obergrenze für die Anzahl der benötigten Wachpersonen. Die Wachposten haben ein 360°-Blickfeld und können unbegrenzt weit schauen [55].

**Satz 2 (Art Gallery Theorem).** Eine Kunstausstellung kann immer von $\lfloor \frac{n}{3} \rfloor$ Wachposten bewacht werden [55].

**Beweis:** Bezeichne $g(\mathcal{P})$ die Anzahl der benötigten Wachleute. Damit beweisen folgende Teilaussagen den Satz:

- $g(\mathcal{P}) \geq \lfloor \frac{n}{3} \rfloor$

  Abbildung 4.1 zeigt zwei Polygone mit den jeweils notwendigen Wachposten. Wegen ihrer Form heißen die Polygone Kronen. Jede Krone benötigt je Zacke, die wiederum aus 3 Ecken besteht, einen Wachposten. Induktion beweist, dass derartige Polygone mindestens $\lfloor \frac{n}{3} \rfloor$ Wächter benötigen.

[1] Bemerkung: Im musée du Louvre in Paris werden 2D-Laserscanner zur Bewachung der Gemälde eingesetzt. Sie sind an allen Wänden aufgestellt und tasten parallel zu diesen eine Ebene ab. Ein Alarm wird ausgelöst, falls die gescannte Ebene sich verändert.

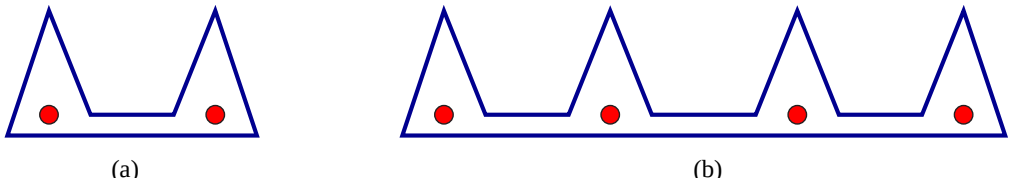


**Abbildung 4.1:** Wachposten in Polygonen: (a) 6 Ecken, 2 Wachposten; (b) 14 Ecken, 4 Wachposten.

- $g(\mathcal{P}) \leq \lfloor \frac{n}{3} \rfloor$

  Mit einem bemerkenswert einfachen Trick gelingt dieser Nachweis. Jedes Polygon lässt sich durch das Hinzufügen von inneren Diagonalen triangulieren (vgl. Anhang A.3). Anschließend wird die entstandene Triangulation als Graph betrachtet. Dieser Graph ist planar und kann nach dem berühmten 4-Farben-Theorem mit vier Farben eingefärbt werden. Der Beweis des 4-Farben Theorems war der erste, der vollständig vom Computer geführt wurde [45]. Einfärben des Triangulationsgraphs bedeutet, Knoten mit Farben zu beschriften, wobei keine zwei Knoten, die durch eine Kante verbunden sind, die gleiche Farbe erhalten. Anhang A.3 zeigt, dass der aus einer Triangulation entstandene Graph sich sogar mit nur 3 Farben kolorieren lässt. Abbildung 4.2 zeigt exemplarisch einen eingefärbten Triangulationsgraphen.

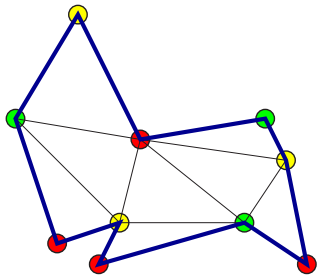

**Abbildung 4.2:** Einfärbung einer Triangulation mittels Triangulationsgraph. 3 Farben werden benötigt.

  Um den Beweis zu beenden, genügt die Feststellung, dass eine der Farben nicht öfter als an 1/3 der Ecken benutzt wird. O.B.d.A. sei diese Farbe grün. An jeder grünen Ecke postiere man eine Wache. Somit ist gezeigt, dass maximal $\lfloor \frac{n}{3} \rfloor$ Wachposten notwendig sind.

□

Der letzte Satz liefert eine Obergrenze für das Kunstausstellungsproblem bzw. für den vorliegenden Fall der Robotik ein oberes Limit für die Anzahl der Sensoroperationen zum vollständigen Erfassen der 2D-Umgebung bei gegebener Karte. Ziel ist es, die Anzahl der benötigten Sensoroperationen gering zu halten, also im Kunstausstellungsfall möglichst wenig Wachpersonen einzusetzen. Wächter überschauen so genannte Sternpolygone; jeder Punkt dieses Polygons lässt sich mittels einer Linie, die die Kanten des Polygons nicht schneidet, mit dem Wachposten (auch Kern genannt) verbinden. Die einzelnen Sternpolygone dürfen sich dabei überlappen. Es ist folglich nicht nach einer Partition gesucht.

Formal lässt sich das Kunstausstellungsproblem auf diese Weise definieren:

**StarC:**

*Instanz:* Ein Polygon $\mathcal{P}$ und eine positive Konstante $K$.

*Frage:* Gibt es eine Überdeckung von $\mathcal{P}$ mit $K$ oder mit weniger als $K$ Sternpolygonen, d.h. existieren Sternpolygone $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$ mit $k \leq K$, so dass $\mathcal{S}_1 \cup \mathcal{S}_2 \cup \ldots \cup \mathcal{S}_k = \mathcal{P}$?

Im weiteren Verlauf wird gezeigt, dass dieses Problem NP-hart ist. Für den Nachweis der NP-Härte wird nachfolgend das Problem 3-SAT definiert und danach auf StarC reduziert. Cook hat 1971 gezeigt, dass das 3-SAT NP-vollständig ist [47].

**3-SAT:**

*Instanz:* Eine Menge von booleschen Variablen $U = \{\mathsf{u}_1, \mathsf{u}_2, \ldots, \mathsf{u}_n\}$ und eine Menge von Klauseln (Disjunktion von Literalen) $C = \{\mathsf{c}_1, \mathsf{c}_2, \ldots, \mathsf{c}_m\}$, definiert über der Menge $U$. Jede Klausel ist eine Disjunktion von 3 Literalen.

*Frage:* Gibt es eine erfüllende Belegung von $C$, d.h. gibt es eine Belegung der $n$ Variablen, so dass $\mathsf{c}_1 \wedge \mathsf{c}_2 \wedge \cdots \wedge \mathsf{c}_m$ *wahr* wird?

**Satz 3 (Art Gallery is NP-hard).** StarC ist NP-hart [55].

**Beweis:** 3-SAT wird in polynomieller Zeit auf StarC reduziert (3-SAT$\leq_p$StarC). Für jede mögliche 3-SAT Formel lässt sich ein Polygon konstruieren, wobei eine Lösung des StarC Problems, also die Postierung der Wachposten beim Kunstausstellungsproblem, einer Lösung von 3-SAT entspricht.

Der vollständige Beweis befindet sich in Anhang A.4. □

Da das Kunstausstellungsproblem NP-hart ist, ist dieses Problem und somit auch die Berechnung einer minimalen Menge von Sensorpositionen nach dem heutigen Wissensstand nicht in polynomieller Zeit zu lösen. Der folgende Abschnitt beschreibt eine Approximation der Berechnung von minimal notwendigen Sensorpositionen.

### 4.2.1 Ein randomisierter Approximationsalgorithmus für das Kunstausstellungsproblem

Banos et al. reduzieren das Kunstausstellungsproblem auf ein weiteres NP-vollständiges, nämlich das Set-Cover Problem [28, 29, 30]. Es ist wie folgt definiert [47]:

**Set-Cover:**

*Instanz:* Ein Tupel $(X, F)$, wobei $X$ eine beliebige endliche Menge ist. $F$ ist eine Teilmenge der Potenzmenge von $X$, also $F \subset \mathfrak{P}(X)$ und es gilt $X = \bigcup_{s \in F} s$.

*Frage:* Bestimme $C \subset F$, so dass $X = \bigcup_{s \in C}$ gilt und $C$ minimale Kardinalität hat. $C$ ist also die minimale Menge, die alle Elemente von $X$ überdeckt.

Die Reduktion auf das Set-Cover Problem geschieht randomisiert. Der Roboter ist mit einer Karte seiner Umgebung ausgestattet. Im ersten Schritt des Algorithmus erfolgt die Generierung einer relativ großen Menge von Kandidatenpositionen. Dabei werden die Positionen zufällig ermittelt. Zudem besitzt jeder Punkt innerhalb der Karte die gleiche Wahrscheinlichkeit. Anschließend wählt der Algorithmus aus dieser Menge der Kandidatenpositionen eine minimale Menge, so dass die ganze Karte überblickt wird. Damit ist das Problem auf Set-Cover reduziert, d.h. $X$ ist die Karte und die Menge $F$ enthält die Teile der Karte, die von den Kandidatenpositionen überschaut werden können. Der randomisierte Approximationsalgorithmus ist:

1. Generiere zufällig $n_K$ Kandidatenpositionen innerhalb der als Polygon $\mathcal{P}$ gegebenen Karte der Roboterumgebung.
2. Berechne für jede Kandidatenposition den Teil des Polygons $\mathcal{P}$, der von dieser Position aus sichtbar ist.
3. Löse das Set-Cover Problem, d.h. bestimme eine minimale Menge von Positionen, so dass das ganze Polygon $\mathcal{P}$ eingesehen wird.

Schritt 2 dieses Algorithmus ermöglicht, dass Eigenschaften von realen Sensoren, wie beispielsweise minimaler und maximaler Abstand zu Objekten oder Öffnungswinkel, berücksichtigt werden können.

Im dritten Schritt muss ein NP-vollständiges Problem gelöst werden. Glücklicherweise gibt es einen Greedy-Approximationsalgorithmus dafür, der sehr gute Ergebnisse liefert. Folgende Schritte werden bei einer Eingabe von $(X, F)$ ausgeführt:

1. Setze $U = X$ und $C = \{\}$.
2. Solange $U \neq \{\}$ ist, führe folgende Schritte aus:
   - Bestimme $S \in F$, das $\|S \cap U\|$ maximiert.
   - Setze $U = U \setminus S$ und $C = C \cup S$.

Durch die Schleife hat der Algorithmus eine Laufzeit von $\mathcal{O}(|X|\,|F|\,\min(|X|\,|F|))$. Nach [14] gibt es sogar eine Implementation, die nur lineare Zeit benötigt. Die Approximation mittels Greedy-Algorithmus liefert Ergebnismengen, deren Anzahl nicht sehr viel größer als das optimale Set-Cover ist [14]. Der folgende Satz gibt darüber Auskunft, doch zuvor muss die Approximationsgüte (engl.: *approximation ratio*) definiert werden. Man sagt, ein Approximationsalgorithmus für ein Problem hat die Approximationsgüte $\varrho(n)$, wenn für jede Instanz der Größe $n$ die Kosten $C$ der Lösung, die der Algorithmus liefert, innerhalb des Faktors $\varrho(n)$ der Kosten $C^*$ der optimalen Lösung liegen [14]:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \varrho(n).$$

**Satz 4 (Approximationsgüte von Set-Cover).** Der Greedy-Algorithmus für Set-Cover hat eine Approximationsgüte von $(\ln(|X|) + 1)$.

**Beweis:** Siehe [14] oder [47]. □

Nachdem die optimalen Scanpositionen bei gegebener Karte approximiert worden sind, müssen diese vom Roboter nacheinander angefahren werden. Dieses Problem entspricht demjenigen des Handlungsreisenden (engl.: *Travelling Salesman Problem*).

**TSP (euklidisch):**

*Instanz:* Eine Menge von Wegpunkten in der Ebene $U = \{u_1, \ldots, u_n\} \subset \mathbb{R}^2$ mit $u_i = (x_i, y_i)$.

*Frage:* Bestimme eine Permutation $\sigma$ über der Menge $\{1, \ldots, n\}$, so dass die Summe der Strecke $\sum_{i=1}^{n-1} d(u_{\sigma(i)}, u_{\sigma(i+1)}) + d(u_{\sigma(n)}, u_{\sigma(1)})$ minimal ist.

Dieses Problem ist ebenfalls NP-vollständig, lässt sich aber auch sehr gut approximieren [14, 47].

Der nächste Abschnitt beschreibt den implementierten Algorithmus zum Finden der optimalen nächsten Scanposition. Er basiert auf Ideen aus obigem Approximationsalgorithmus. Dabei werden Kandidatenpunkte in die bereits explorierte Umgebung gestreut und anschließend ähnlich dem Greedy-Verfahren der Punkt ausgewählt, der den größen Informationsgewinn verspricht. Da der Roboter aber in diesem Fall keine vollständige Karte der Umgebung hat, handelt es sich um ein Explorationsproblem. Die dreidimensionale Umgebung des Roboters soll vollständig erfasst werden.

## 4.3 Die Berechnung der optimalen nächsten Scanpositionen

Die optimale nächste Scanposition soll berechnet werden, ohne dass der Roboter über eine vollständige Karte seiner Umgebung verfügt. Daraus folgt, dass zuerst die Umgebung des Roboters gescannt werden muss und dann die aufgenommenen Scans als Ausgangspunkt für die Berechnung dienen.

Der erste Schritt des Algorithmus besteht in der Aufnahme eines 3D-Scans. Anschließend wird aus den aufgenommenen Daten ein so genanntes Riss-Polygon generiert. Ein Riss-Polygon repräsentiert einen Schnitt durch die Roboterumgebung (Aufriss) und besteht aus detektierten und hinzugefügten Linien. Jenseits dieser hinzugefügten Linien befindet sich noch nicht explorierte Umgebung. Zur Erzeugung des Riss-Polygons werden alle Messpunkte herausgefiltert, die sich auf einer definierten Höhe befinden. Abbildung 4.4 (oben) verdeutlicht dies; es werden alle Punkte ermittelt, die eine Höhe von 150cm ± 2cm aufweisen.

Die derart bestimmten Punkte dienen als Eingabe für einen Linienerkenner. Da die Punkte nicht sortiert sind – sie stammen aus unterschiedlichen Scanschnitten –, kommt die Hough-Transformation [74, 75] zum Einsatz. Ihr Ergebnis ist in Abbildung 4.4 (unten links) zu sehen. Die detektierten Linien verbindet der Algorithmus nun so, dass ein Polygon entsteht. Zwischen den Sprungkanten (vgl. Kapitel 3.4) werden Linien eingeführt, die einen Teil des Risses der zugrunde liegenden Höhe verdecken; d.h. dass sich ein Teil der Umgebung hinter dieser Linie befindet, aber noch nicht gescannt wurde. Wenn die Linien sortiert vorliegen, ist es einfach, sie richtig zu verbinden. Der kleinste Winkel zwischen den Endpunkten einer Linie und der Scannerposition dient dabei als Sortierkriterium. Abbildung 4.3 verdeutlicht diesen Vorgang.

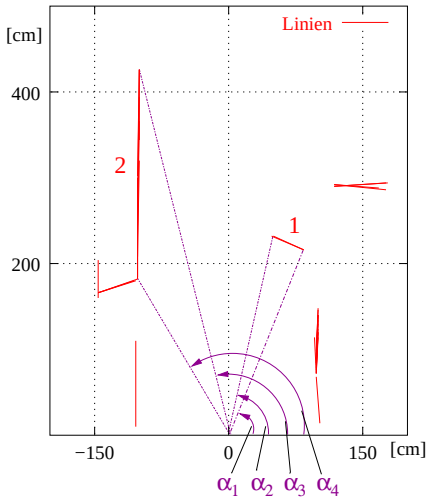


**Abbildung 4.3:** Das Sortieren der Linien zur Erstellung von Riss-Polygonen. Als Sortierkriterium fungiert der kleinste Winkel zwischen den Endpunkten einer Linie und der Scannerposition. Linie 1 wird mit Linie 2 verbunden, da $\alpha_1 < \alpha_3$ ist und kein $\alpha$ einer anderen Linie zwischen diesen Werten liegt.

Die Art und Weise, wie die detektierten Linien verbunden werden, entspricht dem Vorgehen von so genannten Sweepline-Algorithmen. Eine analoge Sichtweise besteht darin, einen Strahl von

der Scannerposition aus von rechts nach links laufen zu lassen (also über alle möglichen Werte von $\alpha$). Falls der Strahl auf eine Linie trifft, wird diese mit der Vorgängerlinie verbunden. Das Riss-Polygon entsteht durch das Verbinden aller detektierten Linien (vgl. Abbildung 4.4 unten rechts).

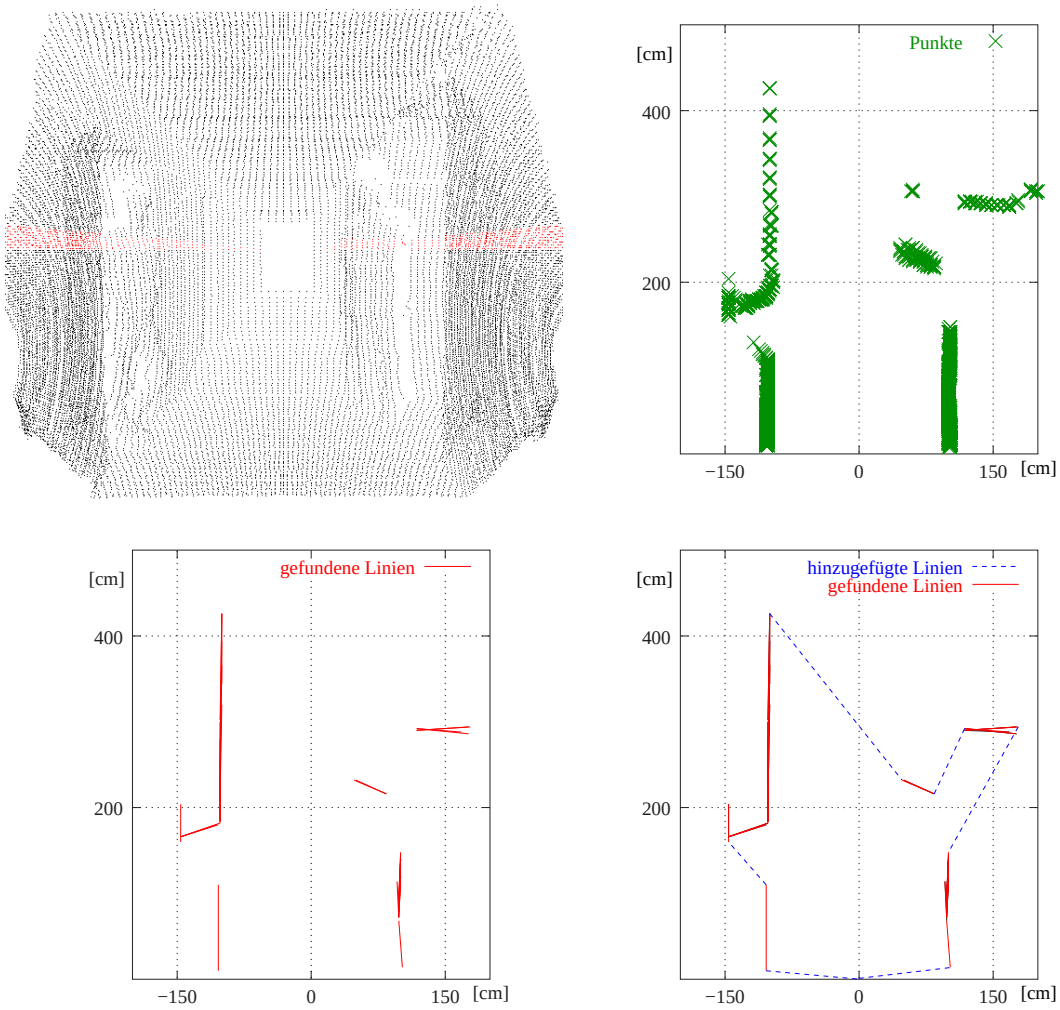


**Abbildung 4.4:** Das Generieren von Riss-Polygonen aus einem 3D-Scan. Oben links: Alle Punkte in Höhe $\sim$ 150cm sind markiert. Oben rechts: $(x, z)$-Koordinaten dieser Punkte. Unten links: Erkannte Linien. Unten rechts: Generiertes Polygon (detektierte Kanten rot, hinzugefügte Linien blau).

Der zweite Schritt des Algorithmus zur Berechnung der optimalen nächsten Scanposition erzeugt Kandidatenpositionen innerhalb des Riss-Polygons (vgl. Abbildung 4.5 links). Alle Positionen innerhalb des Polygons haben die gleiche Wahrscheinlichkeit. Die Aufgabe besteht nun darin, diejenige Kandidatenposition auszuwählen, an der ein weiteres Scannen den maximalen Zuwachs an Information verspricht. Möglichst viele Messpunkte des 3D-Scanners sollen in Richtung verdeckter, also hinzugefügter Kanten liegen.

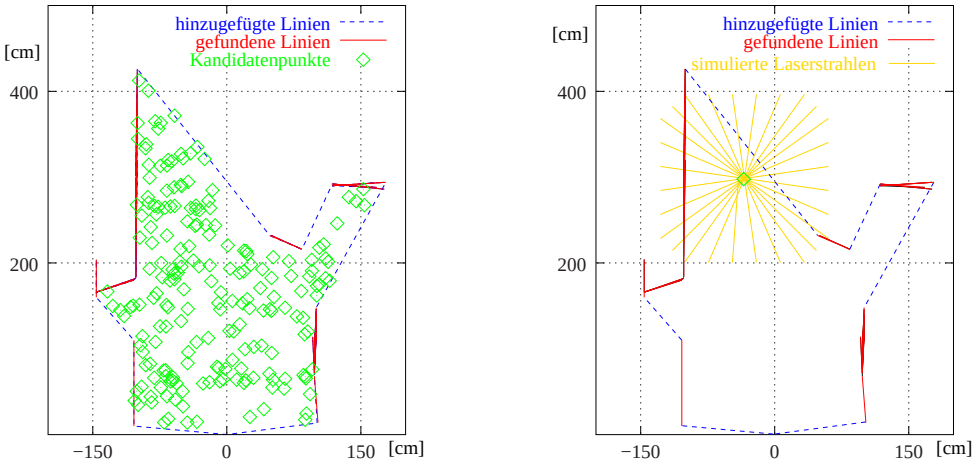


**Abbildung 4.5:** Kandidatenpunkte im Riss-Polygon. Links: Kandidatenpunkte. Rechts: Evaluation durch Scannersimulation.

Der dritte Schritt evaluiert die Kandidatenpunkte (vgl. Abbildung 4.5 rechts). Für jeden dieser Punkte muss errechnet werden, welcher Anteil vom Riss-Polygon sichtbar ist. In der implementierten Fassung geschieht dies über eine Simulation des Laserscanners. Von der Kandidatenposition aus werden virtuelle Laserstrahlen ausgesendet, und die Anzahl der Schnittpunkte wird mit detektierten bzw. mit hinzugefügten Linien bestimmt. Entscheidend für eine Kandidatenposition ist dabei die Anzahl der Schnittpunkte mit den hinzugefügten Linien. Bei der Bestimmung der Schnittpunkte finden die physikalischen Eigenschaften des Sensors Berücksichtigung. Schnittpunkte, die zu nah am Sensor liegen, werden nicht gezählt. Gleiches gilt für zu große Entfernungen vom Laserscanner.

An dieser Stelle bezieht der Planungsalgorithmus die dritte Dimension ein. Der 3D-Laserscanner ist fest in einer Höhe von 105 cm auf dem Roboter montiert. Wenn der Scanner Objekte in einer anderen Höhe erfassen will, muss er einen Mindestabstand einhalten. Die Ursache dafür liegt im Öffnungswinkel des Sensors ($\sim 90°$) (vgl. Abbildung 4.6). Des Weiteren nimmt die Strahldichte mit dem Quadrat der Entfernung ab [54]. Daraus folgt: Je weiter der Scanner vom Objekt entfernt steht, desto weniger Laserstrahlen treffen auf das Objekt. Optimal ist daher eine Scannerposition nahe dem Objekt, aus der es gerade noch erfasst werden kann.

Die Mindestentfernung $m_{\|}$, die der 3D-Scanner vom Objekt bzw. den Kanten der Riss-Ebene haben muss, lässt sich aus der Höhe des Scanners $h_s$ und der Riss-Ebene $h$ errechnen. (vgl. Abbildung 4.6). Falls das zu scannende Objekt unterhalb der Höhe des Scanners liegt, ist

$$m_{\|} = \frac{h_s - h}{\sin\theta_1}.$$

Ansonsten gilt:

$$m_{\|} = \frac{h - h_s}{\sin\theta_2}.$$

Dabei ist $m_{\|}$ der horizontale Abstand zum Objekt in der Richtung, die durch die Orientierung des Roboters gegeben ist.

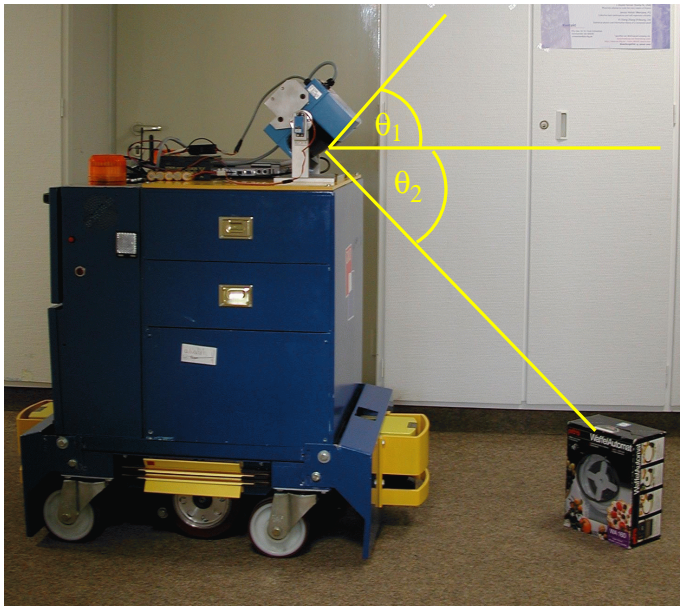


**Abbildung 4.6:** Der Öffungswinkel des AIS 3D-Laserscanner macht einen Mindestabstand beim Erfassen eines Objektes notwendig, wenn dieses sich auf einer von der Montagehöhe des Scanners abweichenden Höhe befindet.

Nachdem alle Kandidatenpunkte unter Berücksichtigung der minimalen Entfernung $m_{\|}$ und einer maximalen Entfernung evaluiert sind, wird die beste Orientierung des Roboters am Kandidatenpunkt bestimmt. Dabei versucht der Roboter in den größten noch nicht gesehenen Bereich zu schauen. Das Ergebnis des Algorithmus für die Riss-Ebene aus Abbildung 4.4 ist in Abbildung 4.7 angegeben.

Die optimale nächste Scanposition besitzt drei Eigenschaften: Erstens ist der bei der Evaluierung jedes NBV-Kandidatenpunktes $q$ errechnete Wert $W(q)$ gleich der Anzahl der Schnittpunkte mit hinzugefügten Linien. Zweitens sollte die Position nicht allzu weit von der aktuellen Position entfernt sein. Und drittens ist ein ständiges Drehen des Roboters zu vermeiden. Die folgende Formel trägt dem Rechnung und definiert dadurch den Begriff „optimale nächste Scanposition“:

$$\hat{W}(q) = W(q)\ \exp(-c_1\ ||r - q||)\ \exp(c_2\ ||\theta_R - \theta_q||). \tag{4.1}$$

Die $(x, y)$-Position des Roboters ist mit $r$ bezeichnet, die des Kandidatenpunktes mit $q$. $\theta_R$ gibt die Orientierung des Roboters, $\theta_q$ die Orientierung des Kandidatenpunktes an. Die Faktoren $c_1$ und $c_2$ gewichten die relativen Kosten der Bewegungen mit dem Informationsgewinn. Die Gewichtsfunktion (4.1) verhindert, dass der Roboter zwischen Regionen mit ähnlichen Gewinnen an visuellen Informationen oszilliert [29].

Wendet man das oben beschriebene Verfahren auf Riss-Ebenen verschiedener Höhe an, lässt sich die optimale nächste Scanposition für den 3D-Laserscanner bestimmen. Es gibt Situationen, in denen Verdeckungen auftreten, die der 3D-Laserscanner nicht erfassen kann. Dies ist durch den Aufbau bedingt: Der Sensor hat eine feste Höhe.

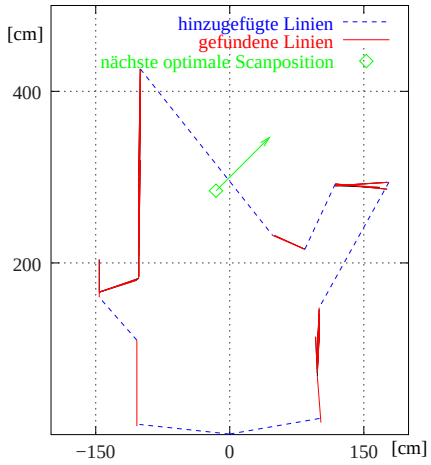


**Abbildung 4.7:** Die nächste optimale Scanposition. Nach der Auswertung aller Kandidatenpunkte im Riss-Polygon der Abbildung 4.5 ergibt sich, dass die markierte Position am besten ist.

**Die optimale nächste Scanposition aus mehreren 3D-Scans.** Liegen mehrere 3D-Scans vor, muss aus diesen ein Riss-Polygon erstellt werden, um die optimale nächste Scanposition zu errechnen. Kapitel 3 beschreibt die Transformationen von 3D-Scans zum Aufbau einer konsistenten Szene. Aus jedem 3D-Scan entsteht ein separates Riss-Polygon. Alle erzeugten Risse befinden sich auf gleicher Höhe. Die Polygone werden mittels Vattis Polygon Clipping Algorithmus [83] zu einem einzigen Polygon vereinigt. Dabei verändert sich die Art der Kanten (detektierte bzw. hinzugefügte Kanten) nicht. Abbildung 4.8 veranschaulicht diesen Prozess.

Der verwendete Polygon Clipping Algorithmus arbeitet nach einem Sweepline Verfahren. Die beiden Polygone tastet man dabei durch horizontale Streifen ab. Die Streifen entstehen durch aufeinander folgende horizontale Strahlen durch alle Eckpunkte des Polygons. Bei diesem Durchlaufen der Polygone wird eine Tabelle aufgebaut, die alle Kantenzüge im aktuellen Streifen enthält. Mit Hilfe der Tabelle werden die im Streifen enthaltenen Ecken und Kanten klassifiziert. Anhand der Klassifizierung berechnet der Clipping Algorithmus die Vereinigung der beiden Polygone [83]. Das so entstandene neue Polygon dient als Basis für den randomisierten Approximationsalgorithmus.

Der alternative Weg zur Riss-Polygonerzeugung, zuerst verschiedene 3D-Scans auf Messdatenebene zu einem Metascan zu vereinigen, um anschließend mit einem Linienerkenner den sichtbaren Teil des Risses zu erzeugen, hat den Nachteil, dass der einfache Sweepline Algorithmus (vgl. Seite 49) nicht funktioniert. Die Schwierigkeit besteht im Hinzufügen der verdeckten Kanten.

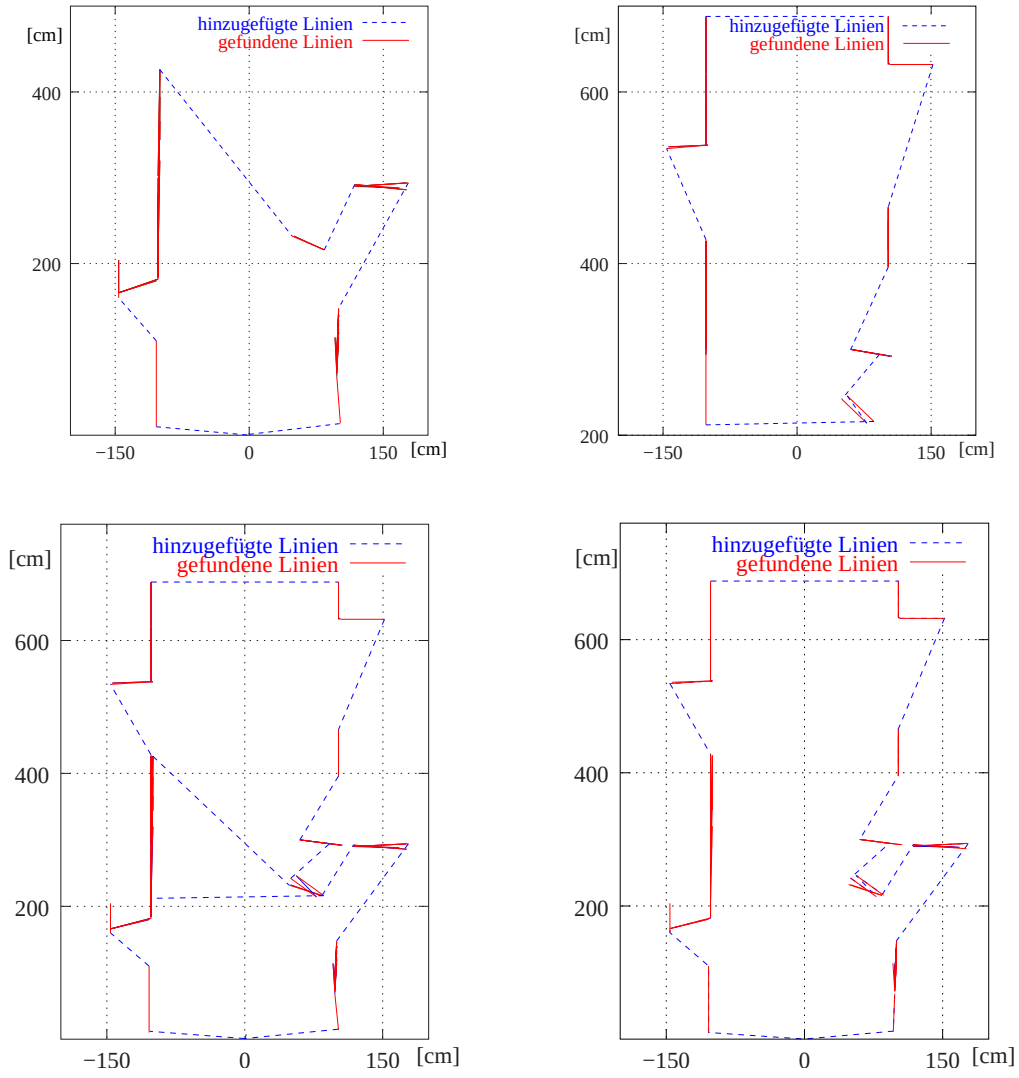


**Abbildung 4.8:** Das Zusammenfügen zweier Riss-Polygone; Oben: Zwei Polygone aus unterschiedlichen 3D-Scans, Unten: Die Vereinigung der beiden Polygone mittels Vattis Polygon Clipping Algorithmus.

Der NBV-Algorithmus lässt sich wie folgt zusammenfassen:

Drei Schritte werden benötigt, um den nächsten optimalen Scanpunkt zu errechnen:

1. Generiere ein Riss-Polygon aus einem 3D-Scan. Liegen mehrere 3D-Scans vor, erzeuge je Scan ein Riss-Polygon. Benutze in diesem Fall Scanmatching nur, um die Polygone richtig auszurichten, und füge die Riss-Polygone mit einem Polygon-Clipping Algorithmus aneinander.
2. Generiere zufällig $n_K$ Kandidatenpositionen innerhalb der als Riss-Polygon $\mathcal{P}$ gegebenen Karte der bekannten Roboterumgebung.
3. Berechne für jede Kandidatenposition den Teil des Riss-Polygons $\mathcal{P}$, der von dieser Position aus zusätzlich sichtbar gemacht werden kann. Derjenige Kandidatenpunkt, der den größten Informationszuwachs verspricht (vgl. Gleichung (4.1)) und sich leicht anfahren lässt, ist der nächste optimale Scanpunkt.

**Performanz des Planungsalgorithmus für die nächste optimale Scanposition.** Der erste Schritt des Algorithmus betrachtet alle Messpunkte in einem 3D-Scan, um diejenigen Punkte zu filtern, die sich auf der definierten Höhe befinden. Nach dem Filtern der Punkte wird die Hough-Transformation eingesetzt, die eine Laufzeit von $\mathcal{O}(c\,n_R)$ hat. Dabei bezeichnet $n_R$ die Anzahl der Punkte in der gewählten Riss-Ebene ($n_R \ll$ Anzahl aller Messdaten in einem 3D-Scan) und $c$ ist abhängig von der größten auftretenden Entfernung vom 3D-Scanner. Die Polygonerzeugung wendet Bubble-Sort auf die Linien an; die Anzahl der Linien ist allerdings wesentlich kleiner als die Anzahl der Punkte in einer Riss-Ebene. Liegt bereits ein Riss-Polygon vor, muss das neu erzeugte mit dem vorhandenen vereinigt werden. Vattis Polygon Clipping Algorithmus hat eine Laufzeit von $\mathcal{O}(n_P)$. $n_P$ ist die Summe der Kanten beider Riss-Polygone [83]. Die Anzahl der Kanten in einem Riss-Polygon ist vor dem Clipping doppelt so groß wie die Anzahl der detektierten Linien im Riss, d.h. $n_P = 2n_L$.

Dieser erste Schritt benötigt in der Ausführung auf einem PIII-600 weniger als eine Sekunde. Auch für den zweiten Schritt, das Erzeugen der Kandidatenpositionen, genügt ein Bruchteil einer Sekunde. Der dritte Schritt ist der zeitaufwendigste. Die Evaluation der Kandidatenpunkte benötigt $\mathcal{O}(n_K\,n_P)$. Dabei ist $n_K$ die Anzahl der Kandidatenpunkte und $n_P$ die der Kanten im Riss-Polygon. Letztere steigt mit der Größe der explorierten Umgebung. Auch ist die Simulation der Laserstrahlen aufwendig, da für jeden Strahl der zum Scanner nächste Schnittpunkt mit den Kanten des Riss-Polygons ermittelt werden muss. Die Durchführung des dritten Schrittes kann für den Flur des FhG-AIS Gebäudes C2 bis zu 2 Sekunden (PIII-600) in Anspruch nehmen.

## 4.4 Das Anfahren der optimalen nächsten Scanposition

Wurde die nächste optimale Scanposition bestimmt, muss sie vom Roboter angefahren werden. Er muss aus einer gegebenen Pose $(x_R, y_R, \theta_R)$ in die neue $(x_{nbv}, y_{nbv}, \theta_{nbv})$ überführt werden. Ein stetiger Pfad (Trajektorie) soll die beiden Positionen verbinden. Dem Roboter stehen für diese Aufgabe Odometrie-Informationen mit hinreichender Genauigkeit zur Verfügung.

Der Roboter ist ein so genanntes nonholonomes Fahrzeug. Um ihn zum Fahren zu bringen, wird ein Controller eingesetzt, der auch bei den GMD-Fussballrobotern Anwendung findet und von G. Indiveri entwickelt wurde [46]. Der Controller bewirkt, dass sich der Roboter seinem Ziel in einer geraden Linie annähert.

**Das kinematische Modell.** Dem Controller liegt ein kinematisches Modell zu Grunde, das in kartesischen Koordinaten $(x_G, y_G, \phi)$ angegeben wird (vgl. Abbildung 4.9):

$$\begin{aligned} \dot{x}_G &= u\cos\phi \\ \dot{y}_G &= u\sin\phi \\ \dot{\phi} &= \omega = \frac{\tan\psi}{l} = uc. \end{aligned} \tag{4.2}$$

Dabei ist $u$ die Geschwindigkeit, $\phi$ der Stellwinkel, $\omega$ die Winkelgeschwindigkeit, $l$ die Länge des Roboters, $\psi \in (-\frac{\pi}{2}, \frac{\pi}{2})$ der Lenkwinkel und $c$ die Krümmung der Trajektorie. Bei dem verwendeten Roboter spielt der Lenkwinkel keine Rolle, da er einen Differentialantrieb, also zwei getrennt ansteuerbare Antriebsräder besitzt und sich daher auf der Stelle drehen kann.

Das Modell (4.2) wird mit den polaren Zustandsvariablen (vgl. Abbildung 4.9)

$$\begin{aligned} e &= \sqrt{{x_G}^2 + {y_G}^2} \\ \theta &= \text{ATAN2}(-y_G, -x_G) \\ \alpha &= \theta - \phi \end{aligned}$$

zu dem Zustandsraum

$$\begin{aligned} \dot{e} &= -u\cos\alpha \\ \dot{\alpha} &= u\left(c - \frac{\sin\alpha}{e}\right) \\ \dot{\theta} &= u\frac{\sin\alpha}{e}. \end{aligned} \tag{4.3}$$

Dieser Zustandsraum ist vollständig definiert, außer für die Menge $\{e, \alpha, \theta | e = 0\ \forall(\alpha, \theta) \in \mathbb{R}^2\}$. Das Problem besteht nun darin, glatte Funktionen für $u$ und $c$ zu finden, die den Zustandsraum für *jede* anfängliche Konfiguration $(e, \alpha, \theta)$ in das Ziel $(0, 0, 0)$ überführen [46].

Bevor mit der Lösung dieses Kontrollproblems begonnen werden kann, müssen die Pose des Roboters $(x_R, y_R, \theta_R)$ und die der optimalen nächsten Scanposition $(x_{nbv}, y_{nbv}, \theta_{nbv})$, die im globalen Koordinatensystem bestimmt sind, in das Modell mit dem Koordinatensystem $(x_G, y_G, \phi)$

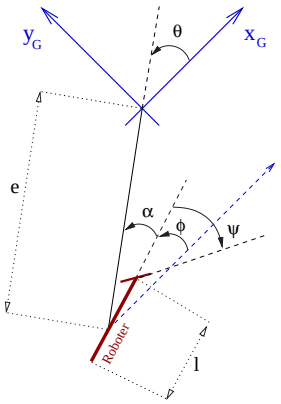


**Abbildung 4.9:** Das kinematische Modell: Das lokale Koordinatensystem $(x_G, y_G)$ des verwendeten Controllers.

überführt werden (vgl. Abbildung 4.10). Die Gleichungen

$$\phi = \theta_R - \theta_{nbv} \tag{4.4}$$

$$x_G = (x_{nbv} - x_R)\cos\left(\theta_{nbv} + \frac{\pi}{2}\right) + (y_{nbv} - y_R)\sin\left(\theta_{nbv} + \frac{\pi}{2}\right) \tag{4.5}$$

$$y_G = (y_{nbv} - y_R)\cos\left(\theta_{nbv} + \frac{\pi}{2}\right) - (x_R - x_{nbv})\sin\left(\theta_{nbv} + \frac{\pi}{2}\right) \tag{4.6}$$

bewirken die gewünschte Koordinatentransformation.

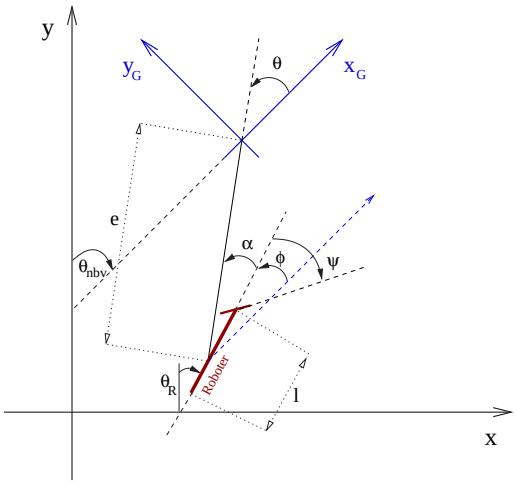


**Abbildung 4.10:** Einbettung der lokalen Motorcontroller-Koordinaten in das globale Koordinatensystem des Roboters.

G. Indiveri wählt folgenden Ansatz für die Lösung des Kontrollproblems [46]: Da die Ableitungen der Zustände in (4.3) für $u = 0$ gleich Null sind und weil der Roboter nur vorwärts fahren soll, bietet sich die Gleichung

$$u = \gamma e \qquad \text{mit} \quad \gamma > 0 \tag{4.7}$$

als Lösung an. Mit diesem Ansatz und unter Verwendung der quadratischen Lyapunov Funktion ergibt sich eine Lösung für $c$:

$$c = \frac{\sin\alpha}{e} + h\frac{\theta}{e}\frac{\sin\alpha}{\alpha} + \beta\frac{\alpha}{e} \qquad \text{mit} \quad h > 1,\ 2 < \beta < h + 1. \tag{4.8}$$

Die beiden Formeln (4.7) und (4.8) beschreiben den glatten Übergang des Zustandsraumes. Die genaue Herleitung der Formeln befindet sich in Anhang A.5. Bei der Implementierung muss dafür gesorgt werden, dass sich $\theta$ und $\alpha$ im Bereich von $[-2\pi, 2\pi]$ befinden, damit sich für $c$ eine stetige Lösung ergibt. Die wesentliche Funktion des Controllers befindet sich in Anhang B.3. Abbildung 4.11 zeigt einige Bahnkurven des Controllers. Es ist ersichtlich, dass jede gegebene Pose $(x_1, y_1, \theta_1)$ in eine andere $(x_2, y_2, \theta_2)$ überführt werden kann.

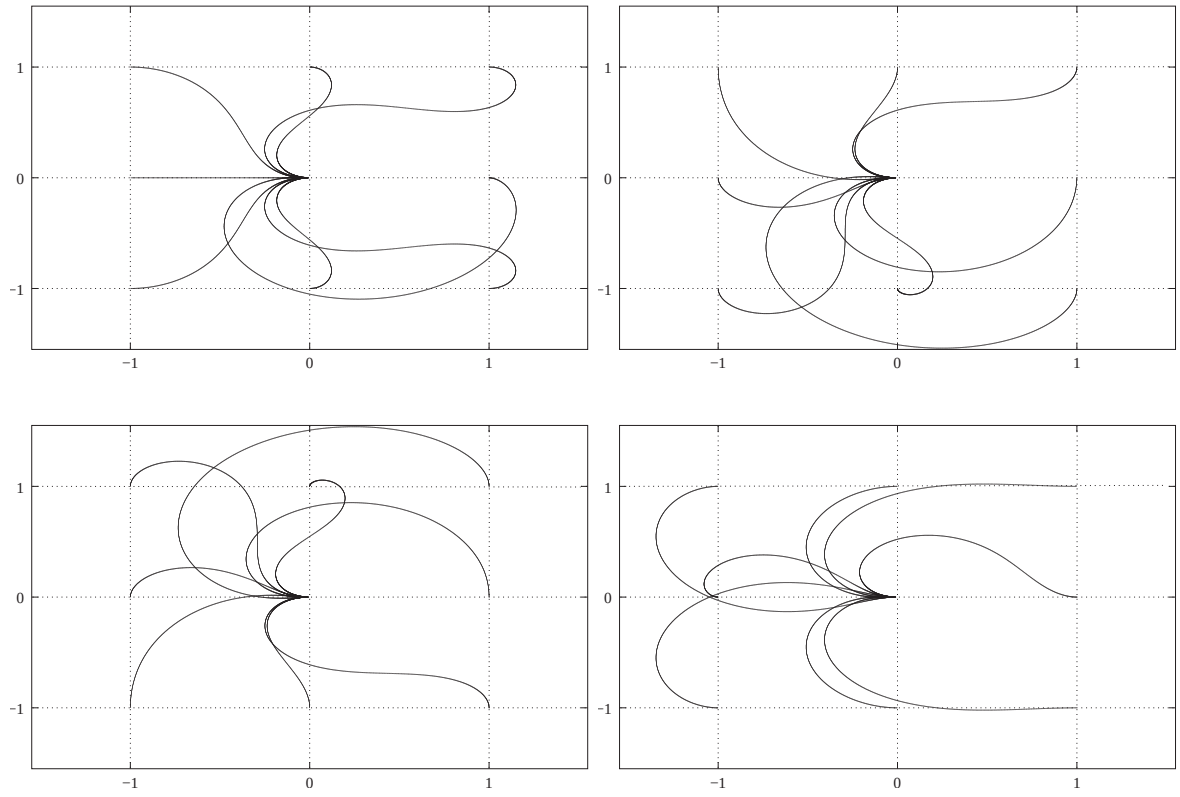

**Abbildung 4.11:** Bahnkurven. Aus jeder gegebenen Pose $(x, y, \theta)$ wird die Zielposition $(0, 0, 0)$ erreicht. Benutzte Konstanten: $\gamma = 1$, $h = 2$, $\beta = 2.9$.

**Dynamik des Systems.** Der letzte Abschnitt beschreibt die Kinematik des Robotersystems. Damit der Roboter wirklich Fahren kann, ist seine Dynamik zu berücksichtigen. Aus der berechneten Geschwindigkeit $u$ und der zu fahrenden Krümmung $c$ werden die Motorwerte ermittelt. Zuerst bestimmt man dazu mittels Formel (4.2) die Winkelgeschwindigkeit $\omega$, um daraus anschließend Werte für den rechten und linken Motor zu generieren. Diese sind eventuell zu skalieren und ein Integrator berücksichtigt die Trägheit des Roboters. Abbildung 4.12 zeigt den zeitlichen Verlauf der Werte $\phi$ (Stellwinkel), $u$ (Geschwindigkeit), $c$ (Krümmung), $\omega$ (Winkelgeschwindigkeit) und die daraus resultierenden Werte für die Motoren an zwei Beispielen. Anhang B.3 zeigt die Implementierung, die zu diesen Kurven führt.

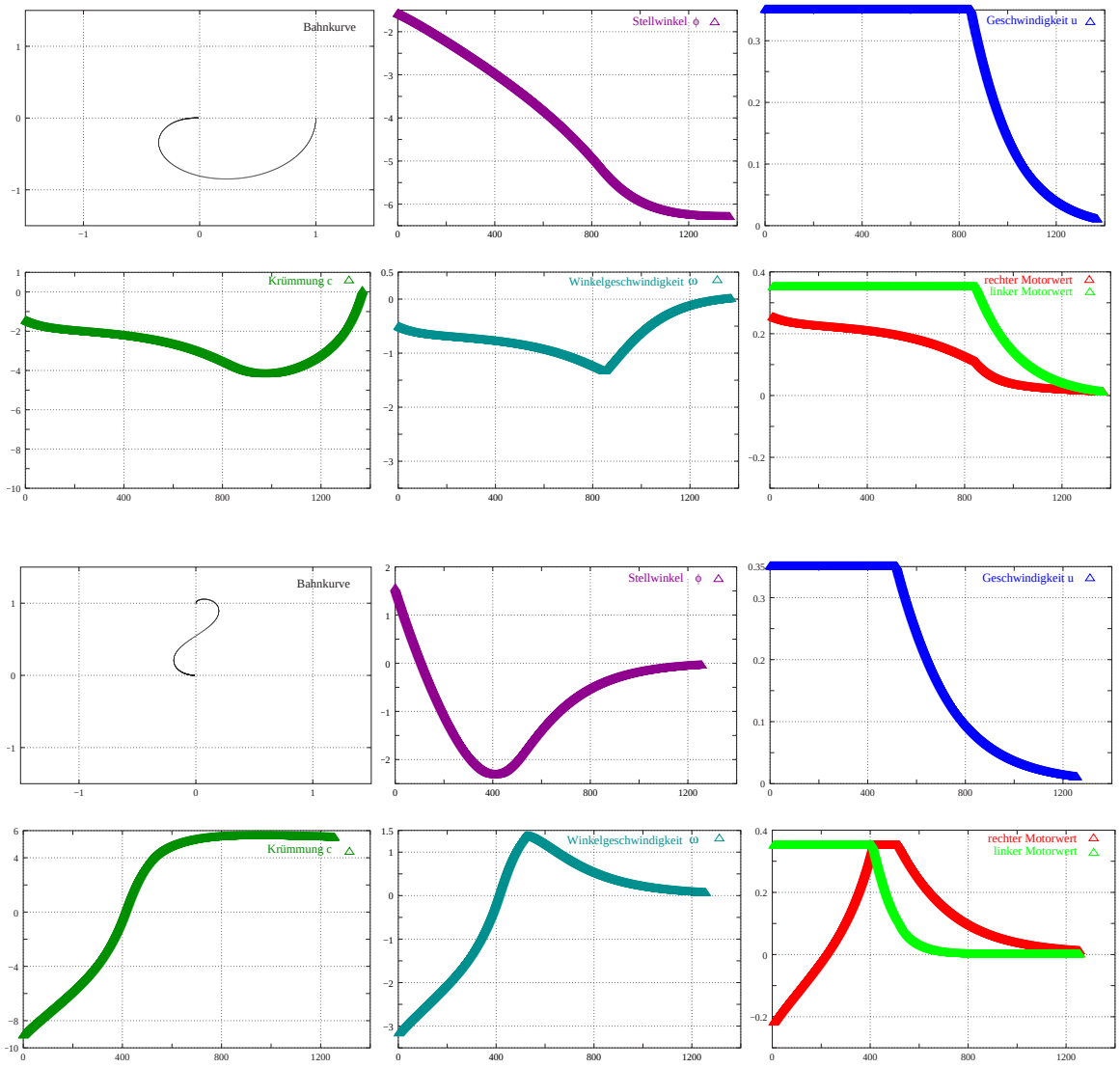


**Abbildung 4.12:** Zeitlicher Verlauf der Werte $\phi$ (Ausrichtung zum Ziel), $u$ (Geschwindigkeit), $c$ (Krümmung), $\omega$ (Winkelgeschwindigkeit) und der Motorwerte für die Bahnkurven mit den Startwerten $(x, y, \theta) = (1.0, 0.0, -\frac{\pi}{2})$ und $(x, y, \theta) = (0.0, 1.0, 0.0)$.

**Kollisionsvermeidung.** Der Controller kann nun dazu verwendet werden, den in Abschnitt 4.3 berechneten nächsten Scanpunkt anzufahren. Nach der Durchführung der entsprechenden Koordinatentransformation (vgl. Formel (4.4), (4.5) und (4.6)) ergibt sich eine Trajektorie wie Abbildung 4.13 für die in Abschnitt 4.3 bestimmte nächste optimale Scanposition zeigt. Die Software überprüft, ob die Trajektorie die Bounding Boxen der Objekte der Szene schneidet (vgl. Kapitel 2.4.2). Wenn dies so ist, wird der Roboter während der Fahrt zur Scanposition mit einem Objekt/Hindernis kollidieren. Daraus folgt, dass die betreffende Scanposition für den Roboter mit dem Controller nicht angefahren werden kann. In einem solchen Fall wird der zweitbeste bzw. der folgende Kanditatenpunkt anhand seiner Trajektorie auf Erreichbarkeit überprüft und eventuell angefahren.

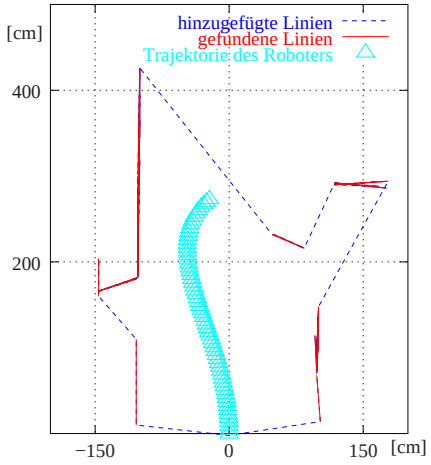


**Abbildung 4.13:** Trajektorie zum Erreichen der nächsten Scanposition.

Zusätzlich zu dem bisher beschriebenen direkten Anfahren der optimalen nächsten Scanposition sind zwei weitere Variationen implementiert. Für die Überführung der Roboterpose $(x_R, y_R, \theta_R)$ in eine neue Pose $(x_{nbv}, y_{nbv}, \theta_{nbv})$ kann es sinnvoller sein, zuerst in Richtung Zielposition zu drehen, um anschließend mit dem Controller das Ziel anzufahren oder die gewünschte Pose in drei Schritten zu erreichen: Als erstes wird in Richtung Ziel gedreht, dieses danach in einer geraden Linie angefahren, um schließlich dort den gewünschten Winkel einzustellen. Diese beiden Variationen basieren ebenfalls auf dem Controller nach [46]. Sie haben gegenüber dem direkten Einsatz des Controllers den Vorteil, dass der gefahrene Roboterweg kürzer ist.

# Kapitel 5

# Ergebnisse

Dieses Kapitel stellt das Gesamtsystem zur autonomen Exploration und Modellierung der 3D-Umgebung eines mobilen Roboters vor. Die Ergebnisse der vorangegangenen Kapitel werden zusammengestellt. Zunächst geht es jedoch um die Ausgabe bzw. Visualisierung der Messdaten.

## 5.1 Visualisierung der Messdaten

Die bisherigen Kapitel beschreiben, wie ein digitales 3D-Modell im Computer aufgebaut wird. Zum einen verwendet der Roboter dieses Modell, zum anderen dienen Funktionen für die Datenausgabe dazu, die gewonnenen Informationen für den Anwender nutzbar zu machen. Eine wichtige Form der Datenausgabe ist die Darstellung auf einem Bildschirm. Dazu müssen die 3D-Messdaten in geeigneter Weise auf die Bildebene projiziert werden. Die Projizierung geschieht mit einem separaten Rendering-Programm, das auf der offenen Grafikbibliothek OpenGL basiert (vgl. Abschnitt 2.4.2). Eine Vorgängerversion dieses Anzeigeprogramms wird detailliert in [75] beschrieben. Mit Hilfe verschiedener Verfahren (Shadings oder Ray-Tracing) werden dabei im Computer generierte Projektionen von Objekten durch Nachbearbeitung (Licht, Schatten, Farbverläufe) in realistisch wirkende dreidimensionale Darstellungen umgewandelt. Dadurch können neben den aufgenommenen Messpunkten, die in Abschnitt 2.4.2 vorgestellten Linien, Flächen, Polygone und Objekte angezeigt werden (vgl. [52, 74, 75, 76]). Alle in den bisherigen Kapiteln verwendeten 3D-Grafiken wurden mit diesem Programm erzeugt.

In der Computergrafik, einem Teilgebiet der Informatik, werden die meisten darzustellenden Objekte durch Gitter (engl.: *mesh*) repräsentiert, bevor sie angezeigt werden [9]. Die Notwendigkeit einer Darstellung der Messpunkte durch ein Gitter wird unter anderem aus Abbildung 5.1 ersichtlich. Werden sehr viele Messpunkte gezeichnet, so überlagern sich die Punkte und der realistische Eindruck geht verloren. Dies lässt sich mit einem Gittermodell vermeiden.

Für die Aufgabe, ein Gittermodell zu erzeugen, existieren verschiedene Ansätze. Der folgende Abschnitt erläutert die wichtigsten Methoden zur Erstellung eines Gitters aus 3D-Punkten [5, 9, 17, 39, 81], bevor in Abschnitt 5.1.2 die auf Octalbäumen basierende Methode von Pulli et al. vorgestellt wird [59].

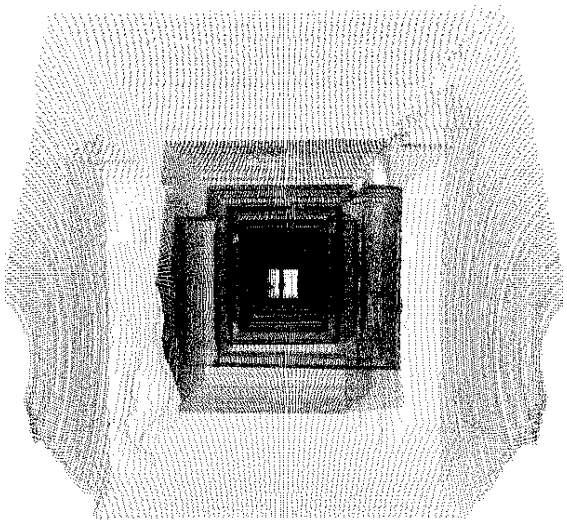

**Abbildung 5.1:** 650000 Messpunkte aufgenommen im Flur des FhG-AIS Gebäudes C2. Bei der Darstellung sehr vieler Messpunkte, geht der realistische Eindruck verloren.

### 5.1.1 Oberflächenrepräsentation durch Gittermodelle

#### Polygongitter von Tiefenbildern und das Reißverschlussverfahren

Turk und Levoy [81] nehmen verschiedene Tiefenbilder von Objekten mit Hilfe eines auf Lichtstreifen basierenden Triangulationsscanners auf. Die erste Stufe des von ihnen entwickelten Algorithmus wandelt jedes Tiefenbild in ein Gitter um. Dabei sind alle 3D-Messpunkte potentielle Knoten in diesem Gitter, das nur aus Dreiecken besteht. Aus jeweils vier benachbarten Punkten (Spalte und Zeile) des Tiefenbildes entstehen kein, ein oder zwei Dreiecke. Turk und Levoy suchen nach der kürzesten Diagonale zwischen diesen vier Punkten, um unter Anwendung eines Schwellenwertes die Punkte zu identifizieren, die die Dreiecke bilden [81].

Der zweite Schritt des Algorithmus verbindet jeweils 2 Gitter, die zuvor in einem gemeinsamen Koordinatensystem registriert wurden. Dabei kommt das so genannte Reißverschlussverfahren (engl.: *mesh zippering*) zum Einsatz. 3 Teilschritte sind dazu nötig: Redundante Bereiche der Gitter werden im ersten Schritt entfernt, so dass sich die Gitter gerade noch überlappen. Als zweites werden die restlichen überlappenden Flächen abgeschnitten und die Dreiecke im Reißverschlussverfahren zusammengefügt. Zum Abschluss entfernt man alle kleinen Dreiecke [81], um das Gitter an den Verschlussstellen zu glätten.

Dieses Verfahren wird in vielen 3D-Scansystemen eingesetzt. Einige Beispiele dafür finden sich in [53, 64, 69, 82]. Die Gitterrekonstruktion aus einem einzigen 3D-Scan findet auch beim AIS 3D-Laserscanner Anwendung [75]. Der Unterschied zu [81] besteht darin, dass die Vereinigung mehrerer 3D-Scans nicht implementiert ist und das Gitter aus Vierecken besteht.

#### Oberflächenrekonstruktion aus ungeordneten Punktmengen

Hoppe et al. [39] verwenden ungeordnete 3D-Punktmengen für die Gittererzeugung. Damit unterscheiden sie sich wesentlich von Turk und Levoy [81], die gerade die Struktur in den Tiefenbildern

ausnutzen. Der Oberflächenrekonstruktionsalgorithmus arbeitet in zwei Schritten. Zuerst definieren Hoppe et al. eine Funktion $f : D \rightarrow \mathbb{R}$, wobei $D \subset \mathbb{R}^3$ die Daten sind, bzw. eine Region in der Nähe der Daten, so dass $f$ die vorzeichenbehaftete Distanz zur unbekannten Oberfläche $M$ schätzt. Der Schlüssel zur Definition der Distanzfunktion $f$ besteht darin, orientierte Ebenen mit jedem 3D-Punkt zu assoziieren. Diese sind Tangentialflächen und dienen als lokale lineare Approximationen der Oberfläche [39].

Da $Z(f)$, das Urbild von $f$, eine Schätzung für $M$ ist, benutzt der zweite Schritt einen so genannten Contour-Algorithmus (Marching Cubes) für die Approximation von $Z(f)$ durch eine einfache Fläche. Der Marching-Cube-Algorithmus legt ein feines 3D-Raster durch das Objekt und evaluiert die Distanzfunktion $f$ an den Rasterpunkten, um die Dreiecke des Gitters aus den Quadern des Rasters zu erzeugen [39].

#### Volumenbasierte Rekonstruktion von Oberflächen

Curless und Levoy geben einen sehr effizienten Algorithmus zur Rekonstruktion von Oberflächen aus Laserscandaten an [17]. Dabei vereinigen sie den Ansatz von Turk und Levoy mit dem von Hoppe et al. Sie erzeugen aus jedem einzelnen 3D-Scan eine vorzeichenbehaftete Distanzfunktion auf einem Volumenraster und nutzen dabei die Struktur in den Daten aus. Anschließend führen sie die Volumenraster und die Distanzfunktionen aus mehreren 3D-Scans zusammen. Um schließlich die Oberfläche zu erzeugen, wird ein Marching-Cube-Algorithmus verwendet. Der Algorithmus von Curless und Levoy ist für sehr große Datenmengen entworfen und wird z.B. im Digital Michelangelo Projekt eingesetzt [50].

#### Die Power Crust Methode

Ein sehr neues Verfahren ist die so genannte Power Crust Methode, die neben einer Gitterdarstellung für Punktwolken eine Approximation der Duchschnittsachse (engl.: *medial axis*) liefert [5, 15]. Die Messpunkte können in beliebiger Reihenfolge vorliegen. Zuerst berechnet der Algorithmus die so genannte Durchschnittsachsentransformation (engl.: *medial axis transform*), um danach mit einer inversen Transformation zu einer Oberflächenrepräsentation zu kommen [5].

Der erste Schritt approximiert die Durchschnittsachsentransformation mit Hilfe der Messpunkte. Diese Transformation repräsentiert das gescannte Objekt durch möglichst große Kugeln, die vollständig im Objekt liegen. Die Approximation gelingt durch Berechnung der Voronoi-Knoten aus den Messpunkten. Diese Voronoi-Knoten werden auch Pole genannt. Kugeln um sie herum berühren jene Messpunkte, die den Polen an nächsten gelegen sind. Nach der Berechnung der Pole klassifiziert sie der Algorithmus anhand ihrer Lage. Dabei gibt es Pole im Objekt und außerhalb von ihm. Die Grenze zwischen den Kugeln, die im Objekt gelegen sind, und jenen außerhalb von ihm ergibt eine Oberflächenrepräsentation, die Power Crust [5].

Die Performanz des Algorithmus hängt im Wesentlichen von der Berechnung des Voronoi-Diagramms ab, die für sehr große Datenmengen nicht durchgeführt werden kann. Auf einem PIII-600 mit 512 Megabyte Hauptspeicher können 75000 Punkte in 2 Minuten verarbeitet werden. Für größere Punktwolken, wie vom AIS Laserscanner erzeugt, lässt sich die Methode nicht einsetzen. Das Power Crust Programm [15] erzeugt ein Gitter um Messpunkte eines 3D-Scans herum; erst eine Perspektive innerhalb der Szene zeigt die entstandene Triangulation (vgl. Abbildung 5.2), wobei sich das Gitter auch über nicht abgetastete Bereiche der Szene erstreckt.

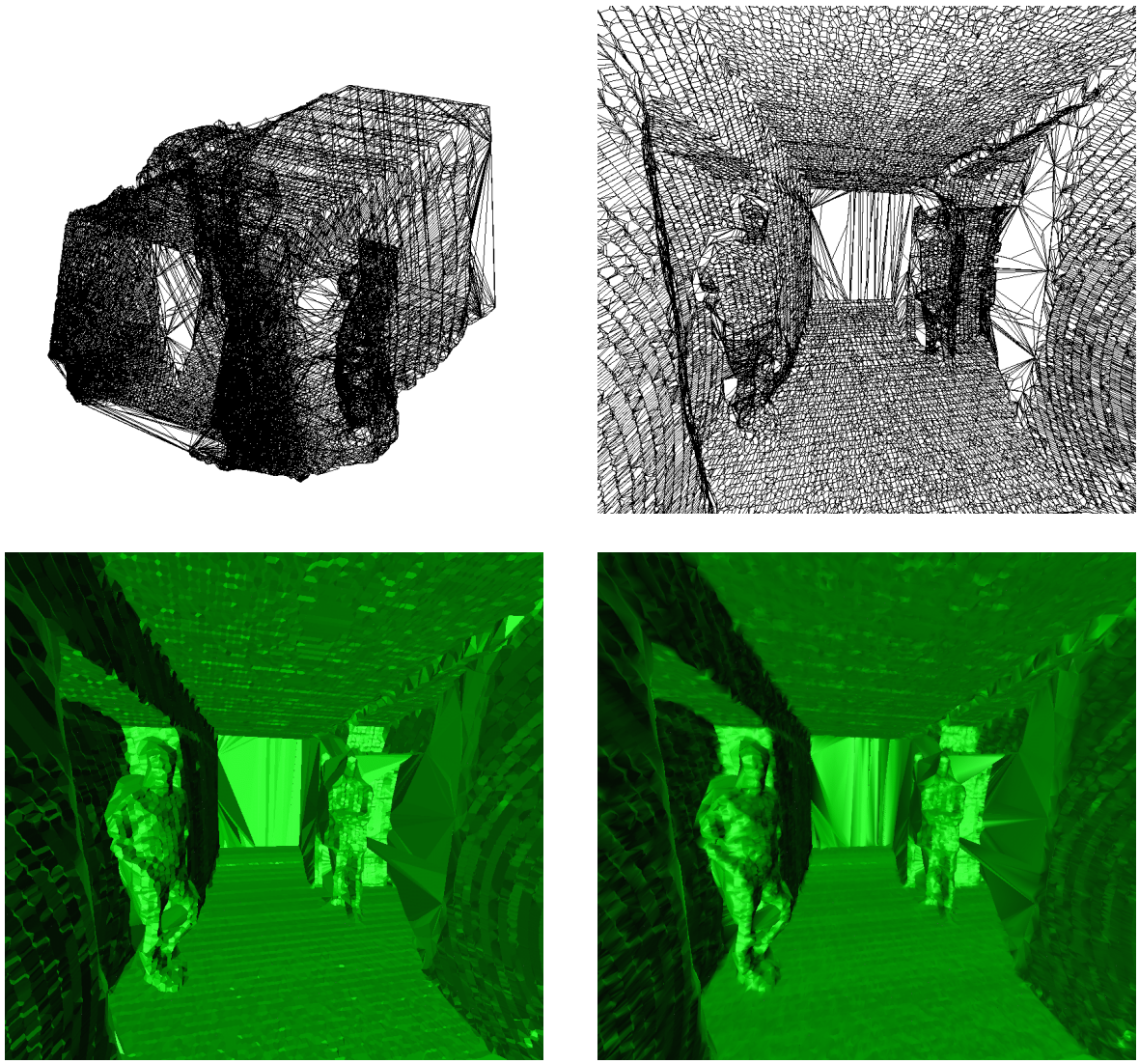

**Abbildung 5.2:** Triangulierung mit Power Crust. Oben links: Triangulierte Szene. Oben rechts: Triangulation aus der Perspektive des Laserscanners. Unten links: Die entstandenen Dreiecke sind in verschiedenen Schattierungen dargestellt. Unten rechts: Die Schattierungen sind zusätzlich mit glattem Übergang untereinander dargestellt. In dieser Abbildung beruhen Schattierungen *nicht* auf den Intensitätswerten.

### 5.1.2 Erzeugung eines einfachen Gittermodells durch Octalbäume

Wie der Name schon sagt, handelt es sich bei der Datenstruktur Octalbaum um einen Baum, also einen kreisfreien Graphen[45]. Jeder innere Knoten des Baumes besitzt acht Nachfolgerknoten. Die Blätter haben keine Nachfolger, enthalten aber die im Baum gespeicherten Daten. Die Idee besteht nun darin, die Messdaten in den Blättern eines Octalbaumes zu speichern.

Alle Datenpunkte können von einem Quader umschlossen werden, der die Wurzel des Octalbaumes darstellt und entlang der Achsen des Koordinatensystems ausgerichtet ist. Durch das Einfügen von 3 Ebenen (horizontal, vertikal und in der Flucht) wird der Quader in kleinere, gleichgroße Quader zerteilt. Danach werden die Daten auf die kleineren Quader verteilt. Für diese ermittelt man, ob sie die Oberfläche eines Objekte enthalten. Abbildung 5.3 zeigt die drei dabei zu unterscheidenden Fälle: Ein Quader kann vor, hinter oder auf der Oberfläche liegen. Leere Quader gehören sicherlich nicht zur Oberfläche des Objektes. An diesen Stellen schneidet man den Baum ab und bestimmt keine Nachfolgerknoten. Die Quader, die zum Objekt gehören, werden wiederum durch das Einfügen von 3 Ebenen in acht kleinere zerlegt. Diese Rekursion erzeugt an den Blättern Quader, die die Oberfläche der Objekte, auf denen die Messdaten liegen, approximieren. Abbildung 5.4 zeigt den beschriebenen Prozess. Das Verfahren funktioniert unabhängig von den einzelnen 3D-Scans und der Struktur der Daten darin, da es ausschließlich Messpunkte verwendet.

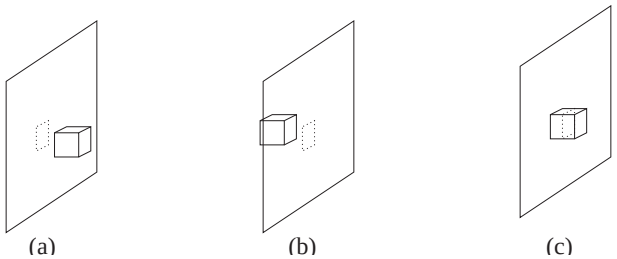


**Abbildung 5.3:** Drei Fälle bei der Erzeugung des Octalbaumes. (a) Der Quader liegt vor der Oberfläche des gescannten Objekts. (b) Der Quader liegt vollständig hinter der Oberfläche. (c) Der Quader schneidet die Oberfläche. Abbildung aus [59].

**Performanz des Octalbaumes.** Der Octalbaum lässt sich wie beschrieben in $\mathcal{O}(n)$ aufbauen. In jeder Rekursionsstufe müssen alle Punkte betrachtet werden. Da aber nur wenige Iterationen benötigt werden (vgl. Abbildung 5.4), sind die Konstanten im Algorithmus klein und das Verfahren läuft im Bruchteil einer Sekunde (PIII-600) ab.

Die Anzahl der Quader steigt nur langsam. Daher ist der Speicherverbrauch des Octalbaumes hauptsächlich durch die Anzahl der im Baum gespeicherten Daten bestimmt, also $\mathcal{O}(n)$. Sie ist gleichzeitig die Obergrenze für die Anzahl der Blätter im Baum. Abbildung 5.5 zeigt die Zahl der Quader im Octalbaum verglichen mit der maximal möglichen, also jener eines vollständigen Octalbaums.

**Darstellung des Octalbaumes als Drahtgitter durch** OpenGL**.** Der erzeugte Octalbaum muss so angezeigt werden, dass ein Gittermodell entsteht. Alle verdeckten Kanten eines Qua-

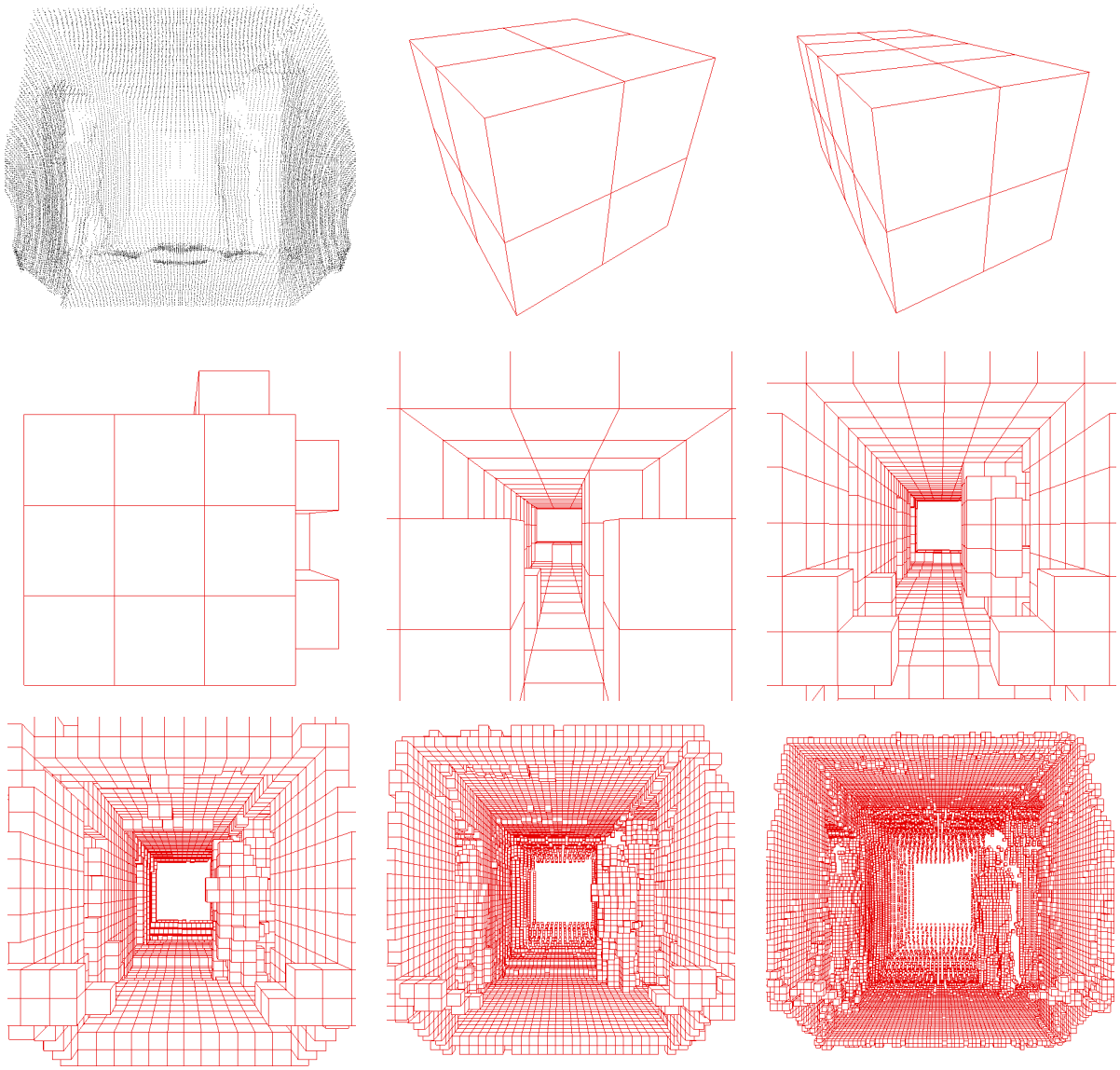

**Abbildung 5.4:** Aufbau eines Octalbaumes aus den Messdaten. Links oben: Messdaten. Der Octalbaum entsteht durch die Aufteilung eines Quaders in 8 Nachfolger und durch das Abschneiden leerer Quader. Die Baumtiefen 1 bis 8 sind dargestellt, wobei für Tiefe 1 und 2 eine andere Perspektive gewählt wurde.

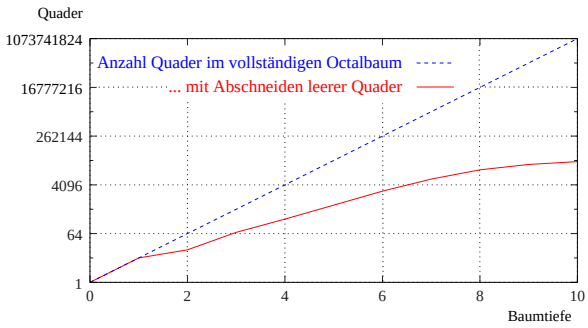


**Abbildung 5.5:** Anzahl der Blätter im Octalbaum (logarithmisch aufgetragen).

ders dürfen nicht gezeichnet werden. Diese Aufgabe ist äquivalent dazu, dass man Flächen mit hervorgehobenen Kanten rendert. Nach dem Zeichnen einer Quaderfläche wird die Umrandung nochmals mit einer anderen Farbe nachgezogen. In Abbildung 5.4 sind die Quader in der Hintergrundfarbe und ihre Umrandung rot dargestellt. OpenGL sorgt mittels der Funktion `void glPolygonOffset(GLfloat factor, GLfloat units)` dafür, dass wirklich nur die Umrandungen gerendert werden.

Neben dem Zeichnen des Octalbaumes als Gittermodell lassen sich auch photorealistische Darstellungen der Szene erzeugen. Dazu rendert das Anzeigeprogramm die Quader, indem es ihnen je nach der Intensität ihrer Messpunkte unterschiedliche Grauwerte zuweist (vgl. Abschnitt 2.4.2, Seite 12). Abbildung 5.6 zeigt den mit den Intensitätswerten eingefärbten Octalbaum.

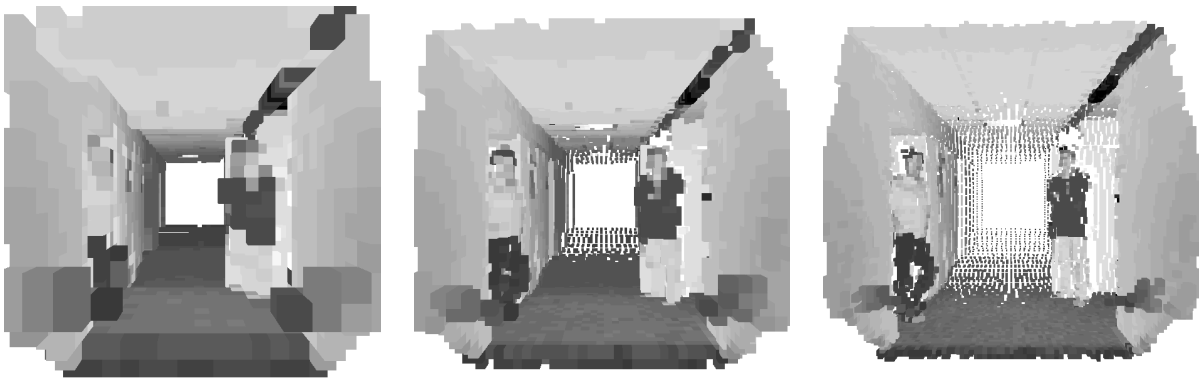

**Abbildung 5.6:** Octalbaum mit Intensitätswerten. Baumtiefe 6 (links) bis 8 (rechts) sind abgebildet. Siehe auch Abbildung 2.8.

## 5.2 3D-Modellierung

Fünf Schritte sind notwendig, um ein vollständiges digitales 3D-Modell aufzubauen:

I. **Datenaufnahme.** Während der Akquisition des Tiefenbildes wird die Oberfläche der Szene bzw. eines Teiles der Szene abgetastet.

II. **Registrierung.** Anfänglich liegt jedes Tiefenbild in seinem eigenen Koordinatensystem vor. Mehrere 3D-Scans werden mit Hilfe einer Schätzung der Roboterpose und durch Scanmatching in einem globalen Koordinatensystem vereinigt.

III. **Planung.** Die nächste optimale Scanposition muss berechnet werden. Anschließend wird ein kollisionsfreier Weg zu dieser bestimmt.

IV. **Roboterbewegung.** Der Roboter muss letztlich zu der vom Planungsmodul bestimmten Position gebracht werden. Er stellt die Motoren ein und schätzt dabei seine Position über die gefahrenen Radumdrehungen (Odometrie). Der Roboter soll während der Fahrt dynamischen Hindernissen, zum Beispiel vorbeilaufenden Menschen, ausweichen.

V. **Integration.** Der letzte Schritt wird im Gegensatz zu den vorigen nicht iterativ ausgeführt, sondern erst nachdem alle Scans aufgenommen wurden. Aus den separaten, aber bereits registrierten 3D-Scans, wird ein Drahtgitter als Oberflächenbeschreibung erstellt. Ein Octalbaum dient zur Erzeugung des Gitters.

Wie bereits in Punkt IV angedeutet, verwendet die Roboterkontrolle während der Bewegung zusätzlich zu der im Planungsalgorithmus stattfindenden Kollisionsvermeidung, die auf den Bounding-Boxen der Objekte basiert, eine dynamische Kollisionsvermeidung. Dazu benutzt der Roboter die 2D-Laserscanner, die als Sicherheitssensoren in Bodennähe angebracht sind (siehe Abbildung 1.1 Seite 2). Unterschreitet ein Messwert dieses Sicherheitssensors einen Schwellenwert, so befindet sich der Roboter in der Nähe eines Hindernisses. Es wird ein vom Hindernis weg zeigender Vektor errechnet und mit den Werten des Motorcontrollers zusammengeführt.

### Experimente in der Einhangshalle (Robobench)

Das im Folgenden beschriebene Experiment findet in der Eingangshalle des FhG-AIS Gebäudes statt. Dort soll der Roboter autonom fahren und nach Beendigung der Fahrt ein vollständiges 3D-Modell seiner Umgebung erzeugen. Die Abbildungen 5.7, 5.8 und 5.9 zeigen die ersten neun 3D-Scans, wobei links die Intensitätswerte dargestellt sind. Die Szene ist in diesem Bild verzerrt, da die Drehung des Scanners vernachlässigt wurde. In der Mitte ist der mit Grautönen eingefärbte Octalbaum zu sehen. Im rechten Teil der Abbildungen befindet sich ein Polygon, das die Riss-Ebene in einer Höhe von 1.35 Metern darstellt. Rote Kanten wurden gescannt, hinter den blauen liegt nicht erfasstes Terrain. Außerdem ist die Trajektorie zur nächsten optimalen Scanposition eingezeichnet.

Die 3D-Scans werden online zu einem vollständigen Modell zusammengefügt. Das Ergebnis zeigt Abbildung 5.10, wobei oben der eingefärbte Octalbaum dargestellt ist. Zusätzlich sind in der Mitte die erkannten Flächen (horizontale grün und vertikale blau bzw. rot, je nach Lage im globalen Koordinatensystem). Die daraus resultierenden Bounding-Boxen der Objekte sind in Abbildung 5.11 visualisiert.

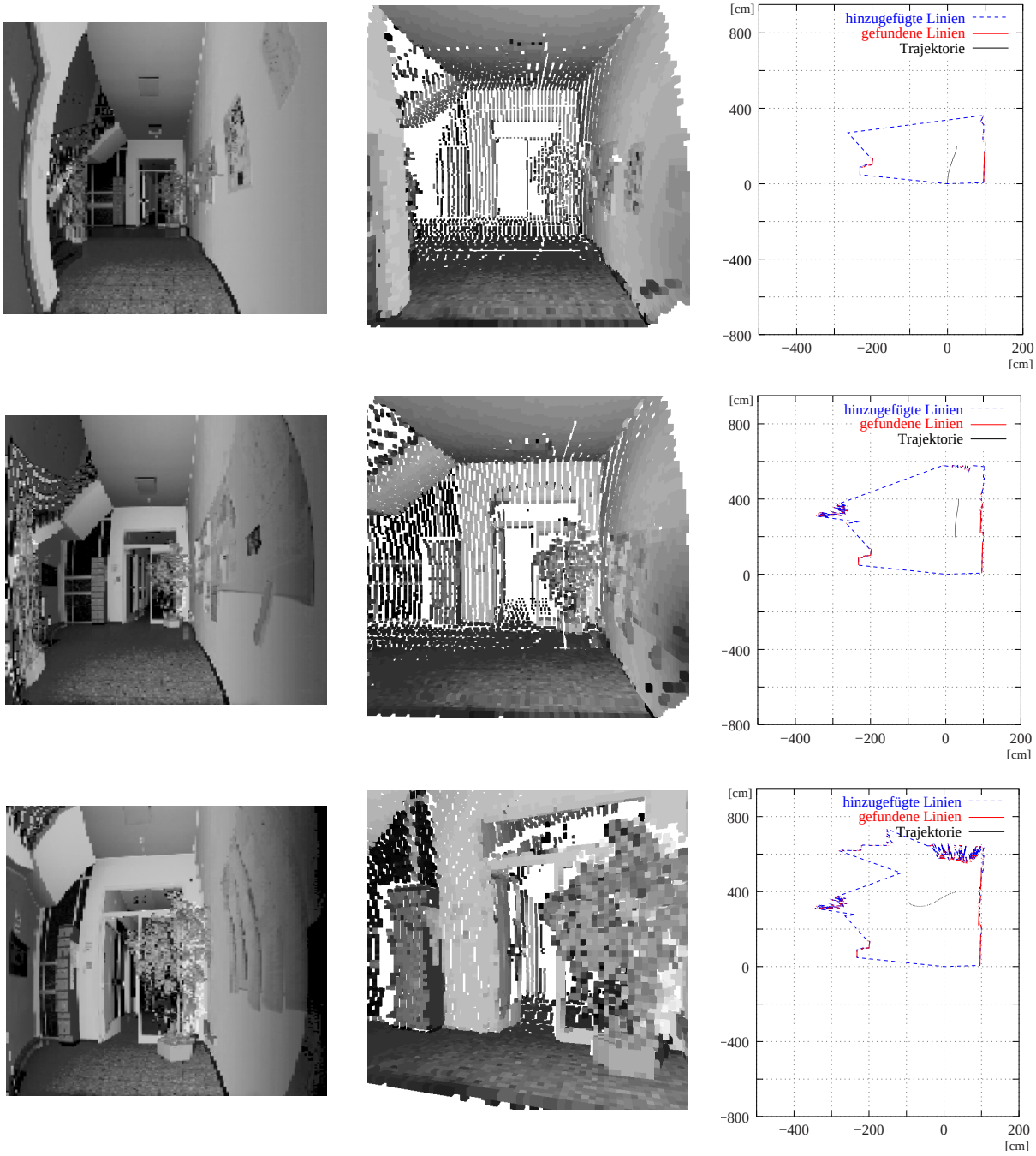


**Abbildung 5.7:** Die Fahrt des Roboters durch die Eingangshalle des FhG-AIS Gebäudes C2. Es werden die ersten drei 3D-Scans dargestellt, wobei jeweils die Remissionswerte, der mit den Remissionswerten eingefärbte Octalbaum und eine Riss-Ebene zu sehen sind. Zusätzlich wird die geplante Trajektorie des Roboters gezeigt.

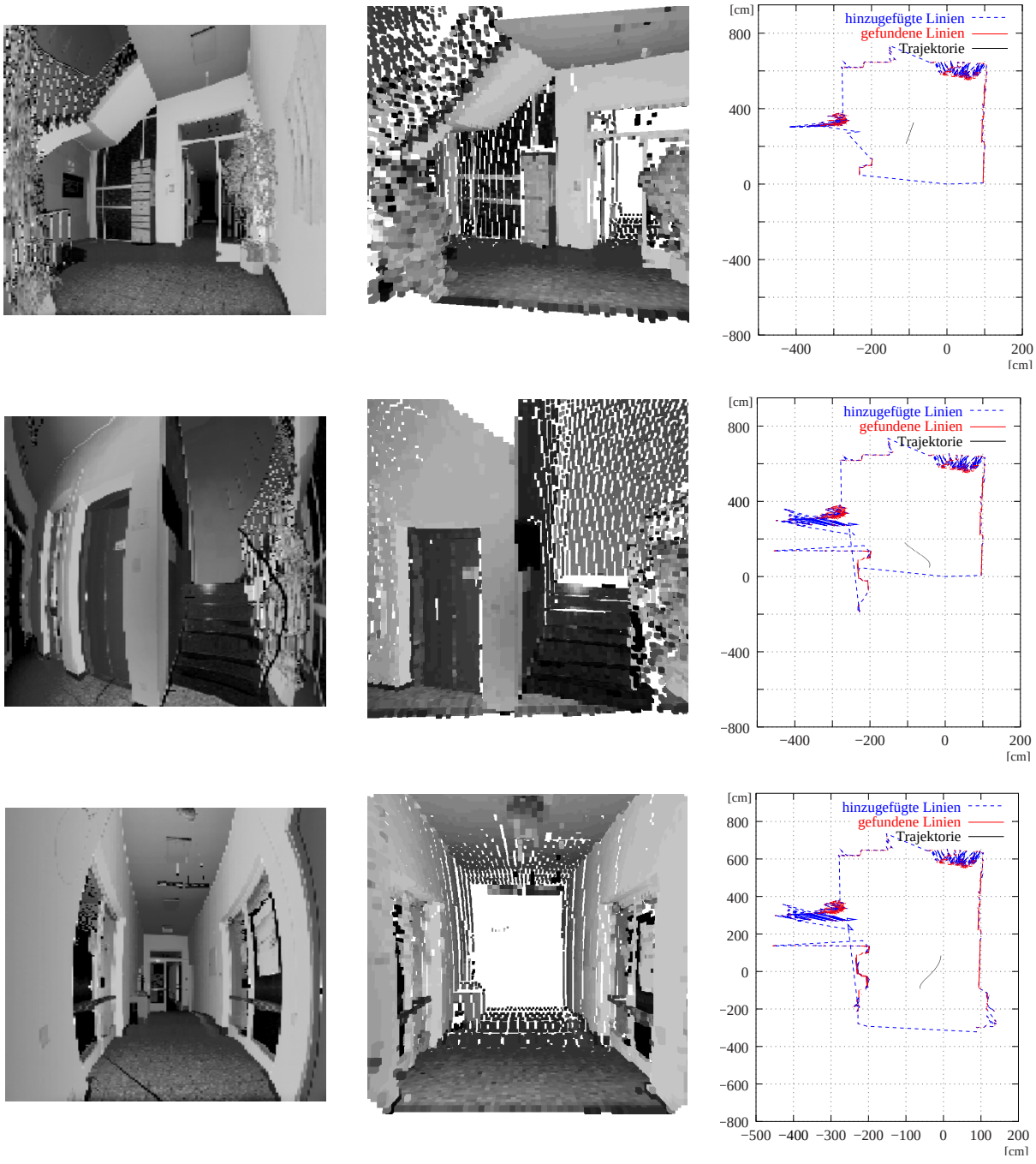


**Abbildung 5.8:** Die Fahrt des Roboters durch die Eingangshalle des FhG-AIS Gebäudes C2. Es werden die Scans 4 bis 6 dargestellt, wobei jeweils die Remissionswerte, der mit den Remissionswerten eingefärbte Octalbaum und eine Riss-Ebene zu sehen sind. Zusätzlich wird die geplante Trajektorie des Roboters gezeigt.

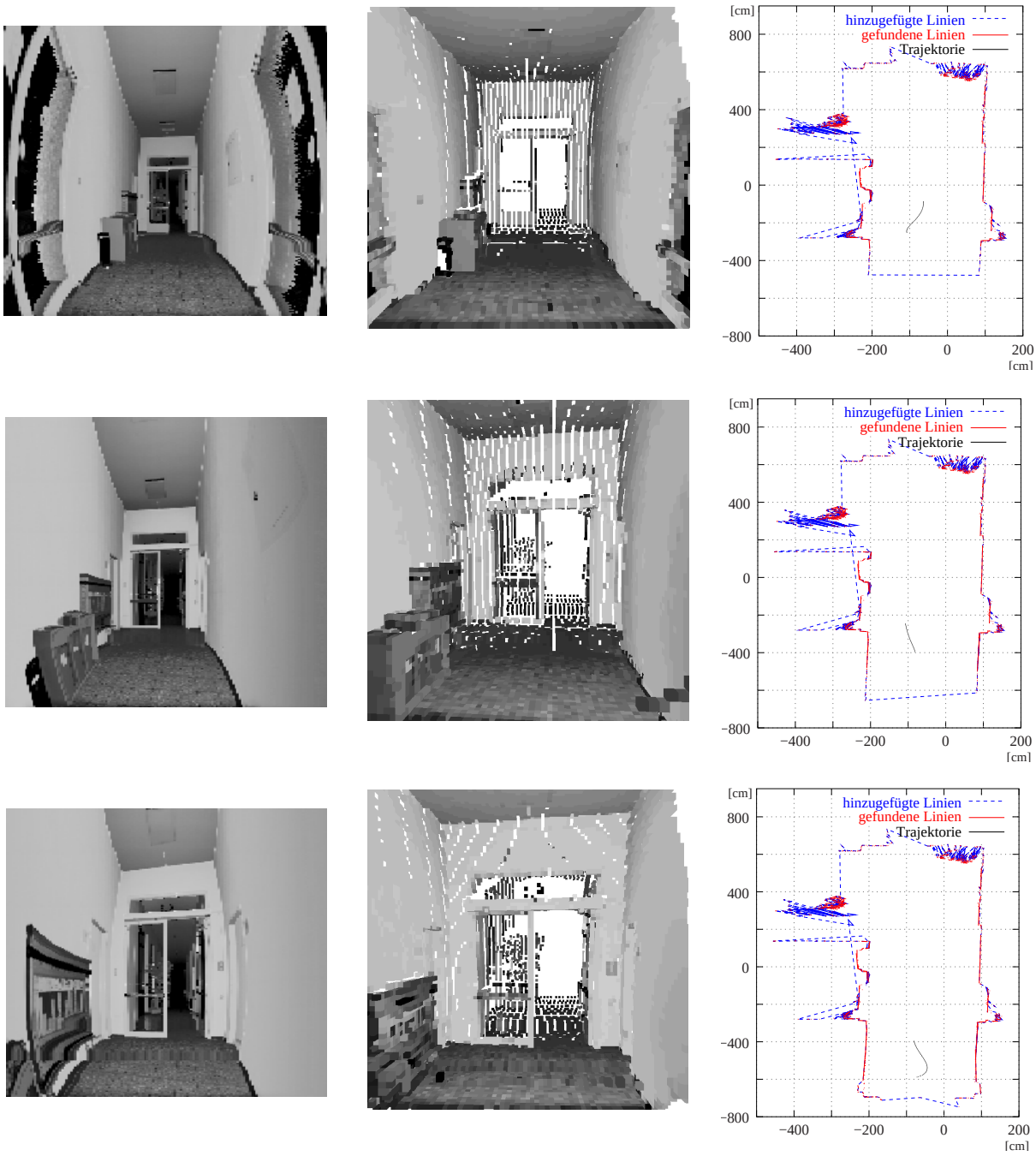


**Abbildung 5.9:** Die Fahrt des Roboters durch die Eingangshalle des FhG-AIS Gebäudes C2. Es werden die Scans 7 bis 9 dargestellt, wobei jeweils die Remissionswerte, der mit den Remissionswerten eingefärbte Octalbaum und eine Riss-Ebene zu sehen sind. Zusätzlich wird die geplante Trajektorie des Roboters gezeigt.

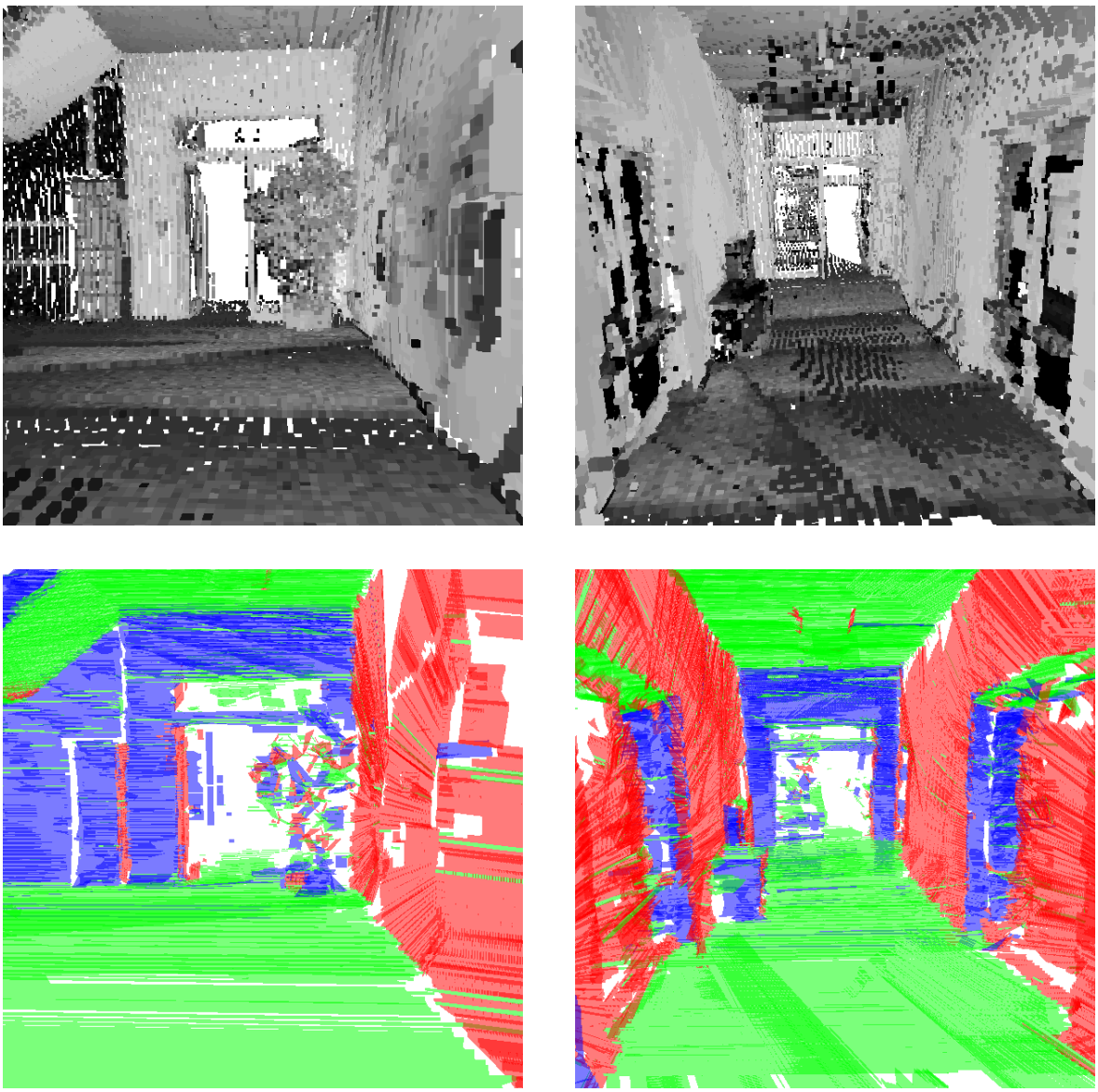

**Abbildung 5.10:** Modell der Eingangshalle aus 9 Scans (vgl. Abbildungen 5.7 – 5.9). Oben: Octalbaumdarstellungen. Unten: Darstellungen der Flächen, die als Linienstreifen gerendert wurden.

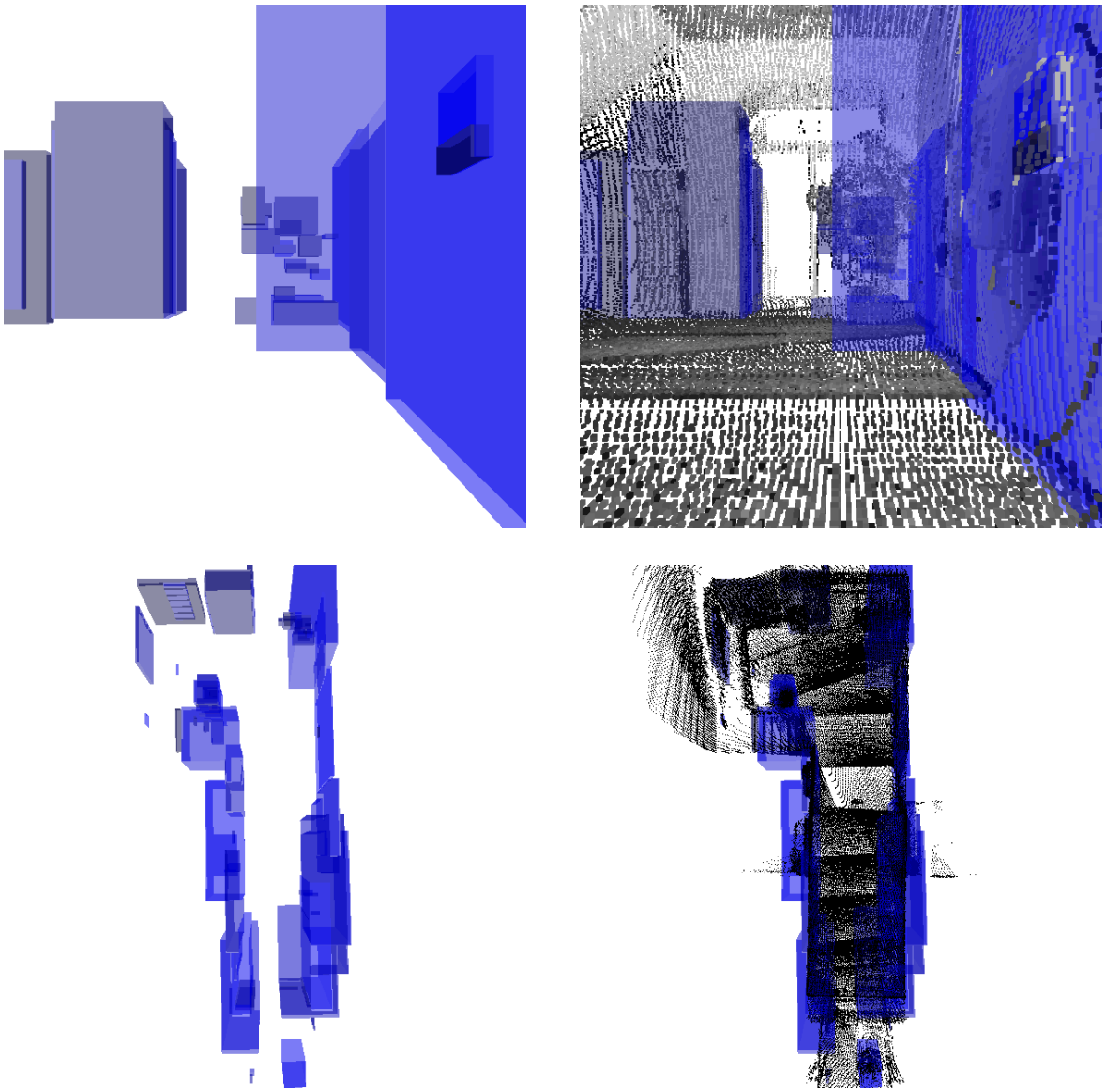

**Abbildung 5.11:** Modell der Eingangshalle, erzeugt aus den 9 Scans der Abbildungen 5.7 – 5.9. Oben: Erkannte Objekte und dazugehörige Messpunkte. Die Kameraposition entspricht der in Abbildung 5.10 oben. Unten: Objekte und dazugehörige Messpunkte (Ansicht von oben).

### Experimente im Flur (Robobench)

Ein weiteres Experiment findet im Flur des FhG-AIS Gebäudes C2 statt. Der Flur ist zirka 30 Meter lang. Abbildung 5.12 zeigt den Weg des Roboters in einer Riss-Ebene. Die Diskontinuitäten im Pfad kommen durch das Registrieren (paarweises Scanmatching) der 3D-Scans zustande, da dabei gleichzeitig die Roboterposition korrigiert wird. Weiterhin sind die drei verschiedenen Fahrweisen des Roboters zu erkennen: Direktes Fahren mit dem Motorcontroller; erst Drehung in Richtung des Ziels und anschließendes direktes Fahren mit dem Controller; Drehung in Richtung des Ziels, Anfahren des Ziels in einer geraden Linie, danach Einstellen der gewünschten Orientierung.

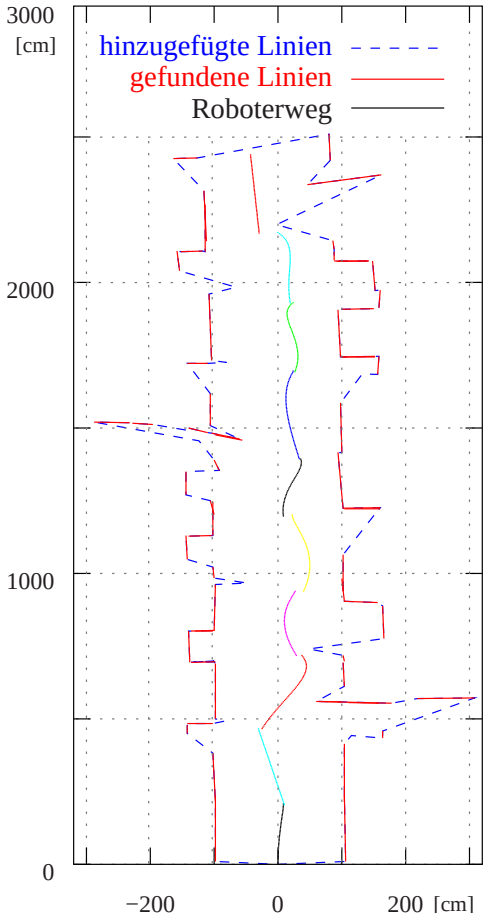


**Abbildung 5.12:** Die Fahrt des Roboters durch das FhG-AIS Gebäude C2. Die roten Kanten des Risses wurden gescannt, hinter den blauen liegt noch nicht erfasstes Gebiet. Der Weg des Roboters startet bei (0,0). Die Scanpositionen befinden sich jeweils am Ende einer Trajektorie (Farbwechsel). Sprünge im Roboterweg kommen durch das Ausgleichen der Positionierungsfehler mit Hilfe von Scanmatching zustande.

Den aus diesem Beispiellauf erzeugten Octalbaum zeigt Abbildung 5.13 in drei verschiedenen Ansichten. Die erkannten 3D-Objekte/Hindernisse sind zumeist Wände und Türen und werden in Abbildung 5.14 dargestellt. Ihre Erkennung dient zur Kollisionsvermeidung im Planungsalgorithmus.

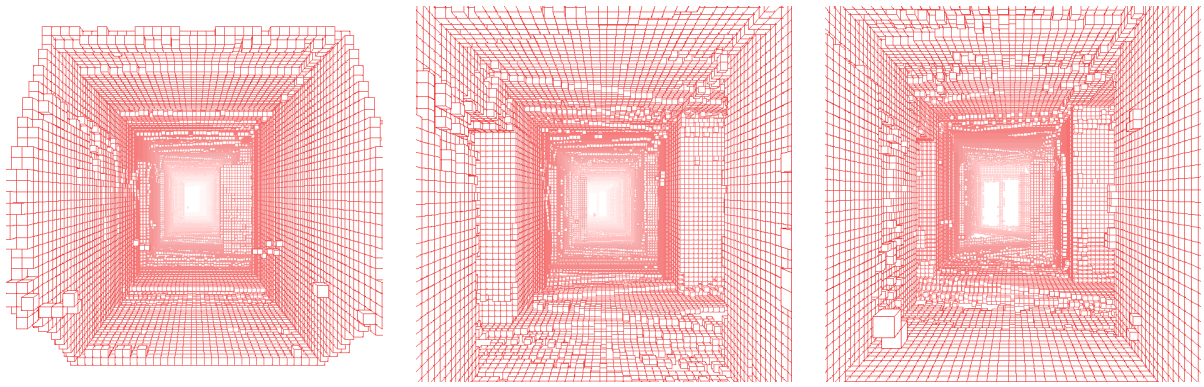

**Abbildung 5.13:** Octalbaum aus einer Roboterfahrt (vgl. Abbildung 5.12). Der Baum wird aus drei Perspektiven gezeigt, die der Startposition des Roboters, der Position nach 10 und jener nach 20 Metern Fahrt entsprechen. Die dargestellte Szene wurde zusätzlich mit „Nebel“ versehen, um Gitterlinien mit großer Entfernung zur virtuellen Kamera auszublenden, bzw. sie mit einer weniger kräftigeren Farbe zu zeichen.

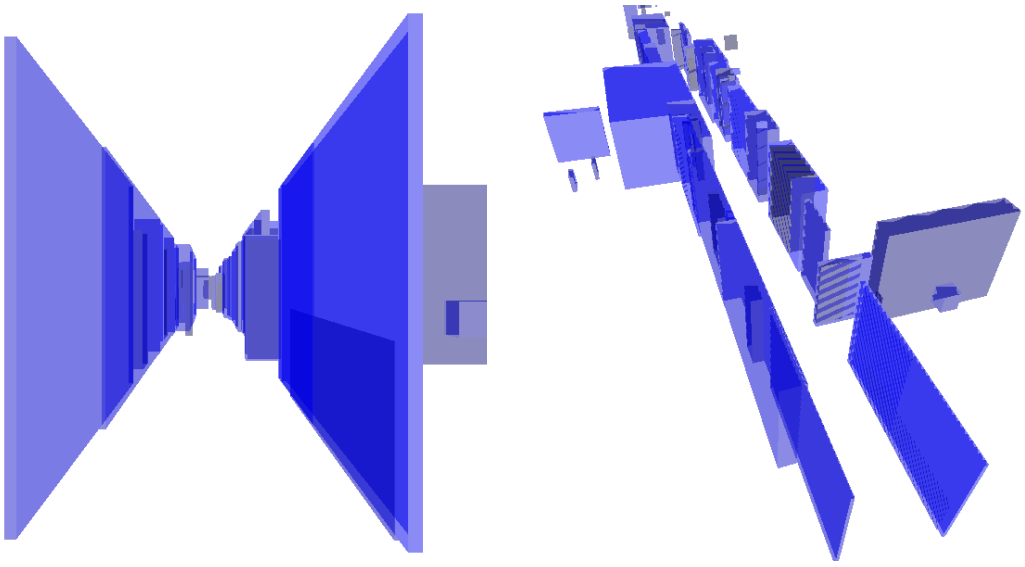

**Abbildung 5.14:** Erkannte Objekte während der Roboterfahrt (vgl. Abbildung 5.12).

### Experimente im Schloss Birlinghoven

Als letztes Experiment wurde der Roboter im großen Saal von Schloss Birlinghoven getestet. Abbildung 5.15 zeigt den Roboter in der dortigen Umgebung (oben) und einen kleinen Ausschnitt des erzeugten Modells (unten), wobei wiederum der mit Remissionswerten eingefärbte Octalbaum gezeigt wird. Ein Film von der Roboterfahrt und Animationen durch die rekonstruierte Szene befinden sich auf der beigefügten CD.

**Abbildung 5.15:** Der Roboter scannt den großen Saal des Schlosses Birlinghoven. Oben: Roboter im Schloss. Unten: rekonstruierte Szene.

# Kapitel 6

# Zusammenfassung und Ausblick

Roboter benötigen zur sicheren Navigation 3D-Sensoren und 3D-Karten. Die Akquisition eines 3D-Modells ist daher eine wichtige Fragestellung in der Robotik. Die vorliegende Arbeit hat gezeigt, dass sich der AIS 3D-Laserscanner zur Erstellung eines dreidimensionalen Weltmodells eignet. Effiziente Algorithmen für diesen Sensor werden vorgestellt. Durch die Kombination des 3D-Laserscanners und des Roboters mit den ausgefeilten Algorithmen entsteht ein geschlossenes System zur Exploration und Modellierung von Umwelten.

Bereits während der Aufnahme der 3D-Scans beginnt die Extraktion von Fakten über die Welt, indem auf jeder Schnittebene Linien erkannt und diese zu Flächen zusammengeführt werden. Die Objekterkennung liefert danach Bounding-Boxen um Objekte herum. Der Roboter kennt somit die Areale, in die er nicht fahren kann.

Anschließend werden aus verschiedenen Positionen aufgenommene 3D-Scans zusammengefügt, also registriert. Gleichzeitig wird dabei die Pose des Roboters korrigiert. Im Mittelpunkt stehen Algorithmen, die das Scanmatching trotz der Datenflut von bis zu 200000 Messpunkten pro 3D-Scan sehr schnell ausführen. Simultanes Scanmatching propagiert den beim Zusammenfügen der Scans entstehenden Fehler auf bereits registrierte Scans zurück und minimiert den globalen Fehler. Doch ist das paarweise Matching wegen des geringeren Rechenaufwands für eine Roboterkontrollarchitektur vorzuziehen. Es kann sehr schnell ausgeführt werden ($< 2$ Sekunden).

Auf der Grundlage theoretischer Ergebnisse wird dann ein schneller randomisierter Approximationsalgorithmus für die Berechnung der nächsten optimalen Scanposition vorgestellt. Die implementierte Strategie lässt den Roboter in Richtung großer noch nicht explorierter Teile der Karte fahren. Der geplante Weg führt an den Bounding-Boxen der Objekte vorbei. Eine Kollision mit statischen Hindernissen ist somit ausgeschlossen. Während der Fahrt kommt zusätzlich Kollisionsvermeidung mittels der 2D-Laserscanner zum Einsatz. Gesteuert wird der Roboter durch eine geschlossene Motorregelung.

2D-Laserscanner gehören bereits zum Standard in der Robotik. Zunehmend werden nun aber 3D-Laserscanner als Sensoren eingesetzt [22, 32, 75, 82]. Die Entwicklung wird auch hier weitergehen. So erforschen verschiedene Gruppen die Konstruktion von 3D-Kameras [49, 66], die ähnlich wie der in dieser Arbeit verwendete Scanner funktionieren. Ein Laserscanner sendet einen

Lichtstrahl aus und misst die Zeit, bis reflektiertes Licht zum Sensor zurückkehrt. Aus dieser Zeit kann die Tiefeninformation errechnet werden. Die Kameras arbeiten ebenfalls nach dem Lichtlaufzeitverfahren. Die abzutastende Szene wird mit Licht geblitzt und der Sensor ermittelt für alle Richtungen, wie viel Licht reflektiert wird [2, 49, 66]. Dieser Prozess benötigt nur den Bruchteil einer Sekunde, so dass die Wiederholrate von Tiefenbildern bei mehreren Hertz liegt [49].

Diese Entwicklung verdeutlicht, dass bei der Performanz der Algorithmen angesetzt werden muss, um eine hohe Wiederholrate beim Registrieren zu erreichen. Das Scanmatching mit ICP-Algorithmus lässt sich auf Tiefenbildern in weniger als 2 Sekunden durchführen, wenn die Odometrie des Roboters eine erste Schätzung liefert. Es besteht also ein Bedarf an genauen Odometriewerten, um den Prozess zu beschleunigen. Auch wird intensiv nach weiteren Geschwindigkeitsverbesserungen beim Scanmatching mit ICP-Algorithmus gesucht, wie die aktuellen Referenzen belegen [62, 65].

Darüber hinaus ist das Planen im Zusammenhang mit Sensoren ebenfalls ein sehr aktuelles Forschungsgebiet [30]. Da es sich um NP-vollständige Probleme handelt, sind gute und effiziente Approximationen gefragt. Es stellt sich die interessante und noch zu behandelnde Frage des Vergleichs mit anderen Strategien. Die Kompetitivität muss untersucht werden. Wie oben beschrieben, versprechen neue Sensoren zudem, kontinuierlich einen Strom von 3D-Daten zu liefern. Damit ist das Sensorplanen nicht mehr mit dem Kunstausstellungsproblem vergleichbar. Es resultiert vielmehr im so genannten Wachpostenweg- (engl.: *watchman route problem*) bzw. Polygonexpolorationsproblem. Hierbei sind nicht diskrete Orte gesucht, die den Überblick über die gesamte Karte erlauben, sondern ein Weg, von dem aus die gesamte Karte eingesehen werden kann [38, 48, 70].

Als die wichtigste Aufgabe [36] bei der Roboternavigation mit Hilfe von Tiefenbildern wird die Kombination der deterministischen Techniken zur Registrierung (vgl. Kapitel 3) mit den stochastischen Lokalisations-Techniken [79] genannt. Es muss das Ziel verfolgt werden, die Genauigkeit der Registrierungstechniken mit der Flexibilität der Lokalisierungsmethoden zu verbinden [36].

Des Weiteren muss die Kontrollarchitektur des Roboters noch weiter ausgebaut werden. So ist es dem Roboter zurzeit nicht möglich, aus Sackgassen herauszumanövrieren oder den Weg zur optimalen nächsten Scanposition mit mehreren Zwischenstationen anzufahren.

Eine weitere noch zu lösende Aufgabe ergibt sich aus dem Wunsch, die Handlungsfähigkeit des Roboters zu verbessern. Kollisionsvermeidung ist, wie diese Arbeit auch gezeigt hat, mit den in Abschnitt 2.4.2 vorgestellten Boxen um Objekte herum möglich. Auf einer in dieser Art und Weise segmentierten Szene könnte man nun auch Objekterkennung betreiben und eine symbolische Repräsentation erzeugen.

# Anhang A

# Herleitungen und Beweise

## A.1 Das Quaternion zur Darstellung von Rotationen

Quaternionen verallgemeinern das Konzept der komplexen Zahlen [18]. Sir William Rowan Hamilton hat 1833 als erster gezeigt, dass die komplexen Zahlen eine Algebra formen, d.h. es lassen sich auf der Basis von Zahlenpaaren konsistente Rechenregeln definieren. Dabei wird eine komplexe Zahl $z \in \mathbb{C}$ mittels $\mathfrak{i}$, der Wurzel aus $-1$ dargestellt. Obwohl $\mathfrak{i}$ keine reelle Zahl ist, kann eine komplexe Zahl durch reelle Zahlen angeben werden, indem ein Real- und Imaginärteil benutzt wird:

$$z = a + b\mathfrak{i} \quad \text{mit } a, b \in \mathbb{R}.$$

Die konjugiert komplexe Zahl und der absolute Betrag der Zahl $z$ können ebenfalls definiert werden:

$$\begin{aligned} z^* &= a - b\mathfrak{i} \\ \|z\| &= \sqrt{zz^*} = \sqrt{a^2 + b^2}. \end{aligned}$$

Unter Benutzung dieser Eigenschaften kann man die Multiplikation von $z_1 = a_1 + b_1\mathfrak{i}$ mit $z_2 = a_2 + b_2\mathfrak{i}$ wie folgt beschreiben:

$$z_1 z_2 = (a_1 a_2 - b_1 b_2) + (a_1 b_2 + b_1 a_2)\mathfrak{i}.$$

Nach der Entwicklung der komplexen Zahlen hat Hamilton 10 Jahre lang vergeblich versucht, dieses Konzept auf Zahlentripel, also komplexe Zahlen mit zwei imaginären Anteilen, zu erweitern [1, 34]. Es wird erzählt, dass ihm die Lösung, vier Zahlen zu verwenden, schließlich unvermutet während eines Spaziergangs mit seiner Frau einfiel und er so begeistert war, dass er sie (A.1) spontan in den Brückenpfeiler der Broom Bridge (heute Hamilton Bridge) in Dublin gravierte [1].

Das Quaternion erweitert das Konzept der komplexen Zahlen und ist als 4-Tupel definiert [34, 35].

$$\dot{q} = q_0 + \mathfrak{i}q_x + \mathfrak{j}q_y + \mathfrak{k}q_z \quad \text{mit } q_0, q_x, q_y, q_z \in \mathbb{R}.$$

Dabei gelten die folgenden Gleichungen:

$$\mathfrak{i}^2 = \mathfrak{j}^2 = \mathfrak{k}^2 = -1, \qquad \mathfrak{ij} = \mathfrak{k},\ \mathfrak{ji} = -\mathfrak{k}. \tag{A.1}$$

Die Multiplikation ist ähnlich wie jene der komplexen Zahlen definiert. Sei $\mathbf{i} = (\mathfrak{i}, \mathfrak{j}, \mathfrak{k})^T$ der Spaltenvektor mit den imaginären Anteilen und $\mathbf{q} = (q_x, q_y, qz)^T$. Dann ist

$$\dot{q} = q_0 + \mathbf{q} \cdot \mathbf{i}.$$

Nimmt man weiterhin an, dass $\dot{q}_1$ und $\dot{q}_2$ zwei Quaternionen sind, so ergibt sich für die Multiplikation

$$\begin{aligned}
\dot{q}_1\dot{q}_2 &= (q_{10} + \mathbf{q}_1 \cdot \mathbf{i})(q_{20} + \mathbf{q}_2 \cdot \mathbf{i}) \\
&= q_{10}q_{20} - q_{1x}q_{2x} - q_{1y}q_{2y} - q_{1z}q_{2z} \\
&\quad +(q_{10}q_{2x} + q_{1x}q_{20} + q_{1y}q_{2z} - q_{1z}q_{2y})\mathfrak{i} \\
&\quad +(q_{10}q_{2y} - q_{1x}q_{2z} + q_{1y}q_{20} + q_{1z}q_{2x})\mathfrak{j} \\
&\quad +(q_{10}q_{2z} + q_{1x}q_{2y} - q_{1y}q_{2x} + q_{1z}q_{20})\mathfrak{k} \qquad (A.2) \\
&= q_{10}q_{20} + (q_{10}\mathbf{q}_2 + q_{20}\mathbf{q}_1) \cdot \mathbf{i} + (\mathbf{q}_1 \cdot \mathbf{i})(\mathbf{q}_2 \cdot \mathbf{i}). \qquad (A.3)
\end{aligned}$$

Mit (A.1) folgt für den letzten Term

$$\begin{aligned}
(\mathbf{q}_1 \cdot \mathbf{i})(\mathbf{q}_2 \cdot \mathbf{i}) &= (q_{1x}\mathfrak{i} + q_{1y}\mathfrak{j} + q_{1z}\mathfrak{k})(q_{2x}\mathfrak{i} + q_{2y}\mathfrak{j} + q_{2z}\mathfrak{k}) \\
&= -(q_{1x}q_{2x} + q_{1y}q_{2y} + q_{1z}q_{2z} + (q_{1y}q_{2z} - q_{1z}q_{2y})\mathfrak{i} \\
&\quad +(q_{1z}q_{2x} - q_{1x}q_{2z})\mathfrak{j} + (q_{1x}q_{2y} - q_{1y}q_{2x})\mathfrak{k} \\
&= -\mathbf{q}_1 \cdot \mathbf{q}_2 + (\mathbf{q}_1 \times \mathbf{q}_2) \cdot \mathbf{i}
\end{aligned}$$

und für (A.3) ergibt sich somit

$$\dot{q}_1\dot{q}_2 = (q_{10}q_{20} - \mathbf{q}_1 \cdot \mathbf{q}_2) + (q_{10}\mathbf{q}_2 + q_{20}\mathbf{q}_1 + \mathbf{q}_1 \times \mathbf{q}_2) \cdot \mathbf{i}. \tag{A.4}$$

Analog zu den komplexen Zahlen wird das konjugierte Quaternion definiert. Sei $\dot{q} = q_0 + \mathbf{q} \cdot \mathbf{i}$, dann ist $\dot{q}^* = q_0 - \mathbf{q} \cdot \mathbf{i}$ das konjugierte Quaternion. Aus (A.4) folgt

$$\dot{q}\dot{q}^* = q_0^2 + ||\mathbf{q}||^2 .$$

Dies entspricht genau der 2-Norm des Vektors $(q_0, q_x, q_y, q_z)^T$. $\sqrt{\dot{q}\dot{q}^*}$ wird Betrag des Quaternions genannt und mit $||\dot{q}||$ bezeichnet. Ein Quaternion mit Betrag 1 ist ein so genanntes Einheitsquaternion. Wie man leicht sieht, existiert zu jedem von Null verschiedenen Quaternion $\dot{q}$ ein Quaternion $\dot{q}^{-1}$, so dass $\dot{q}\dot{q}^{-1} = 1$. Dabei bezeichnet $\dot{q}^{-1}$ das inverse Quaternion zu $\dot{q}$.

Die Menge aller von Null verschiedenen Quaternionen mit der oben definierten Multiplikation formen eine nicht kommutative Gruppe. Es kann gezeigt werden, dass die Multiplikation zweier Einheitsquaternionen zu einem Einheitsquaternion führt. Somit formen die Einheitsquaternionen eine Untergruppe, was für die Darstellung einer Rotation notwendig ist. Für ein Einheitsquaternion gilt weiterhin die folgende Beziehung $\dot{q}^{-1} = q^*$.

Das Skalarprodukt zweier Quaternionen $\dot{q}_1$ und $\dot{q}_2$ ist die Summe aus den Produkten der Komponenten

$$\dot{q}_1 \cdot \dot{q}_2 = q_{10}q_{20} + q_{1x}q_{2x} + q_{1y}q_{2y} + q_{1z}q_{2z}. \tag{A.5}$$

Somit gilt folgende Beziehung immer:

$$\dot{q} \cdot \dot{q} = (q_0^2 + q_1^2 + q_2^2 + q_3^2) = \dot{q}\dot{q}^*.$$

Seien $\dot{p}, \dot{q}, \dot{r}$ Quaternionen. Folgende Gleichungen lassen sich mit den Definitionen (A.5) und (A.2) leicht verifizieren. Sie spielen in Kapitel A.2.2 eine wichtige Rolle:

$$(\dot{p}\dot{q}) \cdot \dot{r} = \dot{p} \cdot (\dot{q}\dot{r}) = \dot{p} \cdot (\dot{r}\dot{q}^*)\,. \tag{A.6}$$

### A.1.1 Berechnung der Rotationsmatrix aus dem Einheitsquaternion

Jedes Quaternion kann durch eine $4 \times 4$ Matrix repräsentiert werden. Sei das Quaternion $\dot{q} = q_0 + \mathbf{q} \cdot \mathbf{i}$ mit $\mathbf{q} = (q_x, q_y, q_z)^T$, dann wird eine Matrix $\mathbf{Q}$ wie folgt definiert:

$$\mathbf{Q} = \begin{pmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & -q_z & q_y \\ q_y & q_z & q_0 & -q_x \\ q_z & -q_y & q_x & q_0 \end{pmatrix} = \begin{pmatrix} q_0 & -\mathbf{q}^T \\ \mathbf{q} & q_0\mathbb{1}_{3\times 3} + \mathbf{C}_\mathbf{q} \end{pmatrix} \tag{A.7}$$

mit

$$\mathbf{C}_\mathbf{q} = \begin{pmatrix} 0 & -q_z & q_y \\ q_z & 0 & -q_x \\ -q_y & q_x & 0 \end{pmatrix}.$$

Die Schreibweise eines Quaternions als Matrix stellt eine bijektive Abbildung zwischen der Menge der Quaternionen und der Menge der $4 \times 4$ Matrizen der Form (A.7) dar. Wie folgender Satz zeigt, entspricht die Multiplikation von Matrizen der Form (A.7) jener von Quaternionen.

**Satz 5.** Seien $\dot{q}_1 = q_{10} + \mathbf{q}_1 \cdot \mathbf{i}$ und $\dot{q}_2 = q_{20} + \mathbf{q}_2 \cdot \mathbf{i}$ zwei Quaternionen und $\mathbf{Q}_1$ und $\mathbf{Q}_2$ die zugehörigen Matrix-Darstellungen. Wenn $\dot{q}_1\dot{q}_2 = q_0 + \mathbf{q} \cdot \mathbf{i}$ ist, dann gilt:

$$\mathbf{Q}_1\mathbf{Q}_2 = \begin{pmatrix} q_0 & -\mathbf{q}^T \\ \mathbf{q} & q_0\mathbb{1}_{3\times 3} + \mathbf{C}_\mathbf{q} \end{pmatrix}.$$

**Beweis:** Bezeichne $\mathbb{1}$ die $4 \times 4$ Einheitsmatrix. Damit gilt:

$$\mathbf{Q}_1 = q_{10}\mathbb{1} + \begin{pmatrix} 0 & -\mathbf{q}_1^T \\ \mathbf{q}_1 & \mathbf{C}_{\mathbf{q}_1} \end{pmatrix} \qquad \mathbf{Q}_2 = q_{20}\mathbb{1} + \begin{pmatrix} 0 & -\mathbf{q}_2^T \\ \mathbf{q}_2 & \mathbf{C}_{\mathbf{q}_2} \end{pmatrix}$$

$$\begin{aligned} \mathbf{Q}_1\mathbf{Q}_2 \;=\; & q_{10}q_{20}\mathbb{1} + q_{10}\begin{pmatrix} 0 & -\mathbf{q}_2^T \\ \mathbf{q}_2 & \mathbf{C}_{\mathbf{q}_2} \end{pmatrix} + q_{20}\begin{pmatrix} 0 & -\mathbf{q}_1^T \\ \mathbf{q}_1 & \mathbf{C}_{\mathbf{q}_1} \end{pmatrix} \\ & + \underbrace{\begin{pmatrix} 0 & -\mathbf{q}_1^T \\ \mathbf{q}_1 & \mathbf{C}_{\mathbf{q}_1} \end{pmatrix}\begin{pmatrix} 0 & -\mathbf{q}_2^T \\ \mathbf{q}_2 & \mathbf{C}_{\mathbf{q}_2} \end{pmatrix}}_{\begin{pmatrix} -\mathbf{q}_1 \cdot \mathbf{q}_2 & -(\mathbf{q}_1 \times \mathbf{q}_2)^T \\ \mathbf{q}_1 \times \mathbf{q}_2 & -\mathbf{q}_1 \cdot \mathbf{q}_2\mathbb{1} + \mathbf{C}_{\mathbf{q}_1 \times \mathbf{q}_2} \end{pmatrix}}. \end{aligned}$$

Es folgt:

$$\begin{aligned}\mathbf{Q}_1\mathbf{Q}_2 &= (q_{10}q_{20} - \mathbf{q}_1 \cdot \mathbf{q}_2)\mathbb{1} \\ &\quad + \begin{pmatrix} 0 & -(q_{10}\mathbf{q}_2 + q_{20}\mathbf{q}_1 + \mathbf{q}_1 \times \mathbf{q}_2)^T \\ (q_{10}\mathbf{q}_2 + q_{20}\mathbf{q}_1 + \mathbf{q}_1 \times \mathbf{q}_2) & \mathbf{C}_{q_{10}\mathbf{q}_2 + q_{20}\mathbf{q}_1 + \mathbf{q}_1 \times \mathbf{q}_2} \end{pmatrix}.\end{aligned}$$

□

Sei $\mathbf{Q}$ die Matrix-Darstellung des Quaternions $\dot{q} = q_0 + \mathbf{q} \cdot \mathbf{i}$. Aus der Definition der Matrix-Darstellung wird ersichtlich, dass die Matrix-Repräsentation von $\dot{q}^*$ gerade die transponierte Matrix $\mathbf{Q}^T$ ist. Interessant ist die folgende Beziehung:

$$\mathbf{Q}\mathbf{Q}^T = (q_0^2 + ||\mathbf{q}||^2)\mathbb{1} = ||\mathbf{q}||^2 \mathbb{1}.$$

Aus diesem Grund ist die Matrix $\mathbf{Q}$ immer orthogonal, wenn $\mathbf{q} \neq 0$ ist. Speziell für ein Einheitsquaternion gilt folgender Satz:

**Satz 6.** Für ein Einheitsquaternion $\dot{q} = q_0 + \mathbf{q} \cdot \mathbf{i}$ ist die zugehörige Matrix $\mathbf{Q}$ eine $4 \times 4$ Rotationsmatrix, d.h. es gilt:

$$\mathbf{Q}\mathbf{Q}^T = \mathbb{1} \qquad \det(\mathbf{Q}) = 1.$$

**Beweis:** Für ein Einheitsquaternion gilt: $q_0^2 + ||\mathbf{q}||^2 = 1$. Nach (A.7) ist

$$\mathbf{Q} = \begin{pmatrix} q_0 & -\mathbf{q}^T \\ \mathbf{q} & q_0\mathbb{1}_{3\times3} + \mathbf{C}_\mathbf{q} \end{pmatrix} \qquad \mathbf{Q}^T = \begin{pmatrix} q_0 & \mathbf{q}^T \\ -\mathbf{q} & q_0\mathbb{1}_{3\times3} - \mathbf{C}_\mathbf{q} \end{pmatrix}$$

und somit

$$\mathbf{Q}\mathbf{Q}^T = \begin{pmatrix} q_0^2 + ||\mathbf{q}||^2 & 0 \\ 0 & \mathbf{q}\mathbf{q}^T + q_0^2\mathbb{1}_{3\times3} - \mathbf{C}_\mathbf{q}\mathbf{C}_\mathbf{q} \end{pmatrix}.$$

Wegen $\mathbf{C}_\mathbf{q}\mathbf{C}_\mathbf{q} = \mathbf{q}\mathbf{q}^T - ||\mathbf{q}||^2 \mathbb{1}$ ergibt sich:

$$\mathbf{Q}\mathbf{Q}^T = \begin{pmatrix} q_0^2 + ||\mathbf{q}||^2 & 0 \\ 0 & (q_0^2 + ||\mathbf{q}||^2)\mathbb{1}_{3\times3} \end{pmatrix} = (q_0^2 + ||\mathbf{q}||^2)\mathbb{1}.$$

Der zweite Teil des Satzes wird mit Hilfe des Laplaceschen Entwicklungssatzes [21] bewiesen:

$$\det(\mathbf{Q}) = \det\begin{pmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & -q_z & q_y \\ q_y & q_z & q_0 & -q_x \\ q_z & -q_y & q_x & q_0 \end{pmatrix}$$

$$\begin{aligned}
\det(\mathbf{Q}) &= q_0 \det\begin{pmatrix} q_0 & -q_z & q_y \\ q_z & q_0 & -q_x \\ -q_y & q_x & q_0 \end{pmatrix} + q_x \det\begin{pmatrix} q_x & -q_z & q_y \\ q_y & q_0 & -q_x \\ q_z & q_x & q_0 \end{pmatrix} \\
&\quad -q_y \det\begin{pmatrix} q_x & -q_0 & -q_y \\ q_y & q_z & -q_x \\ q_z & -q_y & q_0 \end{pmatrix} + q_z \det\begin{pmatrix} q_x & q_0 & -q_z \\ q_y & q_z & q_0 \\ q_z & -q_y & q_x \end{pmatrix} \\
&= q_0^4 + q_0^2(q_x^2+q_y^2+q_z^2) + q_x^4 + q_x^2(q_0^2+q_y^2+q_z^2) \\
&\quad + q_y^4 + q_y^2(q_0^2+q_x^2+q_z^2) + q_z^4 + q_z^2(q_0^2+q_x^2+q_y^2) \\
&= q_0^4 + q_0^2(1-q_0^2) + q_x^4 + q_x^2(1-q_x^2) + q_y^4 + q_y^2(1-q_y^2) + q_z^4 + q_z^2(1-q_z^2) \\
&= q_0^2 + q_x^2 + q_y^2 + q_z^2 \\
&= 1.
\end{aligned}$$

□

Definiert man für ein Quaternion $\dot{q} = q_0 + \mathbf{q}\cdot\mathbf{i}$ die Matrix

$$\bar{\mathbf{Q}} = \begin{pmatrix} q_0 & -\mathbf{q}^T \\ \mathbf{q} & q_0\mathbb{1}_{3\times 3} - \mathbf{C_q} \end{pmatrix}, \tag{A.8}$$

erhält man ähnliche Eigenschaften wie für $\mathbf{Q}$ aus Definition (A.7). Wenn $\dot{q}$ ein Einheitsquaternion ist, dann stellt auch $\bar{\mathbf{Q}}$ eine Rotationsmatrix im 4-dimensionalen Raum dar und es gilt

$$\bar{\mathbf{Q}}\bar{\mathbf{Q}}^T = \mathbb{1} \qquad \det(\bar{\mathbf{Q}}) = 1.$$

**Herleitung der Formel für die Rotationsmatrix.** Für ein Einheitsquaternion $\dot{q} = q_0 + \mathbf{q}\cdot\mathbf{i}$ sind $\mathbf{Q}$ und $\bar{\mathbf{Q}}$ wie in (A.7), (A.8) definiert. Außerdem handelt es sich um Rotationsmatrizen, weshalb auch $\bar{\mathbf{Q}}^T\mathbf{Q}$ eine Rotationsmatrix ist. Sie hat folgende Form:

$$\begin{aligned}
\bar{\mathbf{Q}}^T\mathbf{Q} &= \begin{pmatrix} q_0 & \mathbf{q}^T \\ -\mathbf{q} & q_0\mathbb{1}_{3\times 3} + \mathbf{C_q} \end{pmatrix}\begin{pmatrix} q_0 & -\mathbf{q}^T \\ \mathbf{q} & q_0\mathbb{1}_{3\times 3} + \mathbf{C_q} \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 \\ 0 & \mathbf{q}\mathbf{q}^T + q_0^2\mathbb{1}_{3\times 3} + \mathbf{C_q}\mathbf{C_q} \end{pmatrix}.
\end{aligned}$$

Wie man sieht, muss die $3\times 3$ Matrix $\mathbf{q}\mathbf{q}^T + q_0^2\mathbb{1}_{3\times 3} + \mathbf{C_q}\mathbf{C_q}$ auch eine Rotationsmatrix sein. Somit entspricht *jedes* Einheitsquaternion *genau einer* Rotationsmatrix in $\mathbb{R}^3$. Diese Matrix hat die folgende Form:

$$\mathbf{R} = \begin{pmatrix} (q_0^2+q_x^2-q_y^2-q_z^2) & 2(q_xq_y + q_zq_0) & 2(q_xq_z + q_yq_0) \\ 2(q_xq_y + q_zq_0) & (q_0^2-q_x^2+q_y^2-q_z^2) & 2(q_yq_z - q_xq_0) \\ 2(q_zq_x - q_yq_0) & 2(q_zq_y + q_xq_0) & (q_0^2-q_x^2-q_y^2+q_z^2) \end{pmatrix}.$$

Die Matrixdarstellung $\mathbf{Q}$ eines Quaternions erlaubt eine effiziente Schreibweise der Multiplikation. Das Produkt zweier Quaternionen lässt sich als Matrix-Vektor-Produkt schreiben, wenn mit

einem Quaternion $\dot{q} = q_0 + \mathfrak{i}q_x + \mathfrak{j}q_y + \mathfrak{k}q_z = q_0 + \mathbf{q} \cdot \mathbf{i}$ der Spaltenvektor $(q_0, q_x, q_y, q_z)^T$ assoziiert wird. Damit ergeben sich $\dot{q}_1$ und $\dot{q}_2$ als

$$\begin{aligned}
\mathbf{Q}_1\dot{q}_2 &:= \left( \begin{pmatrix} q_{10} & -q_{1x} & -q_{1y} & -q_{1z} \\ q_{1x} & q_{10} & -q_{1z} & q_{1y} \\ q_{1y} & q_{1z} & q_{10} & -q_{1x} \\ q_{1z} & -q_{1y} & q_{1x} & q_{10} \end{pmatrix} \begin{pmatrix} q_{20} \\ q_{2x} \\ q_{2y} \\ q_{2z} \end{pmatrix} \right) \cdot \begin{pmatrix} 1 \\ \mathfrak{i} \\ \mathfrak{j} \\ \mathfrak{k} \end{pmatrix} \qquad (A.9) \\
&= q_{10}q_{20} - q_{1x}q_{2x} - q_{1y}q_{2y} - q_{1z}q_{2z} \\
&\quad +(q_{10}q_{2x} + q_{1x}q_{20} + q_{1y}q_{2z} - q_{1z}q_{2y})\mathfrak{i} \\
&\quad +(q_{10}q_{2y} - q_{1x}q_{2z} + q_{1y}q_{20} + q_{1z}q_{2x})\mathfrak{j} \\
&\quad +(q_{10}q_{2z} + q_{1x}q_{2y} - q_{1y}q_{2x} + q_{1z}q_{20})\mathfrak{k} \\
&= \dot{q}_1\dot{q}_2.
\end{aligned}$$

Analog gilt:

$$\bar{\mathbf{Q}}_1\dot{q}_2 = \ \ldots \ = \dot{q}_2\dot{q}_1. \qquad (A.10)$$

Ein Vektor $\mathbf{v} = (v_x, v_y, v_z)^T \in \mathbb{R}^3$ wird durch ein Quaternion dargestellt, das ausschließlich aus einem imaginären Teil besteht, d.h. $\dot{v} = 0 + \mathbf{v} \cdot \mathbf{i} = (0, v_x, v_y, v_z)^T \cdot \mathbf{i}$. Dadurch ist es möglich, eine Rotation durch das Hintereinanderschalten von Multiplikationen der Quaternionen auszudrücken:

$$\mathbf{v}_{rot} = (\bar{\mathbf{Q}}^T\mathbf{Q}) \begin{pmatrix} 0 \\ \mathbf{v} \end{pmatrix} = \bar{\mathbf{Q}}^T (\mathbf{Q}\dot{v}) = (\mathbf{Q}\dot{v})\, \dot{q}^* = \dot{q}\dot{v}\dot{q}^*. \qquad (A.11)$$

Im nun anschließenden Abschnitt wird nachgewiesen, dass das Einheitsquaternion $\dot{q} = q_0 + \mathbf{q} \cdot \mathbf{i}$ mit $q_0 = \cos\frac{\theta}{2}$ und $\mathbf{q} = \mathbf{n}\sin\frac{\theta}{2}$ tatsächlich eine Rotation um den Winkel $\theta$ um den Einheitsvektor $\mathbf{n}$ darstellt (vgl. Abbildung 3.1).

### A.1.2 Berechnung des Einheitsquaternion

Ein Vektor $\mathbf{v} = (v_x, v_y, v_z)^T \in \mathbb{R}^3$ soll um einen Vektor $\mathbf{n}$ um den Winkel $\theta$ gedreht werden. Diese Drehung wird durch das Einheitsquaternion $\dot{q} = q_0 + \mathbf{q} \cdot \mathbf{i}$ mit $q_0 = \cos\frac{\theta}{2}$ und $\mathbf{q} = \mathbf{n}\sin\frac{\theta}{2}$ beschrieben. Eine Möglichkeit, den gesuchten Punkt $\mathbf{v}_{rot}$ zu errechnen, ist die im vorigen Abschnitt vorgestellte Methode, das Quaternion $\dot{v}$ des Vektors $\mathbf{v}$ mit $\dot{q}$ und $\dot{q}^*$ zu multiplizieren (vgl. (A.11)):

$$\begin{aligned}
\mathbf{v}_{rot} &= \dot{q}\dot{v}\dot{q}^* \\
&= \left(\cos\frac{\theta}{2}, \mathbf{n}\sin\frac{\theta}{2}\right)(0, \mathbf{v})\left(\cos\frac{\theta}{2}, -\mathbf{n}\sin\frac{\theta}{2}\right) \\
&= \left(\left(\cos\frac{\theta}{2}, \mathbf{n}\sin\frac{\theta}{2}\right)(0, \mathbf{v})\right)\left(\cos\frac{\theta}{2}, -\mathbf{n}\sin\frac{\theta}{2}\right).
\end{aligned}$$

Daraufhin wird die Quaternionmultiplikation (vgl. (A.4)) angewendet und es ergibt sich:

$$\begin{aligned}
\mathbf{v}_{rot} &= \left(-\sin\frac{\theta}{2}(\mathbf{n}\cdot\mathbf{v}), \cos\frac{\theta}{2}\mathbf{v} + \sin\frac{\theta}{2}(\mathbf{v}\times\mathbf{n})\right)\left(\cos\frac{\theta}{2}, -\mathbf{n}\sin\frac{\theta}{2}\right) \\
&= \Bigg(-\sin\frac{\theta}{2}\cos\frac{\theta}{2}(\mathbf{n}\cdot\mathbf{v}) + \sin\frac{\theta}{2}\cos\frac{\theta}{2}(\mathbf{v}\cdot\mathbf{n}) - \sin^2\frac{\theta}{2}(\mathbf{n}\times\mathbf{v})\cdot\mathbf{n}, \\
&\quad \sin^2\frac{\theta}{2}(\mathbf{n}\cdot\mathbf{v})\mathbf{n} + \cos^2\frac{\theta}{2}\mathbf{v} + \sin\frac{\theta}{2}\cos\frac{\theta}{2}(\mathbf{n}\times\mathbf{v}) - \sin\frac{\theta}{2}\cos\frac{\theta}{2}(\mathbf{v}\times\mathbf{n}) \\
&\quad -\sin^2\frac{\theta}{2}(\mathbf{n}\times\mathbf{v})\times\mathbf{n}\Bigg). \qquad \text{(A.12)}
\end{aligned}$$

Benutzt man nun einige Fakten über Vektoren (dass $\mathbf{n}\cdot\mathbf{v} = \mathbf{v}\cdot\mathbf{n}$, $\mathbf{n}\times\mathbf{v} = -\mathbf{v}\times\mathbf{n}$ und $(\mathbf{n}\times\mathbf{v})\cdot\mathbf{n} = 0$ ist) und bezieht die folgenden trigonometrischen Identitäten mit ein:

$$\begin{aligned}
\cos\theta &= \cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2} \\
\sin\theta &= 2\sin\frac{\theta}{2}\cos\frac{\theta}{2} \\
\cos\theta &= 1 - 2\sin^2\frac{\theta}{2},
\end{aligned}$$

so ergibt sich für (A.12):

$$\begin{aligned}
\mathbf{v}_{rot} &= \left(0, \sin^2\frac{\theta}{2}(\mathbf{n}\cdot\mathbf{v})\mathbf{n} + \cos^2\frac{\theta}{2}\mathbf{v} + 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}(\mathbf{n}\times\mathbf{v}) - \sin^2\frac{\theta}{2}(\mathbf{v} - (\mathbf{n}\cdot\mathbf{v})\mathbf{n}\right) \\
&= \left(0, 2\sin^2\frac{\theta}{2}(\mathbf{n}\cdot\mathbf{v})\mathbf{n} + \left(\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2}\right)\mathbf{v} + 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}(\mathbf{n}\times\mathbf{v})\right) \\
&= (0, (1-\cos\theta)(\mathbf{n}\cdot\mathbf{v})\mathbf{n} + \cos\theta\mathbf{v} + \sin\theta(\mathbf{n}\times\mathbf{v})) \\
&= (0, (\mathbf{n}\cdot\mathbf{v})\mathbf{n} + \cos\theta(\mathbf{v} - (\mathbf{n}\cdot\mathbf{v})\mathbf{n}) + \sin\theta(\mathbf{n}\times\mathbf{v})). \qquad \text{(A.13)}
\end{aligned}$$

Die Formel (A.13) wird nun durch eine grafische Herleitung bewiesen. Die linke Seite der Abbildung A.1 zeigt einen Vektor $\mathbf{v}$, der um den Winkel $\theta$ rotiert wird. Es sei $\mathbf{n}$ ein Einheitsvektor. Die Vektoren $\mathbf{n}$ und $\mathbf{v}$ spannen eine Ebene auf. Auf dieser Ebene liegen die Vektoren $\mathbf{v}_1 = (\mathbf{n}\cdot\mathbf{v})\mathbf{n}$ und $\mathbf{v}_2 = \mathbf{v} - (\mathbf{n}\cdot\mathbf{v})\mathbf{n}$. Der Vektor $\mathbf{v}_3$ steht senkrecht zu dieser Ebene und hat die gleiche Länge wie Vektor $\mathbf{v}_2$. Der Vektor lässt sich berechnen durch $\mathbf{v}_3 = \mathbf{n}\times\mathbf{v}_2 = \mathbf{n}\times\mathbf{v}$, da $\mathbf{n}$ ein Einheitsvektor ist und $\mathbf{v}_2 \perp \mathbf{n}$ ist.

Damit lässt sich der Vektor $\mathbf{v}_{rot}$ schreiben als

$$\begin{aligned}
\mathbf{v}_{rot} &= \mathbf{v}_1 + \cos\theta\mathbf{v}_2 + \sin\theta\mathbf{v}_3 \\
&= (\mathbf{n}\cdot\mathbf{v})\mathbf{n} + \cos\theta(\mathbf{v} - (\mathbf{n}\cdot\mathbf{v})\mathbf{n}) + \sin\theta(\mathbf{n}\times\mathbf{v}). \qquad \text{(A.14)}
\end{aligned}$$

Wie man nun sieht entspricht die Formel (A.14) genau dem Quaternion $\dot{v}$ (A.13). Dies zeigt, dass die angegebene Beschreibung der Rotation mittels des Einheitsquaternions (3.2),(3.3),(3.4),(3.5) tatsächlich eine Rotation um einen gegeben Vektor darstellt.

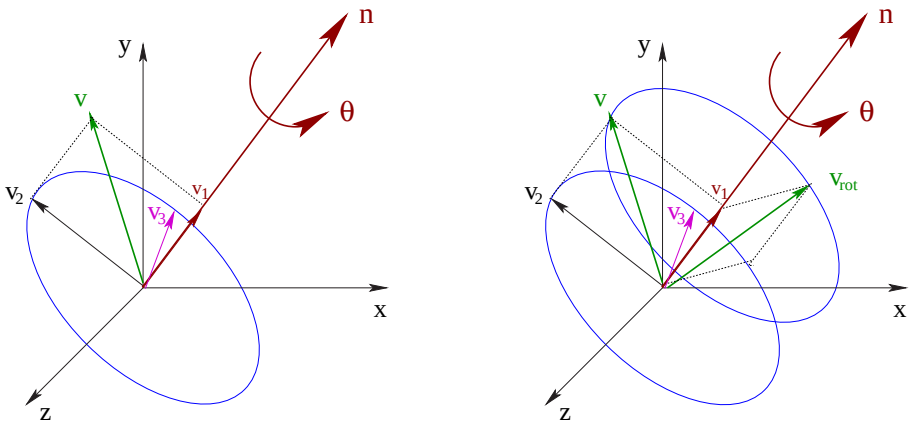


**Abbildung A.1:** Grafische Herleitung des rotierten Vektors $\mathbf{v}_{rot}$

## A.2 Berechnung der optimalen Rotation

### A.2.1 Die zu minimierende Fehlerfunktion

Nach dem Einsetzen der Gleichungen (3.11), (3.12), (3.13) und (3.14) in die Fehlerfunktion

$$E(\mathbf{R},\mathbf{t}) = \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\,||\mathbf{m}_i - (\mathbf{R}\mathbf{d}_j + \mathbf{t})||^2$$

ergibt sich

$$\begin{aligned} E(\mathbf{R},\mathbf{t}) &= \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}||\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_j - \underbrace{(\mathbf{t} - \mathbf{c}_m + \mathbf{R}\mathbf{c}_d)}_{=\tilde{\mathbf{t}}}||^2 \\ &= \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\left|\left|\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_j\right|\right|^2 \qquad \text{(A.15)} \\ &\quad -2\tilde{\mathbf{t}}\cdot\sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\left(\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_j\right) \qquad \text{(A.16)} \\ &\quad +\sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\left|\left|\tilde{\mathbf{t}}\right|\right|^2. \qquad \text{(A.17)} \end{aligned}$$

Um die obige Summe zu minimieren, müssen alle Terme minimiert werden. Der zweite Term (A.16) ist Null, da sich die Werte auf den Schwerpunkt beziehen. Der dritte Teil (A.17) hat für $\tilde{\mathbf{t}} = \mathbf{0}$ bzw. $\mathbf{t} = \mathbf{c}_m - \mathbf{R}\mathbf{c}_d$ ein Minimum. Folglich bleibt der erste Teil (A.15) übrig und die neue Fehlerfunktion lautet:

$$E(\mathbf{R},\mathbf{t}) = \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\left|\left|\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_j\right|\right|^2. \qquad \text{(A.18)}$$

### A.2.2 Rotationsbestimmung

Die neue Fehlerfunktion (A.18) muss nun minimiert werden. Da eine Rotation längenerhaltend ist, gilt immer $||\mathbf{R}\mathbf{d}'_j||^2 = ||\mathbf{d}'_j||^2$. Mit Hilfe dieser Eigenschaft wird die Fehlerfunktion erweitert zu

$$E(\mathbf{R},\mathbf{t}) = \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\left|\left|\mathbf{m}'_j\right|\right|^2 - 2\sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\mathbf{m}'_i \cdot \mathbf{R}\mathbf{d}'_j + \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\left|\left|\mathbf{d}'_j\right|\right|^2.$$

Wie man leicht sieht, geht die Rotation nur in den mittleren Term ein. Um die obige Gleichung zu minimieren, genügt es also, den Ausdruck

$$\sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\mathbf{m}'_i \cdot \mathbf{R}\mathbf{d}'_j \tag{A.19}$$

zu maximieren. An dieser Stelle der Herleitung kommen die Quaternionen zum Einsatz. Die Bestimmung der Rotationsmatrix $\mathbf{R}$, die (A.19) maximiert, kann aufgefasst werden als das Finden des Einheitsquaternions $\dot{q}$, das den Ausdruck

$$\sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\dot{m}'_i \cdot (\dot{q}\dot{d}'_j\dot{q}^*) = \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}(\dot{q}\dot{m}'_i) \cdot (\dot{d}'_j\dot{q})$$

maximiert. Dabei wurde die Rechenregel (A.6) angewendet. Seien nun $\mathbf{M}_i$ und $\bar{\mathbf{D}}_j$ die Matrizen zu den Quaternionen $\dot{m}'_i$ und $\dot{d}_j$ analog zu (A.7) und (A.8). Nach (A.9), (A.10) und (A.11) wird die zu maximierende Summe

$$\sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}(\bar{\mathbf{M}}_i\dot{q}) \cdot (\mathbf{D}_j\dot{q})$$

bzw.

$$\sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\dot{q}^T\bar{\mathbf{M}}_i\mathbf{D}_j^T\dot{q} \;=\; \dot{q}^T\left(\sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\bar{\mathbf{M}}_i\mathbf{D}_j^T\right)\dot{q}.$$

Schreibt man nun die Matrizen für $\mathbf{M}_i$ und $\mathbf{D}_j$, ergibt eine Rechnung:

$$\begin{aligned}
& \dot{q}^T\left(\sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\bar{\mathbf{M}}_i\mathbf{D}_j^T\right)\dot{q} \\
= \;& \dot{q}^T\left(\sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{i,j}\begin{pmatrix} 0 & -m_{ix} & -m_{iy} & -m_{iz} \\ m_{ix} & 0 & m_{iz} & -m_{iy} \\ m_{iy} & -m_{iz} & 0 & m_{ix} \\ m_{iz} & m_{iy} & -m_{ix} & 0 \end{pmatrix}\begin{pmatrix} 0 & -d_{jx} & -d_{jy} & -d_{jz} \\ d_{jx} & 0 & -d_{jz} & d_{jy} \\ d_{jy} & d_{jz} & 0 & -d_{jx} \\ d_{jz} & -d_{jy} & d_{jx} & 0 \end{pmatrix}^T\right)\dot{q} \\
= \;& \dot{q}^T\mathbf{N}\dot{q}.
\end{aligned} \tag{A.20}$$

Die in Kapitel 3.3.3 (vgl. (3.16)) definierte Matrix $\mathbf{N}$ entsteht bei der Multiplikation. Folgender Satz zeigt, wie das Einheitsquaternion, das (A.7) maximiert, bestimmt wird:

**Satz 7.** Das Einheitsquaternion $\dot{q}$ (es gilt $\dot{q} \cdot \dot{q} = 1$), das den Term $\dot{q}^T \mathbf{N} \dot{q}$ (A.20) maximiert, ist der Eigenvektor zu dem größten positiven Eigenwert der Matrix $\mathbf{N}$ [40].

**Beweis:** Die symmetrische Matrix $\mathbf{N}$ ist eine $4 \times 4$ Matrix. Das bedeutet, dass $\mathbf{N}$ vier Eigenwerte besitzt $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$. Zu diesen Eigenwerten können vier Eigenvektoren $(\dot{e}_1, \dot{e}_2, \dot{e}_3, \dot{e}_4)$ konstruiert werden, so dass gilt

$$\mathbf{N}\dot{e}_i = \lambda_i \dot{e}_i \qquad \text{für} \quad i = 1, 2, 3, 4.$$

Die Eigenvektoren spannen einen vier-dimensionalen Vektorraum auf (sie sind linear unabhängig); also kann ein beliebiges Quaternion $\dot{q}$ als Linearkombination

$$\dot{q} = \alpha_1 \dot{e}_1 + \alpha_2 \dot{e}_2 + \alpha_3 \dot{e}_3 + \alpha_3 \dot{e}_3$$

dargestellt werden. Da Eigenvektoren orthogonal sind, gilt:

$$\dot{q} \cdot \dot{q} = \alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_3^2.$$

Dies muss gleich 1 sein, da nach einem Einheitsquaternion $\dot{q}$ gesucht wird. Weiterhin gilt

$$\mathbf{N}\dot{q} = \alpha_1 \lambda_1 \dot{e}_1 + \alpha_2 \lambda_2 \dot{e}_2 + \alpha_3 \lambda_3 \dot{e}_3 + \alpha_4 \lambda_4 \dot{e}_4,$$

da $\dot{e}_1, \dot{e}_2, \dot{e}_3, \dot{e}_4$ Eigenvektoren von $\mathbf{N}$ sind. Daraus lässt sich schließen, dass

$$\dot{q}^T \mathbf{N} \dot{q} = \dot{q} \cdot (\mathbf{N}\dot{q}) = \alpha_1^2 \lambda_1 + \alpha_2^2 \lambda_2 + \alpha_3^2 \lambda_3 + \alpha_4^2 \lambda_4$$

gilt. Angenommen die Eigenwerte seien der Größe nach sortiert, d.h. $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_4$. Dann folgt daraus die Ungleichung

$$\dot{q}^T \mathbf{N} \dot{q} \leq \alpha_1^2 \lambda_1 + \alpha_2^2 \lambda_1 + \alpha_3^2 \lambda_1 + \alpha_4^2 \lambda_1 = \lambda_1$$

und es ist gezeigt, dass die quadratische Form niemals größer als der größte Eigenwert sein kann. Bei der Wahl von $\alpha_1 = 1$ und $\alpha_2 = \alpha_3 = \alpha_4 = 0$ wird das Maximum erreicht und das gesuchte Einheitsquaternion ist $\dot{q} = \dot{e}_1$ [40]. □

## A.3 Triangulationen und Triangulationsgraphen von Polygonen

In diesem Abschnitt soll bewiesen werden, dass sich jedes Polygon $\mathcal{P}$ triangulieren lässt und nur 3 Farben benötigt werden, um den zu $\mathcal{P}$ gehörenden Triangulationsgraphen einzufärben. Einfärben bedeutet, Knoten des Triangulationsgraphs mit Farben zu beschriften. Die Knoten entsprechen den Eckpunkten des Polygons.

Beide Beweise bauen auf dem so genannten Zwei-Ohren-Theorem (engl.: *two-ear-theorem*) von G. Meister auf, das zunächst vorgestellt wird. Dazu ist eine Definition des Begriffs Ohr in diesem Zusammenhang notwendig. Ein Polygon $\mathcal{P}$ hat ein Ohr am Eckpunkt $v$, falls das Dreieck geformt aus $v$ und den benachbarten Eckpunkten vollständig in $\mathcal{P}$ liegt. Abbildung A.2 verdeutlicht die Definition.

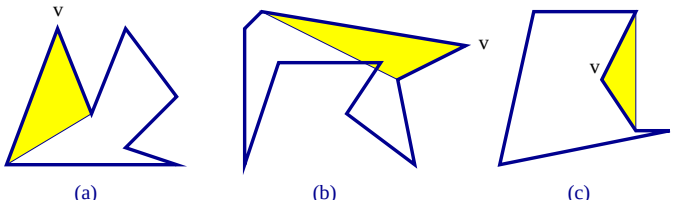


**Abbildung A.2:** Der Begriff Ohr eines Polygons: (a) Ohr; (b),(c) kein Ohr

**Satz 8 (G. Meisters Zwei-Ohren-Theorem).** Jedes Polygon $\mathcal{P}$ mit mehr als 4 Ecken hat mindestens zwei Ohren [55].

**Beweis:** Der Beweis ist eine Induktion über die Eckpunkte von $\mathcal{P}$.

Für den Induktionsanfang $n = 4$ gilt die Aussage: Ein Viereck kann immer in zwei Dreiecke als Ohren aufgeteilt werden. Sei nun $\mathcal{P}$ ein Polygon mit $n$ Ecken und $v_2$ konvexer Eckpunkt. Ein solcher Eckpunkt $v_2$ ist immer vorhanden. Seien $v_1$ und $v_3$ die benachbarten Punkte. Nun sind zwei Fälle zu unterscheiden: Entweder ist $v_1v_2v_3$ ein Ohr oder an $v_2$ gibt es kein Ohr.

Fall 1: $v_1v_2v_3$ ist ein Ohr. Durch das Entfernen des Ohrs entsteht ein neues Polygon mit $n-1$ Eckpunkten. Abbildung A.3 (a) verdeutlicht diesen Fall. Mit der Induktionsvoraussetzung folgt die Behauptung.

Fall 2: An $v_2$ gibt es kein Ohr. Das heißt, es gibt mindestens einen weiteren Eckpunkt des Polygons im Dreieck $v_1v_2v_3$. Nun wird eine Gerade parallel zu $v_1v_3$ verschoben, bis diese den letzten Eckpunkt des Polygons erreicht, der noch innerhalb des Dreiecks $v_1v_2v_3$ liegt. Sei dieser Eckpunkt $x$. Die Strecke $xv_2$ liegt innerhalb des Polygons und teilt es in zwei Teile auf ($\mathcal{P}_1$ und $\mathcal{P}_2$). Abbildung A.3 (b) verdeutlich diesen Sachverhalt.

Auch in diesem Fall besitzt $\mathcal{P}$ wiederum mindestens zwei Ohren. Es kann vorkommen, dass $\mathcal{P}_1$ oder $\mathcal{P}_2$ Dreiecke und somit Ohren sind. Ansonsten folgt per Induktion, dass $\mathcal{P}_1$ und $\mathcal{P}_2$ mindestens zwei Ohren haben. Das aus $\mathcal{P}_1$ und $\mathcal{P}_2$ zusammengesetzte Polygon $\mathcal{P}$ hat somit auch mindestens zwei Ohren.

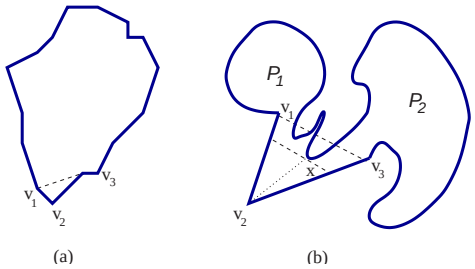


**Abbildung A.3:** Induktionsbeweis des Zwei-Ohren-Theorems. Die Linie $v_2x$ ist eine innere Diagonale.

□

**Satz 9 (Triangulations-Theorem).** Jedes Polygon $\mathcal{P}$ kann trianguliert werden [55].

**Beweis:** Das Zwei-Ohren-Theorem liefert eine Methode, um eine Triangulation von $\mathcal{P}$ zu finden. Dazu werden Ohren im Polygon gesucht und anschließend entfernt, so dass ein kleineres Polygon entsteht. Die dabei eingefügten Kanten sind die Diagonalen der gesuchten Triangulation. □

**Satz 10.** Jeder Triangulationsgraph eines Polygons $\mathcal{P}$ lässt sich mit 3 Farben einfärben [55].

**Beweis:** Auch hier liefert das Zwei-Ohren-Theorem die Vorarbeit zum Beweis. Nach dem Entfernen eines Polygonenohres wird der verbleibende Triangulationsgraph des Polygons induktiv mit 3 Farben koloriert. Fügt man die entfernte Ecke wieder hinzu, bekommt sie eine Farbe, die nicht auf der Diagonalen verwendet wurde. □

## A.4 Das Kunstausstellungsproblem ist NP-hart

Zu zeigen ist, dass sich das Problem 3-SAT in polynomiell beschränkter Zeit auf das Kunstausstellungsproblem (StarC) reduzieren lässt. Vorerst sei das Kunstausstellungsproblem vereinfacht: Wachen dürfen sich nur in den Eckpunkten des Polygons befinden. Diese Einschränkung kann am Ende aufgehoben werden. Zu jeder Instanz von 3-SAT, einer Konjunktion von Klauseln mit 3 Literalen, wird nun ein Polygon konstruiert [55].

**Literale.** Jedes Literal wird zu einem so genannten Literalmuster, wie in Abbildung A.4 dargestellt. Der Punkt $p$ ist von Punkt $f$ und Punkt $t$ aus sichtbar. Das gesamte Polygon ist so konstruiert, dass Punkt $p$ von keinem anderen Eckpunkt aus zu sehen ist. Am Punkt $t$ wird eine Wache aufgestellt, falls das Literal den Wert *wahr* annimmt, ansonsten erhält Punkt $f$ die Wache.

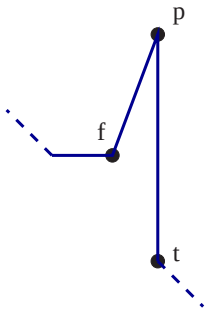


**Abbildung A.4:** Das Literalmuster: Der Punkt $p$ ist nur von den Punkten $t$ und $f$ aus sichtbar.

**Klauseln.** Die Klauseln bestehen aus drei Literalen, die durch so genannte Klauselzusammenführungen repräsentiert werden. Abbildung A.5 zeigt das Teilpolygon für eine Klausel. Literale werden so zu Klauseln vereinigt, dass mindestens ein Wachposten in einer $t_i$-Ecke $i \in \{1, 2, 3\}$ steht. Die korrespondierende Klausel ist dann erfüllt. In diesem Fall wird der markierte Bereich eingesehen.

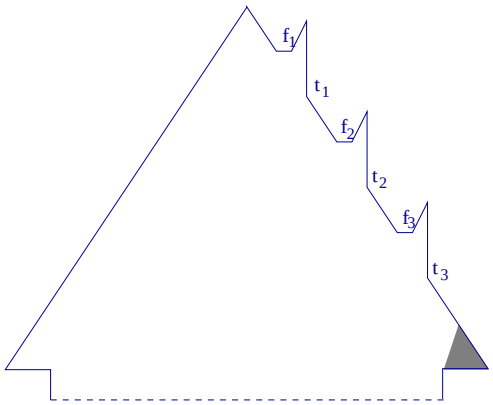


**Abbildung A.5:** Eine Klauselzusammenführung. Der markierte Bereich kann nur von $t_i$ $i = 1, 2, 3$ eingesehen werden.

**Variablenmuster.** Der Sinn der so genannten Variablenmuster ist es, die Konsistenz zwischen den Variablenbelegungen der einzelnen Klauseln zu erzwingen. Dies wird mit einem Variablenmuster erreicht, wie es Abbildung A.6 zeigt. Das Muster einer Variablen besteht aus zwei Schächten mit einer Ecke oben rechts, die mit $F$ beziehungsweise $T$ bezeichnet ist. Zusätzlich hat jeder Schacht $s$ Spitzen, wobei $s$ die Anzahl der Klauseln ist, in denen die Variable vorhanden ist. Einer der beiden Punkte $F$ oder $T$ muss einen Wachposten besitzen, um die Ecke bei $q$ einsehen zu können. Falls allen Variablen konsistente Wahrheitswerte zugewiesen wurden, sind keine weiteren Wachposten nötig, um das Variablenmuster einzusehen. Eine vollständige Konstruktion verdeutlicht diesen Sachverhalt.

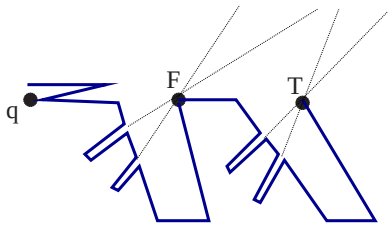


**Abbildung A.6:** Das Variablenmuster. Zwei Schächte mit zwei Spitzen. Der Punkt $q$ kann von $T$ oder $F$ gesehen werden.

**Eine vollständige Konstruktion.** Ein Beispiel für einen booleschen Ausdruck mit drei Variablen ($n = 3$) und zwei Klauseln ($m = 2$) zeigt Abbildung A.7. Die beiden Klauseln sind $(\mathsf{u}_1 \vee \bar{\mathsf{u}}_2 \vee \mathsf{u}_3)$ und $(\bar{\mathsf{u}}_1 \vee \bar{\mathsf{u}}_2 \vee \mathsf{u}_3)$. Jedes Variablenmuster hat vier Spitzen. Für jedes Literal einer Klauselzusammenführung, das die Variable benutzt, gibt es eine Spitze pro Schacht. Sei $\mathsf{u}$ die Variable mit den ausgezeichneten Punkten $T$ und $F$ und $\mathsf{l}$ ein Literal in $\{\mathsf{u}, \bar{\mathsf{u}}\}$ mit den Punkten $t$ und $f$. Die Spitze im linken Schacht des Variablenmuster für $u$ ist linear zu $F$ und $t$, falls $\mathsf{l} = \mathsf{u}$ gilt. Sie ist linear zu $F$ und $f$, falls $\mathsf{l} = \bar{\mathsf{u}}$. Die Spitze im rechten Schacht liegt auf einer Geraden mit $T$ und $f$, wenn $\mathsf{l} = \mathsf{u}$ und auf einer Geraden mit $T$ und $t$, wenn $\mathsf{l} = \bar{\mathsf{u}}$.

Die Konsequenz dieser Anordnung ist, dass ein in $F$ platzierter Wachposten alle Spitzen des linken Schachts und eine Wache in $T$ alle Spitzen im rechten Schacht sieht. Wie bereits erwähnt, wird entweder eine Wache in $F$ oder eine in $T$ benötigt, um den Punkt $q$ des Variablenmusters sehen zu können. Angenommen ein Wachposten wird an der Ecke $T$ platziert. Dieser sieht alle Spitzen im rechten Schacht. Weil aber alle Spitzen des linken Schachts mit $F$ und $t_{i_k}$ bzw. $f_{i_k}$ auf einer Gerade liegen, können diese Spitzen eingesehen werden, wenn bei allen $t_{i_k}$ und $f_{i_k}$ Wachen postiert werden. Die $t_{i_k}$ sind hierbei die $t$-Ecken für alle $\mathsf{u}$-Literale, die $f_{i_k}$ die $f$-Ecken der $\bar{\mathsf{u}}$-Literale. Falls $\mathsf{u}$ mit *wahr* belegt wird und die Variablenbelegung konsistent ist, muss nur eine Wache bei $T$ postiert werden. Falls $\mathsf{u}$ mit *falsch* belegt wird und die Variablenbelegung konsistent ist, wird eine Wache bei $F$ benötigt. In Abbildung A.7 wird $\mathsf{u}_1$ durch die Wache an $F_1$ auf *falsch* gesetzt. Die Spitzen im rechten Schacht des $\mathsf{u}_1$ Variablenmusters werden durch die Wachen an der $f$-Ecke des Literals $\mathsf{u}_1$ der ersten Klausel und der $t$-Ecke des Literals $\bar{\mathsf{u}}_1$ der zweiten Klausel eingesehen [55].

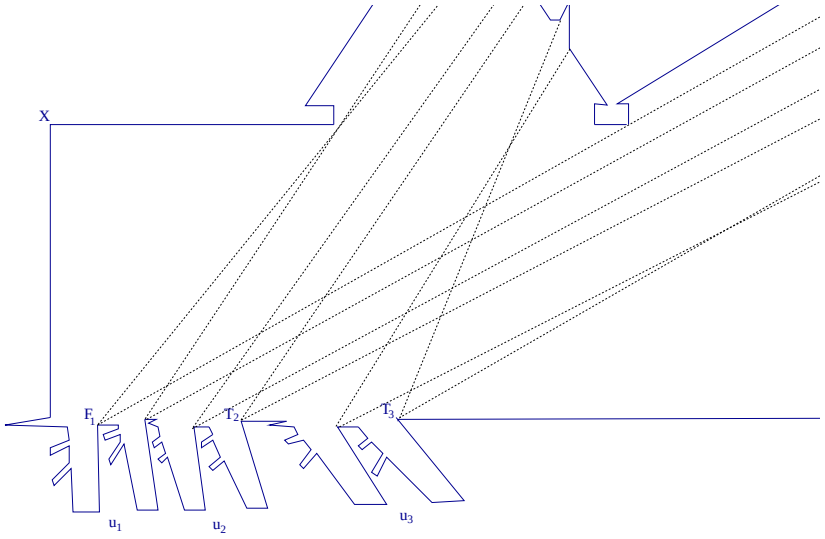


**Abbildung A.7:** Das vollständige Polygon für die 3-SAT Formel $(\mathsf{u}_1 \vee \bar{\mathsf{u}}_2 \vee \mathsf{u}_3) \wedge (\bar{\mathsf{u}}_1 \vee \bar{\mathsf{u}}_2 \vee \mathsf{u}_3)$

Die gesamte Anzahl der benötigten Wachposten beträgt $3m + n + 1$. Folgende Aussage muss bewiesen werden: Eine gegebene 3-SAT Formel ist genau dann erfüllbar, wenn sich das konstruierte Polygon mit $K = 3m + n + 1$ Wachposten beschützen lässt.

Kann die boolesche Formel erfüllt werden, existiert eine Belegung der Variablen mit Wahrheitswerten, so dass jede Klausel *wahr* wird. Die Platzierung von Wachposten an den entsprechenden $t$- und $f$-Ecken garantiert, dass die Klauselzusammenführungen vollständig eingesehen werden, da mindestens eine Wache an einer $t$-Ecke postiert ist. Mit Hilfe der oben präsentierten Argumente werden alle Spitzen eingesehen, falls Variablen konsistent belegt werden. Schließlich muss

man noch eine Wache bei $x$ postieren, um in alle Schächte zu schauen. Daher wird das Polygon mit $K$ Wachen vollständig bewacht.

Angenommen $K$ Wachen lassen sich so aufstellen, dass sie das Polygon vollständig überblicken können. Eine Wache muss bei $x$ platziert werden, da sonst $K$ Wachpersonen nicht ausreichen würde. Die übrigen $3m + n$ werden benötigt, um die Spitzen im Polygon einzusehen. Jedes Literalmuster braucht eine Wache an der $f$- oder $t$-Ecke, jedes Variablemuster eine Wache bei $T$ oder $F$. Jede Klauselzusammenführung wird vollständig überwacht, wenn eine Wache an einer $t$-Ecke steht. Dies impliziert, die entsprechende Klausel ist erfüllt. Die Variablemuster werden – wie beschrieben – nur von je einer Wache bewacht, falls die Literale, die die zugehörige Variable benutzen, konsistent belegt werden. Die vorgestellte Wachpostenaufstellung gibt eine konsistente Belegung für das zugehörige 3-SAT Problem.

Damit ist der Satz für Wachen, die an Eckpunkten stehen, bewiesen. Aggarwal erhielt das gleiche Ergebnis für Wachposten, die sich überall aufstellen dürfen [12, 55]. Dabei wird das Polygon so abgewandelt, dass die ausgezeichneten Eckpunkte zu inneren Punkten des Polygons werden [12].

□

## A.5 Kinematisches Kontrollgesetz zur Steuerung des Roboters

Die Lösung für das Kontrollproblem aus Abschnitt 4.4 soll hergeleitet werden. Setzt man den Ansatz $u = \gamma e$ mit $\gamma > 0$ (Formel (4.7)) in die Zustandsvariablen

$$\begin{aligned} \dot{e} &= -u\cos\alpha \\ \dot{\alpha} &= u\left(c - \frac{\sin\alpha}{e}\right) \\ \dot{\theta} &= u\frac{\sin\alpha}{e} \end{aligned}$$

ein, so ergibt sich

$$\begin{aligned} \dot{e} &= -\gamma e\cos\alpha \\ \dot{\alpha} &= \gamma e\left(c - \frac{\sin\alpha}{e}\right) \\ \dot{\theta} &= \gamma\sin\alpha. \end{aligned} \tag{A.21}$$

Das obige gewöhnliche Differentialgleichungssystem ist über die Position $(e, \alpha, \theta)$ rückgekoppelt und wird durch den Roboter gelöst. Die Fahrt des Roboters entspricht einer Simulation und damit auch der Lösung des Systems. Die verbleibende Aufgabe besteht darin, für die Konstante $c$, die zu fahrende Krümmung, einen Term zu finden, so dass garantiert werden kann, dass die Null-Lösung $(0, 0, 0)$ stabil ist. Zusätzlich soll sich das System dieser Konfiguration kontinuierlich annähern [46]. Eine solche Stabilisierung von dynamischen Systemen durch Rückkopplung stellt eine der wesentlichen Aufgaben der Steuerungstheorie dar.

Die Null-Lösung heißt stabil, wenn es zu jedem $\boldsymbol{\varepsilon} > \mathbf{0}$ ein $\boldsymbol{\delta} > \mathbf{0}$ gibt, das nicht von der Zeit $t$, sondern nur von $\boldsymbol{\varepsilon}$ abhängt, so dass für jede Anfangskonfiguration kleiner $\boldsymbol{\delta}$ die Lösung der

Differentialgleichung auf die ganze positive Halbachse $t > 0$ fortsetzbar und für alle $t > 0$ diese Lösung kleiner als $\varepsilon$ ist (vgl. Abbildung A.8) [6].

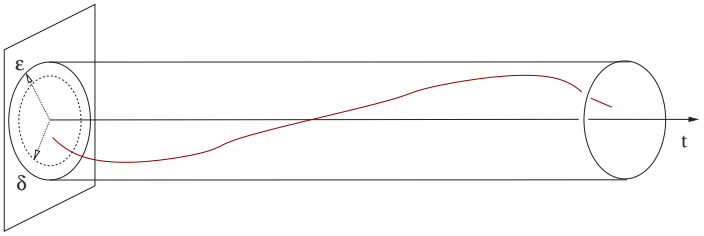


**Abbildung A.8:** Stabile Gleichgewichtslagen.

Ist das zu lösende Differentialgleichungssystem linear, beispielsweise $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, gibt folgender Satz Auskunft über die Stabilität der Null-Lösung.

**Satz 11.** Sei $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ eine lineare Differentialgleichung. Sind alle Eigenwerte $\lambda$ des Operators $\mathbf{A}$ in der linken Halbebene, also $Re(\lambda) < 0$, dann ist die Null-Lösung stabil und alle Lösungen konvergieren gegen diese. Man nennt diese Eigenschaft in diesem Zusammenhang auch asymptotisch stabil (Stabilität zuzüglich Konvergenz).

**Beweis:** Siehe [6]. □

Offensichtlich ist das Differentialgleichungssystem (A.21) nicht linear. Im nichtlinearen Fall kann die Methode von Lyapunov für den Stabilitätsnachweis verwendet werden. Die Lyapunov Funktion ist eine skalare Funktion $V$, definiert über einer verbundenen Region $R$, die den Nullpunkt enthält. Sie ist darüber hinaus positiv definit, d.h. es gilt $V(\mathbf{0}) = \mathbf{0}$ und $V(\mathbf{x}) > \mathbf{0}$ für alle $\mathbf{x} \neq \mathbf{0},\ \mathbf{x} \in R$. Weiterhin sind ihre ersten partiellen Ableitungen an jedem Punkt von $R$ stetig. Die Ableitung von $V$ für ein beliebiges System $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ wird mit $\dot{V}(\mathbf{x})$ bezeichnet und ist definiert als das Skalarporodukt

$$\dot{V}(\mathbf{x}) = \nabla V(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x})$$

[77]. Die Existenz einer Lyapunov Funktion, für die $\dot{V}(\mathbf{x}) \leq \mathbf{0},\ \mathbf{x} \in R$ gilt, garantiert die Stabiltät der Null-Lösung. Handelt es sich bei $\dot{V}$ sogar um eine negativ definite Funktion, also $\dot{V}(\mathbf{0}) = \mathbf{0}$ und $\dot{V}(\mathbf{x}) < \mathbf{0}$ für alle $\mathbf{x} \neq \mathbf{0},\ \mathbf{x} \in R$, dann ist die Null-Lösung sogar asymptotisch stabil [77].

In physikalischen Systemen lässt sich die Stabilität durch die Betrachtung der Gesamtenergie nachweisen. Bleibt diese Energie gleich oder sinkt sie, ist die Stabilität sofort klar. Die Gesamtenergiefunktion ist dann die Lyapunov Funktion [37]. Die Bestimmung einer Energiefunktion erfordert in der Regel die Lösung des vorliegenden Differentialgleichungssystems. Ist nur die Stabilität von Interesse, versucht man eine Lyapunov Funktion zu raten.

Die nun folgende Rechnung zeigt, dass eine Lyapunov Kandidatenfunktion bei geeigneter Wahl von $c$ die Stabilität der Lösung von (A.21) garantiert. Weiterhin werden die auftretenden Grenzwerte nachgewiesen und unter anderem folgender kleiner Hilfssatz benutzt:

**Satz 12 (Barbalats Lemma).** Sei $f : \mathbb{R}^+ \to \mathbb{R}$ eine gleichmäßig stetige Funktion. Wenn $\int_0^\infty f(s)\,ds < \infty$ gilt, folgt $\lim_{t\to\infty} f(t) = 0$.

Die quadratische Lyapunov Kandidatenfunktion

$$V = \frac{1}{2}\left(\alpha^2 + h\theta^2\right) \qquad \text{mit} \quad h > 0$$

hat die Ableitung nach der Zeit

$$\dot{V} = \alpha\dot{\alpha} + h\theta\dot{\theta} = \gamma(\alpha \sin\alpha + h\theta \sin\alpha - \alpha e c).$$

Die letzte Gleichung suggeriert die Wahl von $c$ als

$$c = \frac{\sin\alpha}{e} + h\frac{\theta}{e}\frac{\sin\alpha}{\alpha} + \beta\frac{\alpha}{e} \qquad \text{mit} \quad \beta, h > 1 \tag{A.22}$$

[46], wodurch die Ableitung der Lyapunov Kandidatenfunktion $V$ zu

$$\dot{V} = -\gamma\beta\alpha^2 \leq 0 \tag{A.23}$$

wird. Da $V$ positiv definit ist und radial unbegrenzt, impliziert Gleichung (A.23), dass $V$ ein positives endliches Limit besitzt [46]. Daher sei:

$$\begin{aligned} \lim_{t\to\infty} \alpha &= \bar{\alpha} \\ \lim_{t\to\infty} \theta &= \bar{\theta}. \end{aligned}$$

Zusätzlich gilt, dass $\dot{V}$ gleichmäßig stetig ist ($\ddot{V} = -2\gamma\beta\alpha\dot{\alpha}$ ist begrenzt). Dies impliziert durch Barbalats Lemma, dass $\dot{V}$ gegen Null strebt, also folgt $\bar{\alpha} = 0$. Die Substituierung der Gleichung (A.22) in (A.21) ergibt:

$$\begin{aligned} \dot{e} &= -\gamma e \cos\alpha \\ \dot{\alpha} &= \gamma\left(\beta\alpha - h\theta\frac{\sin\alpha}{\alpha}\right) \\ \dot{\theta} &= \gamma \sin\alpha. \end{aligned} \tag{A.24}$$

Da $\alpha$ gegen Null und $\theta$ gegen $\bar{\theta}$ strebt und weil $\dot{\alpha}$ stetig ist, folgt mittels Barblats Lemma, dass das Limit

$$\lim_{t\to\infty} \dot{\alpha} = -\gamma h\bar{\theta} = 0$$

ist. Daher muss auch der Grenzwert von $\theta$ Null sein ($\bar{\theta} = 0$) [46]. Die letzte der Gleichungen in (A.24) impliziert, dass auch $\dot{\theta}$ asymptotisch gegen Null strebt. Es ergibt sich folgendes Ergebnis:

$$\begin{aligned} \alpha \to 0 \quad ; \quad \dot{\alpha} \to 0 \\ \theta \to 0 \quad ; \quad \dot{\theta} \to 0. \end{aligned}$$

Wenn $t \to \infty$, existiert ein $t^*$ von dem $\cos\alpha > 0$ und $e$ asymptotisch exponentiell gegen Null konvergieren:

$$\dot{e} \to -\gamma e \;\rightsquigarrow\; e \to 0.$$

Durch diese Überlegungen ist das Kontrollgesetz (4.7) und (A.22) nachgewiesen. Das Verhalten hängt von den Parametern $\gamma, \beta$ und $h$ ab. $u$ hat einen endlichen Wert, solange $e$ und $\gamma$ endlich sind. Auch die Winkelgeschwindigkeit $\omega$ muss begrenzt sein. Dazu analysiert man $\lim_{(e,\alpha,\theta)\to(0,0,0)} c$. Wenn $\alpha$ gegen Null geht, können die Zustandsgleichungen aus (A.24) durch folgendes lineare System approximiert werden:

$$\begin{aligned} \dot{e} &= -\gamma e \\ \begin{pmatrix} \dot{\alpha} \\ \dot{\theta} \end{pmatrix} &= \begin{bmatrix} -\gamma\beta & -h\gamma \\ \gamma & 0 \end{bmatrix} \begin{pmatrix} \alpha \\ \theta \end{pmatrix}. \end{aligned} \tag{A.25}$$

Gleichung A.22 wird in diesem Grenzfall zu:

$$c = \frac{\alpha}{e}(1+\beta) + h\frac{\theta}{e}.$$

Die Zielpose $(0,0,0)$ soll auf einer geraden Linie erreicht werden, d.h. dass die Krümmung $c$ gegen Null konvergieren muss. Dies gelingt, wenn der Realteil des dominanten Pols der Gleichung (A.25) größer als $\gamma$ ist (vgl. Satz 11) [46]. Durch Berechnung der Eigenwerte der Matrix des Systems in (A.25)

$$\lambda_{\pm} = \frac{\gamma}{2}\left(-\beta \pm \sqrt{\beta^2 - 4h}\right)$$

und der Bedingung $|Re(\lambda_+)| > \gamma$ ergeben sich die Forderungen für die Konstanten

$$h > 1 \quad ; \quad 2 < \beta < h + 1 \tag{A.26}$$

[46]. Werden $h$ und $\beta$ so gewählt, wie Gleichung (A.26) vorschreibt, ist $c$ während der ganzen Trajektorie begrenzt und konvergiert asymptotisch gegen Null.

Im oben hergeleiteten Kontrollgesetz für $u$ (4.7) und $c$ (4.8) ist die Geschwindigkeit abhängig vom Abstand. Da jedes Fahrzeug eine maximale Geschwindigkeit $\bar{u}$ besitzt, tritt in vielen Applikationen eine Sättigung auf, falls der Zielpunkt zu weit entfernt ist. Demnach ist

$$u = \gamma e \mathrm{sat}(\gamma, e, \bar{u}) \qquad \text{mit} \quad \gamma > 0 \tag{A.27}$$

wobei

$$\mathrm{sat}(x,y) = \begin{cases} 1 & \forall x < y \\ \frac{y}{x} & \forall x \geq y \end{cases} \qquad \forall x, y > 0$$

eine Saturierungsfunktion ist. In dieser Situation wird aus (A.22) und (A.27) die Zustandsgleichung (A.24) zu:

$$\begin{aligned} \dot{e} &= -\gamma e \mathrm{sat}(\gamma, e, \bar{u}) \cos\alpha \\ \dot{\alpha} &= \gamma \mathrm{sat}(\gamma, e, \bar{u}) \left(\beta\alpha - h\theta\frac{\sin\alpha}{\alpha}\right) \\ \dot{\theta} &= \gamma \mathrm{sat}(\gamma, e, \bar{u}) \sin\alpha. \end{aligned}$$

Eine Rechnung ergibt die gleichen Lösungen für diesen Fall; allerdings muss die Konstante $\beta$ die Bedingung

$$\beta \geq \frac{4e_0}{3\pi^2}$$

erfüllen [46].

# Anhang B

# Datenstrukturen und Algorithmen

## B.1 Mehrdimensionale binäre Bäume

Der kd–Baum wird durch folgnde Klasse repräsentiert.

```
1  class KDtree {
2    public:
3      int npts;       // gleich 0, für inneren Knoten. ungleich 0 für Blätter
4      union {
5         struct {
6           double center[3];
7           double dx, dy, dz, r2;
8           int splitaxis;
9           KDtree *child1, *child2;
10        } node;
11        struct {
12          double *p[8];
13        } leaf;
14     };
15     // Konstruktor, für den Aufbau des kd–Baum zuständig
16     KDtree(double **pts, int n);
17     // Destruktor
18     ~KDtree()
19     double *FindClosest(double *_p, double maxdist2);
20   private:
21     static double *closest;
22     static double closest_d2;
23     static double *p;
24     void _FindClosest();
25 };
```

### B.1.1 Aufbau eines $k$d–Baumes

```
KDtree::KDtree(double **pts, int n) {
  if (n <= 8) {     // Blätter Knoten – Kopieren der Daten
    npts = n;
    memcpy(leaf.p, pts, n * sizeof(double *));
    return;
  }
  npts = 0;         // innere Knoten
  // Bestimmung der Bounding Box
  // node.center, node.dx, node.dy, node.dz und node.r2
  ...
  // Bestimmung der längsten Achse
  if (node.dx > node.dy)
    if (node.dx > node.dz) node.splitaxis = 0;
    else node.splitaxis = 2;
   else
    if (node.dy > node.dz) node.splitaxis = 1;
    else node.splitaxis = 2;
  // Aufteilung der Daten auf Felder **left und **right für die Nachfolgerknoten
  // anhand des Wertes splitval
  double splitval = node.center[node.splitaxis];
  double **left, **right;
  ...
  // Aufbau der Unterbäume
  node.child1 = new KDtree(pts, left-pts);
  node.child2 = new KDtree(left, n-(left-pts));
}
```

### B.1.2 Suche innerhalb eines $k$d–Baumes

```
// Suche nach dem nächsten Punkt zu _p, der den Abstand maxdist2 nicht übersteigt
double *KDtree::FindClosest(double *_p, double maxdist2) {
  // Intitialisierung
  closest = NULL; closest_d2 = maxdist2; p = _p;
  // Start der eigentlichen Suche
  _FindClosest();
  return closest;
}

void KDtree::_FindClosest() {
  // Blätter Knoten
  if (npts) {
    for (int i=0; i < npts; i++){
      double myd2 = Dist2(p, leaf.p[i]);
      if ((myd2 < closest_d2)) {closest_d2 = myd2; closest = leaf.p[i];}
    }
    return;
  }
  // schneller Test, ob in dem aktuellen Unterbaum weitergesucht werden muss
  double approx_dist_bbox = max(max(fabs(p[0]-node.center[0])-node.dx,
```

```
21                                          fabs(p[1]-node.center[1])-node.dy),
22                                      fabs(p[2]-node.center[2])-node.dz);
23   if (approx_dist_bbox >= 0 && sqr(approx_dist_bbox) >= closest_d2)
24     return;
25   // Rekursiver Fälle
26   double myd = node.center[node.splitaxis] - p[node.splitaxis];
27   if (myd >= 0.0f) {
28     node.child1->_FindClosest();
29     if (sqr(myd) < closest_d2) node.child2->_FindClosest();
30   } else {
31     node.child2->_FindClosest();
32     if (sqr(myd) < closest_d2) node.child1->_FindClosest();
33   }
34 }
```

## B.2 Pseudocode für simultanes Scanmatching

Das sequentielle Scanmatching wird mittels folgendem Algorithmus ausgeführt:

```
1  while not empty scan_set       // scan_set = Menge aller zu registrierenden 3D–Scans
2     curr = next(scan_set)
3     active_set.add(curr)
4     scan_set.remove(curr)
5     if (last) align(curr, last)
6     queue.push(curr);           // Initialisierung mit aktuellem Scan
7     while not empty queue
8        curr    = queue.pop()
9        nbors   = active_set.neighbors(curr)
10       relative_change = align(curr, nbors)
11       if (relative_change > tolerance)
12          queue.merge(nbors)
13    last = curr
```

## B.3 Der Controller des Roboters

Die Berechnung der Geschwindigkeit und des Drehwinkels für den Motorkontroler des Roboters geschieht mit folgender Funktion:

```
1  int CalcNextVelocities(double u_max, double x, double y,
2                         double phi,
3                         double *u, double *omega)
4  {
5    double c, theta;
6    double alpha;
7    double e;
8    // h > 1; 2 < beta < h + 1 nach Lyaponv Kriterium
9    double gamma = 1.0, beta = 2.9, h = 2.0;
10   double epsilon = 0.00001;
```

```
11
12    e = sqrt(x*x + y*y);                  // Entfernung zum Ziel
13    if (e > epsilon) {
14      theta = atan2(-y,-x);
15      if (theta >  M_PI) theta -= 2.0 * M_PI;
16      if (theta < -M_PI) theta += 2.0 * M_PI;
17      alpha = theta - phi;
18      if (alpha >  M_PI) alpha -= 2.0 * M_PI;
19      if (alpha < -M_PI) alpha += 2.0 * M_PI;
20
21      *u = gamma * e;
22      if (*u >= u_max) *u = u_max;   // maximale Geschwindigeit des Roboters
23
24      if (fabs(alpha) > epsilon) {
25        c = (sin(alpha) + (h * theta * sin(alpha) / alpha + beta * alpha))
26            / e;
27      }
28      else {
29        // Orientierung ist in Richtung Ziel
30        c = (alpha + alpha * beta + h * theta) / e;
31      }
32      *omega = c * *u;                  // Berechnung Drehwinkel
33    }
34    else {
35      *u = 0.0;                         // Ziel erreicht
36      *omega = 0.0;
37    }
38    return 0;
39  }
```

Die Umsetzung der Werte für die Geschwindigkeit $u$ und die Winkelgeschwindigkeit $\omega$ geschieht durch folgende Anweisungen:

```
1   leftspeed  = (u - (fabs(omega)/4.0*0.362)) + omega/4.0*0.362;
2   rightspeed = (u - (fabs(omega)/4.0*0.362)) - omega/4.0*0.362;
3
4   scale_factor = 1.0;
5
6   if (fabs(leftspeed)  > u_max) scale_factor = fabs(u_max / leftspeed);
7   if (fabs(rightspeed) > u_max) scale_factor = fabs(u_max / rightspeed);
8   leftspeed  *= scale_factor;
9   rightspeed *= scale_factor;
```

# Literaturverzeichnis


[1] Sir William Rowan Hamilton (1805-1865). `http://www.maths.tcd.ie/pub/HistMath/People/Hamilton/`, 2000.

[2] Siemens 3D-Sensor. http://w4.siemens.de/fui/de/archiv/newworld/heft2_01/artikel03/, 2001.

[3] P. Allen, I. Stamos, A. Gueorguiev, E. Gold, and P. Blaer. AVENUE: Automated Site Modelling in Urban Environments. In *Proceedings of the Third International Conference on 3D Digital Imaging and Modeling (3DIM '01)*, Quebec City, Canada, May 2001.

[4] P. Allen and R. Yang. Registering, Integrating, and Building CAD Models from Range Data. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '98)*, Leuven, Belgium, May 1998.

[5] N. Amenta, S. Choi, and R. Kolluri. The Power Crust. *Communications of the ACM*, 35(7):56 – 63, 2001.

[6] V. Arnold. *Gewöhnliche Differentialgleichungen*. Springer Verlag, Berlin Heidelberg, 2001.

[7] J. Banta, Y. Zhieng, X. Wang, G. Zhang, M. Smith, and M. Abidi. A "Best–Next–View" Algorithm for Three–Dimensional Scene Reconstruction Using Range Images. In D. Casasent, editor, *Proceedings SPIE (Intelligent Robots and Computer Vision XIV: Algorithms), vol 2588*, pages 418 – 429, 1995.

[8] R. Benjemaa and F. Schmitt. Fast Global Registration of 3D Sampled Surfaces Using a Multi-Z-Buffer Technique. In *Proceedings IEEE International Conference on Recent Advances in 3D Digital Imaging and Modeling (3DIM '97)*, Ottawa, Canada, May 1997.

[9] M. Bern and P. Plassmann. Mesh generation. In J. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*. Elsevier Science, 2000.

[10] P. Besl and N. McKay. A method for Registration of 3–D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239 – 256, February 1992.

[11] G. Blais and D. Levine. Registration Multiview Range Data to Create 3D Computer Objects. *Pattern Analysis and Machine Intelligence*, 17(8):820 – 824, August 1995.

[12] S. Carlsson and B. Nilsson. Computing Vision Points in Polygons. *Algorithmica*, 24(1):50 – 75, 1999.

[13] Y. Chen and G. Medoni. Object Modelling by Registration of Multiple Range Images. In *Proceedings of the IEEE Conferenc on Robotics and Automation (ICRA '91)*, pages 2724 – 2729, Sacramento, CA, USA, April 1991.

[14] Th. Cormen, Ch. Leiserson, and R. Rivest. *Introduction to Algorithms*. McGraw–Hill Book Company, 1997.

[15] The Power Crust. `http://www.cs.utexas.edu/users/amenta/powercrust/`, 2002.

[16] S. Cunnington and A. Stoddart. N-View Point Set Registration: A Comparison. In *Proceedings of the 10th British Machine Vision Conference (BMVC '99)*, Nottingham, UK., 1999.

[17] B. Curless and M. Levoy. A Volumetric Method for Building Complex Models from Range Images. *Computer Graphics*, 30(Annual Conference Series):303 – 312, 1996.

[18] E. Dam, M. Koch, and M. Lillholm. Quaternion, Interpolation and Animation. Technical Report DIKU-TR-98/5, Department of Computer Science University of Copenhagen, Copenhagen, Denmark, 1998.

[19] D. Eggert, A. Fitzgibbon, and R. Fisher. Simultaneous Registration of Multiple Range Views Satisfying Global Consistency Constraints for Use In Reverse Engineering. *Computer Vision and Image Understanding*, 69:253 – 272, March 1998.

[20] Matrix FAQ. Version 2, `http://skal.planet-d.net/demo/matrixfaq.htm`, 1997.

[21] G. Fischer. *Lineare Algebra*. Friedrich Vieweg & Sohn Verglagsgesellschaft mbH, Braunschweig / Wiesbaden, 1995.

[22] AVENUE (Autonomous Vehicle for Exploration and Navigation in Urban Environments) Project. `http://www.cs.columbia.edu/robotics/projects/avenue/`, 2001.

[23] O. Forster. *Analysis 1*. Friedrich Vieweg & Sohn Verglagsgesellschaft mbH, Braunschweig / Wiesbaden, 1996.

[24] C. Früh and A. Zakhor. 3D Model Generation for Cities Using Aerial Photographs and Ground Level Laser Scans. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR '01)*, Kauai, Hawaii, USA, December 2001.

[25] C. Früh and A. Zakhor. Fast 3D Model Generation in Urban Environments. In *Proceedings of the 4th International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI '01)*, Baden Baden, Germany, August 2001.

[26] RIEGL Laser Measurement Systems GmbH. `http://www.riegl.co.at/`, 2001.

[27] L. Goncalves, R. Grupen, and A. Oliveira. A Control Architecture for Multi-modal Sensory Integration. In *Proceedings of the 9th IEEE International Conference on Computer Graphics and Image Processing (SIBGRAPH '98)*, Rio de Janeiro, RJ, Brazil, October 1998.

[28] H. Gonzales-Banos and J. Latombe. Planning Robot Motions for Range–Image Acqusition and Automatic 3D Model Construction. In *Proceedings of AAAI Fall Symposium*, Menlo Park, CA, USA, October 1998. AAAI Press.

[29] H. Gonzalez-Banos and J. Latombe. Robot Navigation for Automatic Model Construction Using Safe Regions. In D. Russ and S. Singh, editors, *Lecture Notes in Control and Information Sciences, 271, (Proceedings of International Symposium on Experimental Robotics (ISER '00))*, pages 405 – 415, Waikiki, HI, USA, December 2000. Springer.

[30] H. Gonzalez-Banos and J. Latombe. A Randomized Art-Gallery Algorithm for Sensor Placement. In *Proceedings of ACM Symposium on Computational Geometry (SoCG'01)*, Medford, MA, USA, June 2001.

[31] A. Gueorguiev, P. Allen, E. Gold, and P. Blaer. Design, Arcitecture and Control of a Mobile site-Modelling Robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '00)*, San Francisco, CA, USA, April 2000.

[32] D. Hähnel and W. Burgard. Probabilistic Matching for 3D Scan Registration. In *Proceedings of the second German conference on robotics (ROBOTIK '02)*, Ludwigsburg, Germany, June 2002.

[33] D. Hähnel, W. Burgard, and S. Thrun. Learning Compact 3D Models of Indoor and Outdoor Environments with a Mobile Robot. In *Proceedings of the fourth European workshop on advanced mobile robots (EUROBOT '01)*, Lund, Sweden, September 2001.

[34] W. Hamilton. On a new Species of Imaginary Quantities connected with a theory of Quaternions. In *Proceedings of the Royal Irish Academy*, Dublin, Ireland, November 1843.

[35] J. Hart, G. Francis, and L. Kauffmann. Visualizing Quaternion Rotation. *ACM Transactions on Graphics*, 13:256 – 276, July 1994.

[36] M. Hebert, M. Deans, D. Huber, B. Nabbe, and N. Vandapel. Progress in 3–D Mapping and Localization. In *Proceedings of the 9th International Symposium on Intelligent Robotic Systems, (SIRS '01)*, Toulouse, France, July 2001.

[37] H. Heuser. *Gewöhnliche Differentialgleichungen : Einführung in Lehre und Gebrauch.* Teubner, Stuttgart, 1989.

[38] F. Hoffmann, Chr. Icking, R. Klein, and K. Kriegel. The polygon exploration problem. *SIAM Journal on Computing*, 31(2):577–600, 2002.

[39] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71 – 78, 1992.

[40] B. Horn. Closed–form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629 – 642, April 1987.

[41] D. Huber. Automatic 3D Modeling Using Range Images Obtained from Unknown Viewpoints. In *Proceedings of the Third International Conference on 3D Digital Imaging and Modeling (3DIM '01)*, pages 153 – 160, Quebec City, Canada, May 2001.

[42] D. Huber, O. Carmichael, and M. Hebert. 3D map reconstruction from range data. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '00)*, volume 1, pages 891 – 897, San Francisco, CA, USA, April 2000.

[43] D. Huber and M. Hebert. A New Approach to 3–D Terrain Mapping. In *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS '99)*, pages 1121 – 1127, Kyonjyu, Korea, October 1999. IEEE.

[44] Daniel F. Huber and Martial Hebert. Fully automatic registration of multiple 3d data sets. In *IEEE Computer Society Workshop on Computer Vision Beyond the Visible Spectrum (CVBVS '01)*, December 2001.

[45] M. Hudelson. Graph Theory (*Math 453*), Fall Semester, Washington State University, Vorlesungsmitschrift, 1998.

[46] G. Indiveri. Kinematic Time-invariant Control of a 2D Nonholonomic Vehicle. In *Proceedings of the 38th Conference on Decision and Control, (CDC '99)*, Phoenix, USA, December 1999.

[47] M. Karpinski. Approximative Algorithmen für NP–harte Berechnungsprobleme (*APX 00*), Universität Bonn, Vorlesungsmitschrift, 2000.

[48] R. Klein, Chr. Icking, and E. Langetepe. Entwicklung, Analyse und experimentelle Erprobung von kompetitiven Algorithmen für die Bahnplanung autonomer Systeme, `http://web.informatik.uni-bonn.de/I/research/dfg-bahnplanung/`, 2001.

[49] R. Lange and P. Seitz. Solid-state time-of-flight range camera. *IEEE Journal of Quantum Electronics*, 37(3), March 2001.

[50] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anerson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The Digital Michelangelo Project: 3D Scanning of Large Statues. In *Proceedings of the ACM SIGGRAPH*, New Orleans, USA, July 2000.

[51] A. Lorusso, D. Eggert, and R. Fisher. A Comparison of Four Algorithms for Estimating 3-D Rigid Transformations. In *Proceedings of the 5th British Machine Vision Conference (BMVC '95)*, pages 237 – 246, Birmingham, England, September 1995.

[52] A. Nüchter and K. Lingemann. Ein 3D-Laserscanner für autonome mobile Roboter. In *Tagungsband zum 3. Fachwissenschaftlichen Informatikkongress - Informatiktage 2001*, pages 89 – 92, Bad Schussenried, Germany, November 2001.

[53] CAMERA: CAd Modelling of Built Environments from Range Analysis. `http://www.dai.ed.ac.uk/homes/rbf/CAMERA/camera.htm`, 2000.

[54] J. Orear. *Physik*. Weltbild Verlag GmbH, Augsburg, 1991.

[55] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987.

[56] CMU's 3D Robot Mapping Project. `http://www-2.cs.cmu.edu/~thrun/3D/index.html`, 2000.

[57] D. Prytherch and M. McLundie. So what is haptics anyway? *Research issues in Art Design and Media*, Spring(2):1 – 16, 2002.

[58] K. Pulli. Multiview Registration for Large Data Sets. In *Proceedings of the 2nd International Conference on 3D Digital Imaging and Modeling (3DIM '99)*, pages 160 – 168, Ottawa, Canada, October 1999.

[59] K. Pulli, T. Duchamp, H. Hoppe, J. McDonald, L. Shapiro, and W. Stuetzle. Robust meshes from multiple range maps. In *Proceedings IEEE International Conference on Recent Advances in 3D Digital Imaging and Modeling (3DIM '97)*, Ottawa, Canada, May 1997.

[60] M. Reed and P. Allen. 3-D Modeling from Range Imagery: An Incremental Method with a Planning Component. *Image and Vision Computing*, 17:99 – 111, 1999.

[61] M. Reed, P. Allen, and I. Stamos. Automated Model Acquisition from Range Images with View Planning. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR '97)*, Puerto Rico, 1997.

[62] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proceedings of the Third International Conference on 3D Digital Imaging and Modellling (3DIM '01)*, pages 145 – 152, Quebec City, Canada, May 2001.

[63] A. Sappa, M. Devy, and M. Garcia. Modelling Built Environments from Large Range Images using Adaptive Triangular Meshes. In *Proceedings of the 8th International Symposium on Intelligent Robotic Systems, (SIRS '00)*, pages 23 – 29, Reading, United Kingdom, July 2000.

[64] A. Sappa and M. Garcia. Incremental Multiview Integration of Range Images. In *Proceedings of the 15th IAPR International Conference on Pattern Recognition*, pages 546 – 549, Barcelona, Spain, September 2000.

[65] A. Sappa, A. Restrepo-Specht, and M. Devy. Range Image Registration by using an Edge–based Representation. In *Proceedings of th 9th International Symposium on Intelligent Robotic Systems, (SIRS '01)*, Toulouse, France, July 2001.

[66] W. Schröder, E. Forgber, and G. Röh. Laser range camera applications. In *Proceedings of the 5th ESA Workshop on Advanced Space Technologies for Robot Applications*, Noordwijk, The Netherlands, December 1998.

[67] 3D Terrain Mapping Using Range Sensors. `http://www-2.cs.cmu.edu/~dhuber/mapping/`, 1999.

[68] V. Sequeira, J. Goncalves, and M. Ribeiro. 3D environment modelling using laser range sensing. *Robotics and Automation*, 16:81 – 91, 1995.

[69] V. Sequeira, E. Wolfart K. Ng, J. Goncalves, and D. Hogg. Automated 3D reconstruction of interiors with multiple scan–views. In *Proceedings of SPIE, Electronic Imaging '99, The Society for Imaging Science and Technology /SPIE's 11th Annual Symposium*, San Jose, CA, USA, January 1999.

[70] T. Shermer. Recent Results in Art Galleries. *Proceedings of the IEEE*, 80(9):1384 – 1399, September 1992.

[71] D. Simon, M. Hebert, and T. Kanade. Real–time 3–D pose estimation using a high–speed range sensor. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA '94)*, volume 3, pages 2235 – 2241, San Diego, CA, USA, May 1994.

[72] A. Stoddart and A. Hilton. Registration of multiple point sets. In *Proceedings of the 13th IAPR International Conference on Pattern Recognition*, pages 40–44, Vienna, Austria, August 1996.

[73] A. Streri. *Seeing, Reaching, and touching (cited through [27]).* The MIT Press, Cambridge, MA, USA, 1993.

[74] H. Surmann, K. Lingemann, A. Nüchter, and J. Hertzberg. A 3D laser range finder for autonomous mobile robots. In *Proceedings of the of the 32nd International Symposium on Robotics (ISR '01)*, pages 153 – 158, Seoul, Korea, April 2001.

[75] H. Surmann, K. Lingemann, A. Nüchter, and J. Hertzberg. *Aufbau eines 3D–Laserscanners für autonome mobile Roboter, GMD Report 126.* GMD - Forschungszentrum Informationstechnik GmbH, Sankt Augustin, March 2001.

[76] H. Surmann, K. Lingemann, A. Nüchter, and J. Hertzberg. Fast acquiring and analysis of three dimensional laser range data. In *Proceedings of the of the 6th International Fall Workshop Vision, Modeling, and Visualization (VMV '01)*, pages 59 – 66, Stuttgart, Germany, November 2001.

[77] Wolfram Research, The Lyapunov Function. `http://mathworld.wolfram.com/LyapunovFunction.html`, 2002.

[78] S. Thrun, D. Fox, and W. Burgard. A real-time algorithm for mobile robot mapping with application to multi robot and 3D mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '00)*, San Francisco, CA, USA, April 2000.

[79] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence*, Summer 2001.

[80] Project to build a 3D fax machine. `http://graphics.stanford.edu/projects/faxing/`, 2001.

[81] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of the ACM SIGGRAPH '94*, pages 311 – 318, Orlando, FL, USA, July 1994.

[82] The ACTS RESOLV (REconstruction using Scanned Laser and Video) Project. `http://www.comp.leeds.ac.uk/resolv/`, 1999.

[83] B. Vatti. A Generic Solution to Polygon Clipping. *Communications of the ACM*, 35(7):56 – 63, 1992.

[84] WDR. Leonardo Schwerpunkt Haptik `http://capehorn.gmd.de:8080/download/haptik/haptik.mp3`.

[85] Z. Zhang. Iterative point matching for registration of free–form curves. Technical Report RR-1658, INRIA–Sophia Antipolis, Valbonne Cedex, France, 1992.

[86] H. Zhao and R. Shibasaki. Reconstructing Textured CAD Model of Urban Environment Using Vehicle-Borne Laser Range Scanners and Line Cameras. In *Second International Workshop on Computer Vision System (ICVS '01)*, pages 284 – 295, Vancouver, Canada, July 2001.

## Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Bonn, 1. Juli 2002