Hochschule
Bonn-Rhein-Sieg
Fachbereich Informatik

Fraunhofer
IAIS

Abschlussarbeit

im Bachelor-Studiengang

Segmentierung von 3D-Daten

von

Thomas Doll
thomas.doll1@smail.inf.fh-brs.de

Erstbetreuer:	Prof. Dr. Paul G. Plöger
Zweitbetreuer:	Dr. Erich Rome
Eingereicht am:	3. März 2011

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbst angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Thomas Doll

Servatiusstr. 134, 53175 Bonn

Zusammenfassung

Die vorliegende Arbeit wird im Rahmen eines Projektes des Fraunhofer Instituts IAIS erstellt. Hier geht es um die Entwicklung eines neuen 3D-Laserscanners. Basierend auf diesem 3D-Laserscanner soll eine Sicherheits-Anwendung realisiert werden. Für eine Softwarekomponente – die Segmentierung von 3D-Daten – wird der Stand der Forschung untersucht und es werden drei Segmentierungs-Verfahren ausgewählt und implementiert.

Der *RANSAC-Algorithmus* wird zur Detektion von Ebenen eingesetzt. In dieser Arbeit wird er um ein Abbruchkriterium erweitert, welches die Gesamtlaufzeit bei der Segmentierung von mehreren Ebenen verringert.

Über die *Nachbarschaftsanalyse* wird der Merkmalsvektor eines 3D-Datenpunktes durch verschiedene Berechnungen aus der lokalen Nachbarschaft erweitert. Mit diesem erweiterten Merkmalsvektor können bestimmte Merkmale aus den Gesamtdaten – wie Kanten, ebene und unebene Bereiche – extrahiert werden. In Kombination mit dem Fuzzy-C-Means-Algorithmus ist es möglich, diese Bereiche ohne weitere Parameter-Angabe aus 3D-Punktwolken zu bestimmen.

Über den *Bearing Angle* wird eine neue, zweidimensionale Repräsentation der Szene geschaffen. Dabei werden die speziellen Eigenschaften des Laserscanners genutzt und die Polardaten in Winkel überführt. Die Winkel können auf den HSV-Farbraum abgebildet werden. Die zweidimensionale Repräsentation der Winkel ist kontrastreicher als die Projektion der Distanzdaten auf ein Graubild. Sie ermöglicht eine genaue Detektion der Kanten.

Abschließend werden alle Verfahren hinsichtlich der Laufzeit und Güte der Segmentierung ausgewertet. Es wird gezeigt, dass die Laufzeit der Segmentierung bei dem Bearing-Angle-Verfahren im Mikrosekunden-Bereich liegt und sich daher für Echtzeitanwendungen eignet. Auch die Genauigkeit der richtig positiv erkannten Flächen ist bei diesem Segmentierungs-Verfahren, im Vergleich zu den anderen beiden, am höchsten.

Danksagung

Ganz besonders herzlich möchte ich an dieser Stelle meinen Betreuern, Dr. Erich Rome und Prof. Dr. Paul Plöger, danken. Dr. Erich Rome danke ich für die stetige Unterstützung bei meiner Arbeit und seine immer wertvollen Hinweise. Prof. Dr. Paul Plöger danke ich für seine geduldigen Erklärungen und inspirierenden Anstösse.

Bedanken möchte ich mich auch bei allen anderen Kollegen aus dem Fraunhofer IAIS Institut, die mich bei meiner Arbeit tatkräftig unterstützt haben. Ich bedanke mich auch sehr bei Tsung-Han Lin. Bei schwierigen Fragen hat er mich durch seine Hartnäckigkeit beeindruckt. Bei Thorsten Linder möchte ich mich für seine stets guten Ratschläge bedanken.

Nicht zuletzt danke ich meiner Mutter für ihr sorgfältiges Korrekturlesen.

Inhaltsverzeichnis

Algorithmenverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Problemstellung / Ausgangssituation

Das Forschungsprojekt FOVEA-3D am Fraunhofer Institut IAIS beschäftigt sich mit der Entwicklung eines innovativen Laserscanners. Das Projekt wird seit Februar 2010 von drei Fraunhofer-Instituten durchgeführt. Das Fraunhofer IPM in Freiburg ist für die Messtechnik und Hardware-Steuerung zuständig. Das Fraunhofer IPMS in Dresden ist für die Spiegeltechnik verantwortlich, und das Fraunhofer IAIS in Sankt Augustin entwirft die Software. Es gibt mehrere Anwendungsszenarien für den neuen Scanner, unter anderem Robotik, geologische Anwendungen und präventive Sicherheit auf Basis von 3D-Daten. Das IAIS verfügt über ein spezielles C++ Framework, welches Treiber, Datenstrukturen und Methoden für die Verarbeitung von 3D-Daten bereitstellt. Da der FOVEA-Scanner noch entwickelt wird, steht für Vorarbeiten ein IAIS-3D-Scanner auf Basis eines SICK-Laserscanners zur Verfügung. Dieser verfügt zwar nicht über die Auflösung und die Reichweite des neuen FOVEA-Scanners, doch grundsätzlich liefert er die gleichen Polarkoordinaten der Messpunkte und Remissionswerte und kann somit als Übergangslösung für die Entwicklung verwendet werden.

1.2 Zielsetzung

Es sollen Algorithmen entwickelt werden, um die resultierenden Punktwolken des Laserscanners zu analysieren. Langfristig soll eine Objekterkennung auf Basis der 3D-Daten in Echtzeit realisiert werden. Als Vorstufe zu der Objekterkennung soll in dieser Arbeit die Segmentierung vorgenommen werden. Ziel hierbei ist eine robuste, unüberwachte und automatische Segmentierung der Objekte einer Szene innerhalb einer strukturierten Umgebung. Dazu sollen die Daten mit Methoden der Mathematik, Bildverarbeitung und Information Retrieval untersucht werden. Gegenstände dieser Bachelor Arbeit sind:

- Recherche der einzusetzenden State-of-the-Art-Verfahren
- Implementierung und Evaluation dieser Segmentierungsverfahren
- Weiterentwicklung und Verbesserung geeigneter Verfahren – (optional)
- Echtzeitfähige Algorithmen finden und implementieren – (optional)

1.3 Vorgehen

Als erstes wird der Stand der Forschung bezüglich der Segmentierung von 3D- und 2.5D-Punktwolken dargestellt. Darauf basierend werden zunächst einige Verfahren ausgewählt, die für die Segmentierung benutzt werden können. Diese werden dann auf ihre Verwendbarkeit und den bei der Implementierung entstehenden Aufwand hin geprüft. Daraufhin wird entschieden, welche Methoden im Rahmen dieser Arbeit implementiert werden können.

Programmiert wird mit verschiedenen C++ Bibliotheken. Das Fraunhofer IAIS hat eine eigene Robotik-Bibliothek, die FAIRLib, in welcher bereits viele benötigte Datentypen vorhanden sind. Auch die Ansteuerung des Scanners und die Visualisierung der Punktwolken können hiermit vorgenommen werden. Fehlende Funktionalitäten sollen mit Open-Source-Bibliotheken, welche die kommerzielle Nutzung erlauben, ausgeglichen werden. Diese müssen gegebenenfalls noch recherchiert werden.

Die Auswertung der Ergebnisse soll sich an realistischen Szenarien orientieren. Es sind verschiedene Einsatzzwecke für den Fovea-Scanner angedacht. Einige sind laufzeitkritisch, bei anderen ist die erzielte Genauigkeit entscheidend. Ein Aspekt für die Auswertung ist daher die zuverlässige Erkennung von Objekten, wie z. B. von Böden und Wänden. Weitere Aspekte sind die Geschwindigkeit und somit die Echtzeitfähigkeit. Über beide Aspekte soll mit Hilfe einfacher Experimente schließlich eine Aussage ermöglicht werden. Es soll mindestens ein Vergleich der Laufzeiten und der Genauigkeit der realisierten Verfahren vorgenommen werden.

Im zweiten Kapitel wird auf den Projektkontext, benötigte Software-Bibliotheken und grundlegende Begriffe eingegangen. Das dritte Kapitel fasst den aktuellen Stand der Forschung im Bereich „Segmentierung von 3D-Daten“ zusammen. Danach wird im vierten Kapitel das Konzept und die Implementierung von drei ausgewählten Segmentierungs-Verfahren beschrieben. Eine Auswertung dieser drei Segmentierungs-Verfahren wird in Kapitel fünf vorgenommen. Abschließend werden in Kapitel sechs alle Ergebnisse zusammengefasst und ein Ausblick auf weitere Arbeiten gegeben.

2 Grundlagen

Im Folgenden wird auf den Projektkontext, benötigte Software-Bibliotheken und grundlegende Begriffe eingegangen.

2.1 Projektkontext am Fraunhofer IAIS

Das Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme IAIS ist in den Bereichen Marketing, Marktforschung & Mediaanalyse, Unternehmensplanung & Controlling, Digital Media Asset Management, Process Intelligence, Preventive Security, High-Tech-Erlebnisräume, Robotik und Medienproduktion aktiv. Im Gegensatz zu anderen Forschungsinstituten müssen die meisten Fraunhofer-Projekte stets durch Kunden aus der Wirtschaft mitfinanziert werden. Hieraus ergibt sich eine sehr effektive Arbeitsweise [IAIS, 2011].

Das Projekt Fovea-3D ist in das IAIS-Geschäftsfeld Preventive Security eingeordnet. Die Arbeiten für das Projekt begannen im Februar 2010 und die Laufzeit beträgt drei Jahre. Als Pilotanwendungsbereich ist die Überwachung von Liegenschaften angedacht.

Die Hauptaufgaben des Instituts IAIS im Projekt Fovea-3D sind die Realisierung der Basis- und Steuerungssoftware für den Fovea-Scanner sowie die Nachverarbeitung der 3D-Scannerdaten. Die Basis- und Steuerungssoftware beinhaltet vor allem die Schnittstelle zum Scanner. Die Nachbearbeitung der 3D-Daten besteht aus mehreren, voneinander abhängigen, Phasen. Diese sind:

- 1. Bestimmung einer Region of Interest (ROI) - (optional)
- 2. Segmentierung und Kartierung der Szene
- 3. Entdeckung und Grobklassifikation von Szenenänderungen

In der vorliegenden Arbeit sollen Methoden für den zweiten Schritt, Segmentierung und Kartierung der Szene, gefunden und realisiert werden.

2.2 3D-Laserscanner

Es gibt verschiedene Hersteller von Laserscannern, und diese setzen unterschiedliche Verfahren zur Generierung von 2D- und 3D-Daten ein. Eine Gemeinsamkeit haben alle bekannten Techniken: Die generierten Daten sind abhängig von der Position und der eingestellten

Perspektive des Geräts. Alles, was sich außerhalb des Blickfeldes befindet, kann nicht erfasst werden. Dies ist besonders auffällig bei sich überschneidenden Objekten und führt dazu, dass die 3D-Repräsentation dieser Objekte nur unvollständig erfasst wird. Daher wird die Datenerfassung auch als 2.5-dimensional (2.5D) bezeichnet. Die fehlenden räumlichen Informationen können nur aus einer anderen Perspektive aufgenommen werden. Dennoch hat die resultierende Punktwolke alle Eigenschaften einer 3D-Punktwolke. Daher kann von 3D-Daten gesprochen werden.

Die Laserscanner speichern die 3D- bzw. 2.5D-Daten in Form von Polarkoordinaten. Zusätzlich werden Helligkeitswerte gespeichert, die sogenannten *Remissionswerte*. Es gibt mehrere Laserscanner-Modelle, welche mit unterschiedlichen Techniken arbeiten. So wird zwischen Time-of-Flight- und Phasen-Modulations-Technik unterschieden. Die generierten 3D-Daten sind bei beiden Techniken gleich.

Der im Fraunhofer IAIS verwendete 3D-Laserscanner (Abb. 2.1) ist ein 2D-Laserscanner der Firma SICK. Dieser ist mittig auf eine Neigevorrichtung montiert, welche über einen Servomotor angesteuert werden kann. Mit dieser Kombination ist die Aufnahme von 3D-Daten möglich. So scannt der 2D-Laserscanner, von unten beginnend, immer eine horizontale Ebene und wird nach jedem Durchlauf von der Neigevorrichtung ein wenig nach oben rotiert. Somit wird die horizontale Auflösung von dem SICK-2D-Laserscanner und die vertikale Auflösung von der Neigevorrichtung bestimmt. Die (vorhandene) Kombination aus Laserscanner und Neigevorrichtung ermöglicht eine maximale Auflösung von 400 x 498 Punkten bei einem Öffnungswinkel von 100° x 120°. Zusammen sind dies 199.200 Punkte.

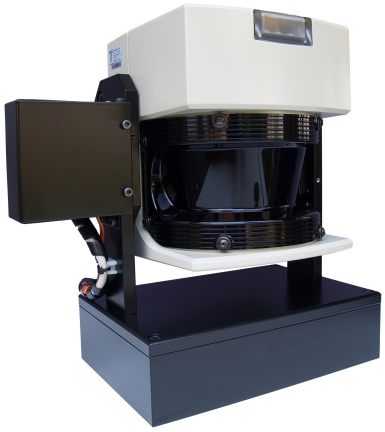

Abbildung 2.1: Verwendeter SICK- Laserscanner [IAIS, 2011]

Die generierten Umgebungsdaten eines Laserscanners bestehen immer aus polaren Koordinaten. In der Praxis werden diese meist in kartesische Koordinaten umgerechnet. Die meisten Segmentierungs-Verfahren basieren auf dem kartesischen Koordinatensystem. Zum einen sind diese für uns Menschen intuitiver zu verarbeiten. Zum anderen wird auch in an-

deren wissenschaftlichen Bereichen seit langer Zeit an der Verarbeitung von 3D-Daten gearbeitet, so z. B. in der Medizin und Computergrafik. Es können daher viele Segmentierungs-Verfahren auf die von Laserscannern erzeugten Punktwolken angewandt werden.

Eine weitere Information, die von Bedeutung sein kann, ist die Reihenfolge von Punkten. So wird in der Literatur oft zwischen „geordneten/structured“ und „ungeordneten/unstructured“ Punktwolken gesprochen [Surmann u. a., 2003]. Der SICK-2D-Scanner nimmt jede Zeile von links nach rechts auf und speichert die Daten in dieser Reihenfolge. Der Servomotor beginnt von unten und inkrementiert nach jedem Scannerdurchlauf 0,25° nach oben. Als Ergebnis wird eine „geordnete“ Punktwolke ausgegeben.

2.3 Mathematik

Die folgenden Verfahren werden benötigt, um die Punktwolken mit statistischen Verfahren zu untersuchen. Sie werden in dieser Arbeit ausschließlich auf kartesische Koordinaten angewandt.

2.3.1 Nächste Nachbarn

Die *Nächsten Nachbarn* können für die unterschiedlichsten Daten berechnet werden. Voraussetzung ist, dass die Daten in Merkmalsvektoren aufgeteilt werden können und eine Distanzfunktion für sie definiert ist. Die nächsten Nachbarn eines 3D-Datenpunktes (x-, y- und z-Koordinate als Merkmalsvektoren) meint in diesem Zusammenhang die Datenpunkte mit dem geringsten Abstand zu dem aktuell untersuchten Datenpunkt. Im kartesischen Koordinatensystem ist hierfür der euklidische Abstand zu berechnen. Typischerweise definiert man eine Anzahl von nächsten Nachbarn, die sogenannten *k-Nächsten-Nachbarn*. Hierbei muss die Anzahl k sinnvoll vom Benutzer gesetzt werden. Alternativ können auch die *Fixierten-Nächsten-Nachbarn* gewählt werden. Dabei kann der Benutzer einen Abstand bestimmen. Innerhalb dieses Abstandes werden alle Datenpunkte als nächste Nachbarn angesehen.

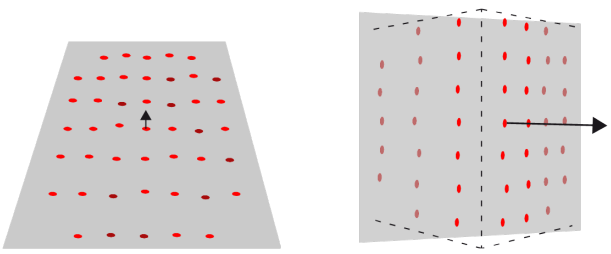

Abbildung 2.2: Nächste Nachbarn: Unterscheidung von Flächen und Kanten [IAIS, 2011]

Mithilfe der nächsten Nachbarn lässt sich eine Aussage über die Punkt-Verteilung in der Nachbarschaft machen und es können Flächen, Kanten und Ecken gefunden werden (s. Abb. 2.2).

2.3.2 Normalenvektor

Der Normalenvektor $\vec{n}$ wird benötigt, um die Lage eines geometrischen Körpers eindeutig zu bestimmen. Im einfachsten Fall ist das die Lage einer Ebene. Der Normalenvektor wird beschrieben durch den Richtungsvektor einer Geraden, welche senkrecht auf der Oberfläche einer Ebene steht. Es genügt eine Koordinate einer Ebene und deren Normalenvektor, um die Ebene eindeutig zu bestimmen. Der Normaleneinheitsvektor $\vec{n}_0$ hingegen hat immer die Länge „Eins“. Somit reicht dieser aus, um eine Ebene zu bestimmen, ohne dass eine Koordinate dieser Ebene bekannt sein muß.

Für die Normalform der Ebene:

$$ax + by + cz = d \tag{2.1}$$

ist

$$\vec{n} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{2.2}$$

der Normalenvektor.

2.3.3 Kovarianz-Analyse

Die Kovarianz-Analyse kann verwendet werden, um 3D-Punkte auf Basis von geometrischen Eigenschaften zu klassifizieren. Dieses Verfahren wird oft als Ausgangspunkt verwendet, um Strukturen innerhalb großer Punktwolken zu finden [Pauly u. a., 2002]. Hierbei wird die Kovarianz-Matrix für die lokale Nachbarschaft eines Punktes von Interesse berechnet.

$$\Sigma_X = COV(X) = \begin{bmatrix} \delta_x^2 & \delta_{xy} & \delta_{xz} \\ \delta_{yx} & \delta_y^2 & \delta_{yz} \\ \delta_{zx} & \delta_{zy} & \delta_z^2 \end{bmatrix} \tag{2.3}$$

Hierbei sind die Werte in der Kovarianz-Matrix für k Nachbarn definiert als

$$\delta_x^2 = var(x) = E(x^2) - E(x)^2 = \frac{1}{k}\sum_{i=1}^{k}(x_i - \bar{x})^2 \tag{2.4}$$

$$\delta_{xy} = cov(x, y) = E(xy) - E(x)E(y) = \frac{1}{k}\sum_{i=1}^{k}(x_i - \bar{x})(y_i - \bar{y})^2 \tag{2.5}$$

Hierbei entspricht $E(x)$ dem Mittelwert, $var(x)$ der Varianz und $cov(x, y)$ der Kovarianz. Die Werte x, y und z symbolisieren die jeweils untersuchte Achse.

Weiterhin werden bei der Kovarianz-Analyse die Eigenwerte (λ_i) und Eigenvektoren (e_i) aus der Kovarianz-Matrix berechnet. Die Eigenvektoren entsprechen den Hauptkomponenten (Richtungen) der Nachbarschaft. Die Eigenwerte kennzeichnen die Varianz in die jeweilige Richtung des zugehörigen Eigenvektors. Als Ergebnis werden drei Eigenvektoren mit ihren Eigenwerten ausgegeben. Die Orientierung der so untersuchten Fläche kann mithilfe des Eigenvektors mit dem kleinsten Eigenwert bestimmt werden. Dieser Eigenvektor ist zwar sehr kurz, doch seine Richtung entspricht dem Normalenvektor der untersuchten Oberfläche.

Das Ergebnis ist analog zu der Berechnung durch den „kleinsten quadratischen Abstand“ (least squares). Hier wird die optimale Ebene durch Minimierung der Summe der quadrierten Residuen bestimmt [Strang, 2003].

Eine weitere Möglichkeit zur Feststellung der Krümmung in der Nachbarschaft ist die Oberflächen-Varianz. Hierbei wird die Varianz in Richtung der *Normale* – dies ist die Gerade, welche den Normalenvektor als Richtungsvektor besitzt – durch die gesamte Varianz der Nachbarschaft geteilt [Tong u. a., 2004].

$$\delta_n^2(p) = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3} \tag{2.6}$$

Der Unterschied zwischen Kovarianz-Analyse und Oberflächen-Varianz ist, dass die Oberflächen-Varianz robuster auf unterschiedliche Nachbarschaftsverteilungen innerhalb einer Punktwolke reagiert. Die Kovarianz-Analyse hingegen benötigt eine sehr homogene Verteilung zur Erreichung des Ergebnisses.

Meist wird für jeden Punkt in der Punktwolke seine Orientierung relativ zur Nachbarschaft und weiterhin die Krümmung dieser Nachbarschaft berechnet, sodass am Ende jeder Punkt eine neue Information besitzt. Dies ist ein Wert, der die Varianz in der Nachbarschaft repräsentiert.

2.4 Information Retrival

Ein wesentliches Problem beim Umgang mit großen Datenmengen besteht darin, die gesuchte Information schnell und zuverlässig zu erhalten. Im einfachsten Fall wird die Datenmenge manuell durchsucht, oder es wird eine Aussage auf Basis von Stichproben vorgenommen. Es gibt einige Algorithmen aus dem Bereich „Information Retrieval“ (zu Deutsch: Informationsbeschaffung), welche automatische und adaptive Verfahren liefern, die große Datenmengen nach den jeweils gewünschten Kriterien untersuchen können. Bei der Segmentierung von 3D-Daten können diese verwendet werden, um Datenpunkte mit ähnlichen Eigenschaften zu gruppieren.

2.4.1 Clustering

Es gibt verschiedene Clustering-Algorithmen, welche es ermöglichen, bestimmte Informationen automatisch in verschiedene Gruppen zu sortieren. Ein einfaches Beispiel, bei dem dieses Verfahren eingesetzt wird, ist die Gruppierung von Webseiten. Suchmaschinen verarbeiten die Texte von Millionen Webseiten und sortieren auf Basis des verwendeten Fachvokabulars ähnliche Webseiten in gleiche Gruppen ein. Diese Vorverarbeitung der Texte erlaubt später ein schnelleres Durchsuchen der Webseiten nach ähnlichen Themengebieten. Bekannt sind vor allem die *Hierarchische Clusteranalyse* und die *k-means-Clusteranalyse*.

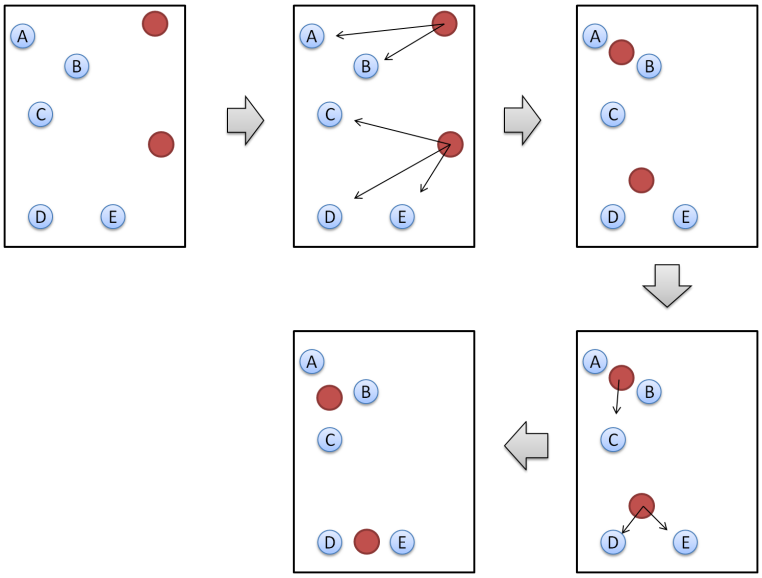

Abbildung 2.3: Beispiel für k-means-Clustering

In Abbildung 2.3 ist ein einfaches Beispiel für *k-means-Clustering* dargestellt. Die blauen Kreise mit den Buchstaben symbolisieren die Datenvektoren, und die roten Kreise stellen die Zentroide dar. Schritt für Schritt wird die Distanz der Datenvektoren zu den Zentroiden berechnet. Dann wird auf Basis der kürzesten Distanz eine Zuweisung von Datenvektor zu Zentroid vorgenommen. Jedes Zentroid berechnet seine neue Position über den Mittelwert der ihm zugewiesenen Datenvektoren. Dies wird so oft wiederholt, bis die Zentroide konvergieren, d. h. nicht mehr ihre Position verändern. Die Distanzfunktion allein entscheidet, wie die Zuordnung vorgenommen wird und kann somit auch auf mehrdimensionale Daten angewandt werden [Segaran, 2008].

2.4.2 Fuzzy-k-means

Ein Nachteil der zuvor erwähnten Clustering-Algorithmen ist, dass sie sehr empfindlich gegen Ausreißer sind. Für ein CAD-Modell würden diese Verfahren reichen, um zwischen Ebenen, Unebenen und Kurven zu unterscheiden. Das von einem Laserscanner erzeugte

3D-Modell ist jedoch nicht annähernd so genau wie ein CAD-Modell. Für Daten mit vielen Unregelmäßigkeiten ist die Fuzzy-Logik gut geeignet. Diese arbeitet mit unscharfen Grenzen (s. Abb. 2.4) und kann sich so an die Daten anpassen.

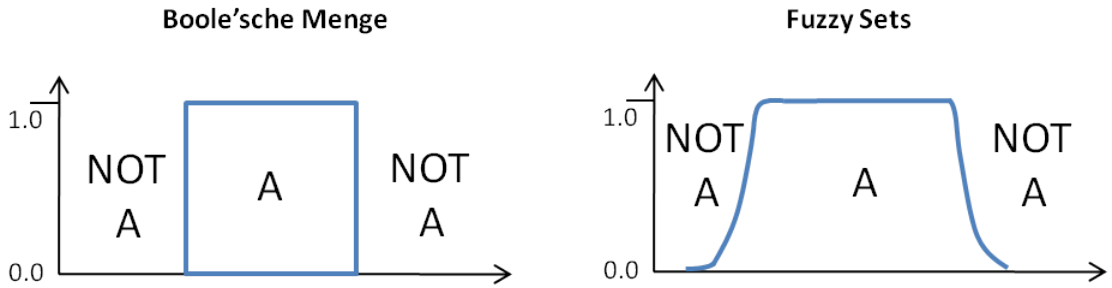

Abbildung 2.4: Scharfe und unscharfe Grenzen

Beim *Fuzzy-k-means-Clustering* wird in mehreren Iterationen eine optimale Gruppierung der vorhandenen Daten vorgenommen. Das Verfahren arbeitet adaptiv. Das heißt, es passt sich den Unregelmäßigkeiten in den Daten an und findet die entscheidenden Grenzen. Das Vorgehen ist analog zum k-means-Clustering – nur ist hier die Formel zur Berechnung der Zugehörigkeit etwas komplizierter.

Zu Beginn müssen die Zentroide zufällig oder bestimmt verteilt werden. Danach wird von jedem Datenvektor die Distanz zu sämtlichen Zentroiden berechnet. Bis hierhin ist das Vorgehen analog zum *k-means-clustering*. Nun wird aber nicht nur die Zugehörigkeit anhand der kürzesten Distanz abgeleitet, sondern es wird der *Membership-Score* $m_{X_iV_j}$ von jedem Zentroid X_i zu jedem Datenvektor V_j mit $j < k$ berechnet:

$$m_{X_iV_j} = \frac{\frac{1}{d^2_{X_iV_j}}}{\sum_{j=1}^{k} \frac{1}{d^2_{X_iV_j}}} \tag{2.7}$$

Danach wird mithilfe des Membership-Score die Position der Zentroide neu berechnet:

$$V_j = \frac{\sum_{i=1}^{N} m^2_{X_iV_j} X_i}{\sum_{i=1}^{N} m^2_{X_iV_j}} \tag{2.8}$$

Auf diese Weise wird die Position der Zentroide so oft neu berechnet, bis diese konvergieren.

2.5 Bildverarbeitung

Die polaren Koordinaten können ähnlich wie ein RGB-Bild dargestellt werden. Durch den fixen Blickwinkel des Laserscanners ist das Prinzip wie das einer Kamera. Diese braucht eine fixierte Position und im Optimalfall eine statische Szene, um ein Bild einzufangen.

Die Auflösung des Scanners ist äquivalent zu der Auflösung einer Kamera. Für jede polare Koordinate wird die x- und y-Achse als Position und die z-Achse als Farbwert im Bild codiert. Diese Darstellung der Punktwolke lässt sich nur von 2.5D-Daten ohne Informationsverlust abbilden. Bei echten 3D-Daten ließen sich nicht alle Punkte in einem 2D-Bild darstellen. Da der SICK-Scanner jede Zeile mit 400 Punkten aufnimmt und der Servomotor 498-mal seine Position nach oben inkrementiert, ergibt sich ein rechteckiges Bild mit 400 x 498 Pixeln. Im Gegensatz zu einer Kamera hat der Laserscanner den einzelnen Pixeln noch keine Farbwerte zugeordnet. Auch werden die sphärischen Effekte nicht korrigiert, d. h., eine gerade Wand wird in den polaren Koordinaten kugelartig dargestellt (s. Abb. 2.5). Dies ist eine Eigenschaft des Scan-Verfahrens, welches während der Generierung der Polardaten um zwei Achsen rotiert .

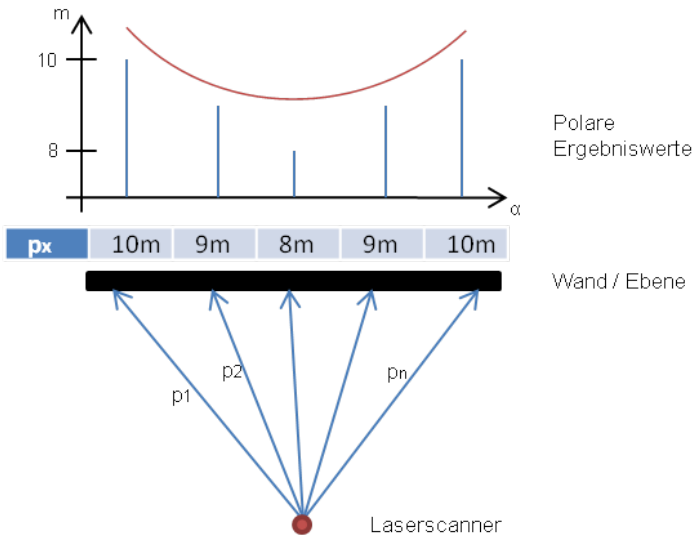

Abbildung 2.5: Entstehung des sphärischen Effektes

Die Ergebnisse einer solchen Repräsentation der polaren Daten können noch davon beeinflusst werden, auf welchen Farbraum die Daten projiziert werden. Im einfachsten Fall werden die polaren Distanzdaten auf ein 8-Bit-Graubild abgebildet [Nüchter und Hertzberg, 2008]. Es ist aber auch möglich, diese auf den RGB- oder HSV-Farbraum abzubilden.

Die Remissionswerte haben etwas andere Eigenschaften als die polaren Koordinaten. Hier gibt es auch sphärische Effekte. Der Unterschied ist jedoch, dass die Remissionswerte direkt als Grauwerte interpretiert werden können, da sie dem Hell-Dunkel-Kontrast einer Oberfläche entsprechen. Dabei werden gerade Kanten jedoch durch die sphärische Verzerrung etwas rundlich abgebildet. Die verschiedenen Repräsentationen einer Aufnahme sind in Abbildung 2.6 dargestellt.

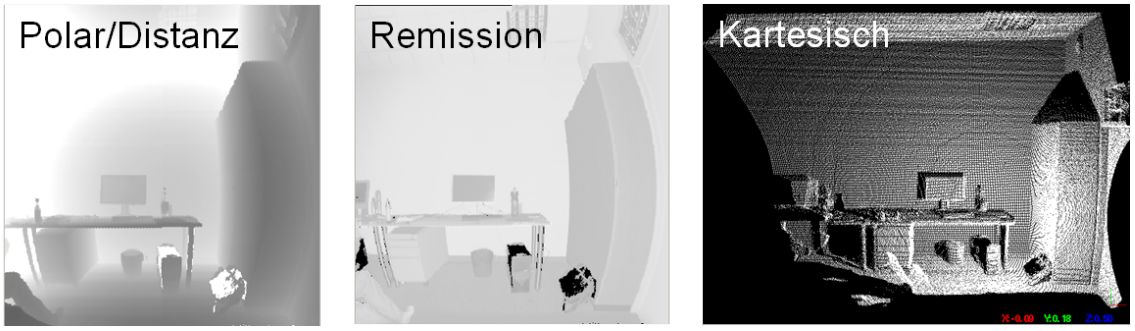

Abbildung 2.6: Bilddarstelung der Entfernungs- und Remissionswerte und 3D-Darstellung der Punkte im kartesischen Koordinatensystem

2.5.1 Isotropes und Anisotropes Filtern

In der Bildverarbeitung werden oft Glättungsfilter eingesetzt. Diese werden benötigt, um Rauschen im Bild zu entfernen. Sie verhalten sich wie Tiefpass-Frequenz-Filter und entfernen hohe Frequenzen im Bild, welche als Rauschen angesehen werden. Einer der bekanntesten Filter dieser Art ist der *Gauss-Filter*. Bei diesem wird eine Maske (Faltungsfilter) über das Bild gelegt, z. B. eine 3 x 3- oder 5 x 5-Maske. Aus dieser Maske wird ein gewichteter Mittelwert berechnet und das Ergebnis in ein neues Bild geschrieben. Nachdem die Maske über alle Pixel des Originalbildes geschoben wurde und jeweils ein Pixel in das neue Bild geschrieben wurde, ist die Ausgabe in dem neuen Bild eine gefilterte Version des Originals [Jähne, 2005]. Bezogen auf die 2.5D-Daten hat dieser Filter jedoch einen wesentlichen Nachteil, nämlich sein Verhalten bei Ausreißern. Da ein Mittelwert über alle Werte in der Nachbarschaft berechnet wird, kann das Ergebnis ungenauer sein als das Original. In der Robotik wird häufig der Medianfilter eingesetzt. Dieser funktioniert nach demselben Prinzip, jedoch wird hierbei der Median innerhalb der Maske ermittelt und in das neue Bild geschrieben. Dieser kann im Gegensatz zu den Mittelwert-Filtern Ausreißer erfolgreich eliminieren.

Bei der Anwendung auf Laserscanner-Daten bleibt jedoch ein wesentliches Problem mit allen Filtermasken aus der Bildverarbeitung bestehen. Alle Pixelwerte im Bild werden voneinander statistisch unabhängig behandelt. Darum werden diese auch *isotrope Filter* genannt. Alle Frequenz-Filter werden in diese Kategorie eingeordnet. Bei einem RGB-Bild einer Kamera sind die Farbwerte der Nachbarpixel unabhängig von der Richtung, weil jedes einzelne Pixel mit 100%iger Sicherheit die Farbe des Ziel-Bereichs wiedergibt. Laserscanner-Daten hingegen liefern keine absolut genauen Werte. Im Gegenteil entsteht das Rauschen in den Daten durch die Fehlertoleranz des Laserscanners und ist durchgängig.

Um das Rauschen aus den Scannerdaten zu entfernen, muss ein Filter gewählt werden, welcher die Richtungsabhängigkeit in der Nachbarschaft beachtet. Solche Filter werden „anisotrop“ genannt. Ein Beispiel hierfür ist der *bilineare Filter*. Dieser verwendet ein Ähn-

lichkeitsmaß zur Bewertung der Beziehung von benachbarten Pixeln und steuert dadurch die Intensität des Filterungsprozesses für verschiedene Regionen. Durch dieses Vorgehen können scharfe Grenzen, wie sie bei Kanten auftreten, beibehalten werden [Harati, 2008].

2.5.2 Wavelet-Transformation

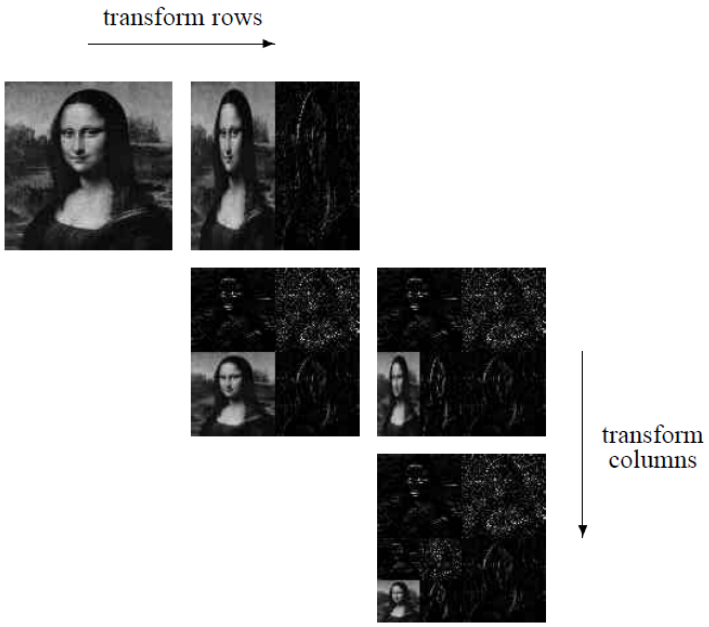

Abbildung 2.7: *Haar-Transformation* abwechselnd in horizontaler und vertikaler Richtung [Stollnitz u. a., 1996a]

Das Wort „Wavelet“ stammt aus der Geophysik und wurde bis 1990 ausschließlich in der Signalverarbeitung verwendet. Nachdem 1988 der Algorithmus „Fast Wavelet-Transformation“ und 1989 in diesem Zusammenhang die „Multi-Resolution-Analysis“ eingeführt wurde, haben sich Wavelet-Transformationen in der Bild- und Audio-Kompression durchgesetzt. Das JPEG-Format zum Beispiel basiert auf dem Konzept der diskreten Wavelet-Transformation [Stollnitz u. a., 1996b].

Eine Wavelet-Transformation besteht immer aus zwei Schritten: Der Wavelet-Analyse und der Wavelet-Synthese. Die Analyse bezeichnet hierbei eine Funktion, die den Übergang in die Wavelet-Darstellung definiert. Die Synthese meint die Rücktransformation, nämlich eine Funktion, mit der die originalen Daten wiederhergestellt werden können.

Eine wesentliche Eigenschaft von 3D-Laserscanner-Daten ist, dass sie ohne weitere Vorverarbeitung in eine niedrigere Auflösung konvertiert werden können. So teilt sich die Auflösung durch Löschen jedes zweiten Wertes in horizontaler und vertikaler Richtung. Diese Eigenschaft wird *Multi-Skalierbarkeit* (Multi-Resolution) genannt. Die Wavelet-Transformation kann genau diese Eigenschaft ausnutzen.

Das einfachste und älteste Wavelet ist das Haar-Wavelet. Dieses wird oft in der Bildverarbeitung eingesetzt. Angewandt auf 2D-Bilder lässt sich mithilfe dieser Transformation das Bild herunterskalieren (Wavelet-Analyse), wobei auch sämtliches Rauschen im Bild verschwindet. Nebenbei werden sogenannte *Detail-Koeffizienten* gespeichert. Mithilfe dieser lässt sich das herunterskalierte Bild wieder in das Original verwandeln (Wavelet-Synthese). Durch geschicktes Manipulieren der Detail-Koeffizienten ist es möglich, ein geglättetes Bild in der originalen Auflösung zurückzuerhalten. Es gibt keine eindeutig definierte Formel für diese Transformation. Der Grundgedanke besteht darin, immer zwei benachbarte Pixel zusammenzufassen. Pro Transformation wird das Bild somit auf die Hälfte skaliert. Im einfachsten Fall wird für jedes Pixelpaar der Mittelwert berechnet. Der Detail-Koeffizient wäre hierbei die Differenz der beiden Pixel geteilt durch zwei. Mit diesem Wert und dem zuvor berechneten Mittelwert lassen sich die zwei Werte des Originalbildes wiederherstellen. Diese Transformation muss für Zeilen und Spalten einzeln durchgeführt werden (s. Abb. 2.7).

2.5.3 Kantendetektion

Um Objekte voneinander zu unterscheiden, ist es sinnvoll, nach Kanten zu suchen. In Grauwert-Bildern lassen sich Kanten gut finden, indem nach scharfen Helligkeitswechseln gesucht wird. Der einfachste Kantendetektions-Filter ist der Sobel-Filter (s. Abb. 2.8). Dieser wird jeweils einzeln auf die horizontalen und vertikalen Bildpunkte angewandt. Die Faltungsmatrix bewegt sich über das ganze Bild und berechnet in jeder Position die Bildpunkt-Helligkeitswerte. Als Ergebnis wird ein Grauwert-Bild ausgegeben, in welchem die Bereiche mit den größten Helligkeitsunterschieden markiert sind. Diese Bereiche entsprechen den Kanten im Originalbild. In der Praxis wird aus diesem Bild über ein Schwellenwertverfahren meist noch ein Binärbild erstellt. Über den eingesetzten Schwellenwert kann die Genauigkeit gesteuert werden. Je niedriger der Schwellenwert angesetzt wird, desto mehr Kanten werden als solche erkannt, aber es werden dadurch auch Ungenauigkeiten und Rauschen verstärkt. Wird der Schwellenwert zu hoch angesetzt, können Kanten unvollständig auftreten. Es ist beim Sobel-Filter also entscheidend, den richtigen Schwellenwert zu finden.

Es gibt weitere Kantendetektions-Filter, wie den *Laplace-* und den *Canny-Filter*. Der Laplace-Filter verwendet eine Faltungsmatrix. Der Canny-Filter hingegen ist etwas komplizierter aufgebaut, arbeitet in mehreren Stufen und liefert dafür optimale Ergebnisse [Jähne, 2005]. Als Startpunkt reicht oft der Sobel-Filter aus.

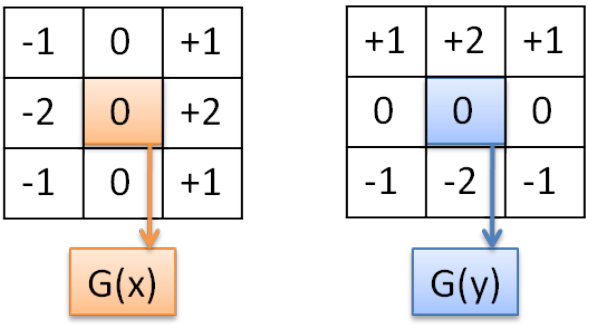

Abbildung 2.8: 3 x 3 Sobel-Maske in horizontaler und vertikaler Richtung

2.5.4 Segmentierung

Bei der Segmentierung geht es darum, eine Szene in kleinere, zusammengehörige Segmente zu teilen. Welche Segmente zusammengehörig sind, kann auf verschiedene Weise bestimmt werden und ist nicht eindeutig definiert. Meist wird ein Homogenitätskriterium ausgewählt. In der Bildverarbeitung wird bevorzugt der Farbwert hierfür verwendet. Auf 3D-Daten ist die Auswertung geometrischer Flächen oder der Oberflächenbeschaffenheit denkbar.

Im Kontext der Bildverarbeitung und des maschinellen Sehens ist die Segmentierung der erste Schritt der Bildanalyse. Die einzelnen Schritte sind hierbei wieder vergleichbar mit der visuellen Wahrnehmung des Menschen. In Abbildung 2.9 ist der Verlauf des maschinellen Sehens vereinfacht dargestellt. Die Segmentierung soll Konturen von Objekten hervorheben. Eine semantische Einordnung der Szene wird erst später vorgenommen. Die Hauptaufgabe der Segmentierung besteht darin, Umrisse von Objekten zu erkennen und diese von anderen Objekten in der Szene abzusetzen.

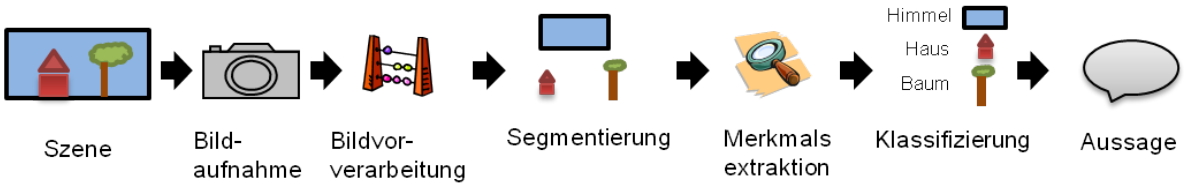

Abbildung 2.9: Vorgang des maschinellen Sehens

3 Stand der Forschung

Der aktuelle Stand der Forschung im Bereich „Segmentierung von 3D-Daten“ ist nicht einfach zu erfassen. Es wird seit über dreissig Jahren an verschiedensten Methoden zur Segmentierung von 3D- und 2.5D-Daten gearbeitet, ohne dass es eine annähernd optimale Lösung für diese Problemstellung gibt. Die einzelnen Verfahren wurden hauptsächlich als Forschungsarbeiten veröffentlicht und sind über Tausende von Dokumenten auf zahlreichen Wissensportalen verstreut.

3.1 Entwicklung

Abbildung 3.1: Microsoft Kinect Kamera [Microsoft, 2011]

Es gibt keine vergleichbaren Fortschritte wie in der Bildbearbeitung, die sich Dank der vielen Geräte und des einheitlichen Standards sehr schnell entwickeln konnte. Dies liegt möglicherweise daran, dass Laserscanner bis vor kurzem nur einem relativ kleinen Personenkreis zugänglich waren. Ende 2010 hat Microsoft für seine aktuelle Spielekonsole Xbox360 eine Kombination aus Tiefensensor und Kamera herausgebracht: Die *Kinect Kamera* [Microsoft, 2011]. Der Hersteller des Tiefensensors (PrimeSense) hat sogar ein *Open Source Entwickler-Kit* für diesen veröffentlicht. Es ist bereits möglich, Kinect am PC anzusteuern. Auch andere Hersteller haben Anfang 2011 angekündigt, ähnliche Geräte für den PC herausbringen zu wollen. Diese äußerst positive Entwicklung kann in der Zukunft dazu beitragen, dass der Umgang mit Tiefeninformationen intuitiver und effektiver wird.

3.2 Algorithmen

Es sind sehr viele unterschiedliche Ansätze für die 3D-Segmentierung in der Literatur zu finden. Viele dieser Verfahren unterscheiden sich jedoch nur beim Ähnlichkeitsmaß und in der Methodik der Berechnung der Zusammengehörigkeit der Punktdaten und Gruppierung. Nachdem ein Ähnlichkeitsmaß definiert ist, können Punkte auf Basis dessen untersucht und danach in verschiedene Segmente zusammengefasst werden. Zur finalen Gruppierung werden meist ein oder auch mehrere Schwellenwerte verwendet [Rabbani u. a., 2006]. Im Folgenden werden einige Ansätze genauer erläutert.

Kantenbasiert: Die kantenbasierte Segmentierung funktioniert genau wie in der Bildverarbeitung. Im ersten Schritt werden alle Kanten markiert. Dies geschieht auf Basis eines Unähnlichkeitsmaßes (*dissimilarity measure*). Hierbei werden einzelne verschiedene Regionen untersucht und innerhalb einer Region nach auffälligen Unterschieden gesucht. Es wird eine lokale Nachbarschaft nach einem lokalen Unähnlichkeitsmaß (meistens ein Schwellenwert) bewertet. Nachdem alle Kanten markiert sind, werden im zweiten Schritt die Bereiche innerhalb dieser Abgrenzungen zusammengefasst. Diese Gruppierungen entsprechen dann den einzelnen finalen Segmenten [Gächter u. a., 2006].

Als Maß für die Unähnlichkeit in der lokalen Nachbarschaft können Normalenvektoren [Sappa und Devy, 2001], die Hauptkrümmung (*principal curvature*) [Belton und Lichti, 2006] oder Gradienten [Bhanu u. a., 1986] verwendet werden.

Surface growing: „Surface growing“ auf Punktwolken ist das Äquivalent zu „Region growing“ auf 2D-Bildern. Aus der lokalen Nachbarschaft werden die Oberflächen-Eigenschaften (surface properties) berechnet. Diese werden als Ähnlichkeitsmaß benutzt, um ähnliche und beieinanderliegende Punkte in Segmente zusammenzufassen.

Es gibt zwei Vorgehensweisen bei diesem Verfahren: *Bottom-Up* und *Top-Down*.

Bottom-Up „von unten nach oben“: Ein einziger Punkt dient als Ausgangsbasis. Von diesem Punkt aus werden naheliegende Punkte, entsprechend des Ähnlichkeitsmaßes, zusammengefasst. Bei jedem Durchgang entsteht so ein neues Segment. Die Auswahl des Startpunktes ist besonders wichtig, da hiervon das Segmentierungs-Ergebnis abhängt.

Top-Down „von oben nach unten“: Bei dieser Technik wird im ersten Durchgang die gesamte Punktwolke wie ein Segment behandelt. Innerhalb dieses Segmentes wird dann wieder auf Basis eines Ähnlichkeitsmaßes entschieden, an welcher Stelle das Segment aufteilbar ist. Die Segmente werden solange aufgeteilt, bis keine Teilsegmente mehr gefunden werden.

Bottom-Up ist wesentlich einfacher umzusetzen als Top-Down. Top-Down eignet sich für eine gezielte Suche nach Objekten. Für die Segmentierung aller Objekte im Raum ist Bottom-Up besser geeignet. Als Ähnlichkeitsmaß lässt sich im einfachsten Fall die „Nähe von Punkten“ (*proximity of points*) verwenden. Dabei wird der Abstand zwischen den Punkten berechnet und mithilfe eines Schwellenwertes die Grenzen bestimmt. Eine weitere Möglichkeit ist es, die optimale Ebene (*best fitting plane*) für die Nachbarschaft zu berechnen und weitere Punkte nur aufzunehmen, wenn ihr orthogonaler Abstand zur Ebene einen bestimmten Schwellenwert nicht überschreitet [Vosselman u. a., 2004].

Eine sehr ausführliche Übersicht weiterer Möglichkeiten wurde in [Hoover u. a., 1996] zusammengestellt.

Scan-line-basiert: Dieses Verfahren ist auf 2.5D-Daten spezialisiert, da bei echten 3D-Daten nicht alle Informationen in einer Zeile dargestellt werden können (vgl. Kapitel 2.5).

Bei der Scan-Line-basierten Segmentierung werden die 3D-Daten in 2D-Zeilen aufgeteilt. Diese Zeilen können aus der horizontalen, vertikalen oder auch diagonalen Achse der 3D-Punkte entnommen und dann unabhängig voneinander segmentiert werden. Gefundene Segmente in jeder Zeile werden markiert. Zum Schluss werden diese wieder zusammengeführt. Die zuvor markierten Bereiche sollten dann die Kanten in der Szene darstellen. Der Vorteil dieses Verfahrens ist die Geschwindigkeit und die Möglichkeit, Algorithmen aus der Bildverarbeitung einzusetzen.

Im einfachsten Fall werden gerade Linien innerhalb der Zeile gesucht und Anfang und Ende der Linien als ein Segment markiert. Da alle Ebenen in der Punktwolke aus vielen beieinanderliegenden Geraden bestehen, lassen sich damit schnell Flächen, wie Wände und Decken, finden [Jiang und Bunke, 1994]. Auf Basis dieser Idee wurden auch einige ähnliche Verfahren aufgebaut [Sithole und Vosselman, 2003] [Khalifa u. a., 2003].

Ein Vergleich verschiedener Scan-Line-Algorithmen ist in [Nguyen u. a., 2007] beschrieben.

Schwellenwerte: Bei vielen Segmentierungs-Verfahren werden Schwellenwerte benötigt. In der Praxis reicht es oft, diese durch Experimente herauszufinden. Da alle Laserscanner spezifische Fehlertoleranzen besitzen, können die Autoren der Segmentierungs-Verfahren keine zuverlässigen Schwellenwerte für diese definieren. Einige Autoren haben jedoch Verfahren vorgestellt, womit das Finden dieser Schwellenwerte dynamisch und automatisch vorgenommen werden kann. In [Biosca und Lerma, 2008] wird mithilfe des *Fuzzy C-Means* (FCM) und des *Possibilistic C-Means* (PCM) Algorithmus automatisch eine Sortierung vorgenommen. In [Belton und Lichti, 2006] wurden mit dem Chi-Squared-Test automatisch verschiedene Oberflächen-Strukturen erkannt. Andere Clustering-Algorithmen sind

in [Xu und Ii, 2005] beschrieben.

Weitere Algorithmen, welche gleichzeitig Methoden aus mehreren Kategorien verwenden:

In [Filin, 2002] wurden sieben Merkmale beschrieben, auf deren Basis ähnliche Punkte gruppiert werden können. Dieser Merkmalsvektor eignet sich sehr gut, um verschiedene Clustering Algorithmen darauf anzuwenden.

In [Harati u. a., 2006] wird eine besonders interessante Repräsentation der Polardaten beschrieben. Dieses Verfahren nutzt die Eigenschaft von 2.5D-Daten aus und arbeitet sehr schnell. Mithilfe der Bildverarbeitung können alle Kanten in der Szene zuverlässig gefunden werden.

4 Konzept und Implementierung

4.1 Software

Die Robotik Abteilung des IAIS arbeitet schon viele Jahre mit verschiedenen Kameras, Sensoren und Motoren und hat dafür eine einheitliche Software-Bibliothek eingerichtet, die *FairLib*. Diese ist, wie die meisten Robotik-Bibliotheken, eine C++ Bibliothek. Sie besteht aus mehreren Modulen und ist objektorientiert aufgebaut.

Um Entwicklung und Demonstrationen zu vereinfachen, wurde die Entwicklungsumgebung auf eine virtuelle Maschine aufgesetzt. Dazu wurde die Virtualisierungslösung VMWare Workstation verwendet und als Betriebssystem Ubuntu 10.04 installiert.

Programmiert wird mit der Open Source Software Eclipse (Galileo) in Verbindung mit dem C/C++ Development Tools (CDT). Als Synchronisationswerkzeug dient ein Subversion-Server (SVN), welcher aus Eclipse heraus über das Subversion-Plugin Subclipse gesteuert werden kann. Als Compiler wird der GNU C Compiler g++ (Version 4.4.3) eingesetzt.

4.1.1 FairLib

Die FairLib ist eine C++ Bibliothek für Robotikanwendungen. Sie wurde speziell für Linux-Systeme konzipiert und bietet unter anderem die Schnittstellen, um den SICK-Scanner und den Servomotor anzusteuern. Die FairLib ist modular aufgebaut, und jede Komponente verfügt über spezielle Funktionen:

- FairCore: Basis-Datentypen (Matrix, Cloud3D, etc.), Input-Output-Kapselung (COM, USB, Sockets, etc.)
- FairDevices: Kapselung der Low-Level-Kommunikation (Kamera, 2D-Laserscanner, 3D-Laserscanner, etc.)
- FairAlgorithm: Filter/Wandler für 3D-Daten (Berechnung polar-kartesisch, etc.), Erkenner (ICP, SIFT, Vocus, etc.)
- FairGraphics: Darstellung von Bildern und 3D-Punktwolken (statisch und dynamisch)

Alle Komponenten bieten eine hohe Performance und wurden speziell für echtzeitfähige Anwendungen konzipiert.

Die Aufnahmefunktion und Speicherung der Ausgabedaten ist bereits implementiert, und es können beliebige Szenen mit dem Scanner aufgenommen werden. Dabei werden die Daten direkt in polarer und kartesischer Form gespeichert. Die beiden Objekte, welche den Zugriff auf diese Daten bereitstellen, werden kurz vorgestellt (s. dazu auch Abb. 4.1):

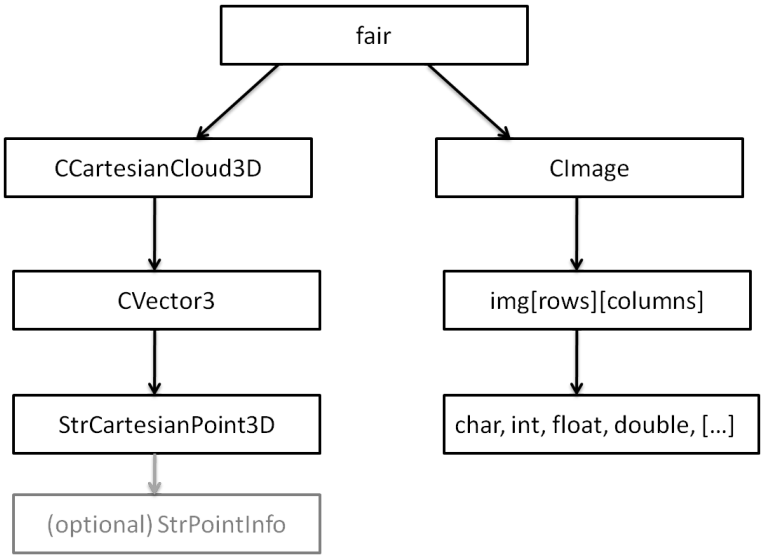

Abbildung 4.1: Aufbau CCartesianCloud3D und CImage

Fair::CCartesianCloud3D: Dieses Objekt stellt den Datencontainer für die kartesischen Koordinaten dar. Mithilfe der size()-Funktion kann über alle Datenpunkte innerhalb des Objektes iteriert werden. Die kartesischen Koordinaten werden über das Objekt

fair::CVector3 dargestellt. Diese Klasse beinhaltet Methoden zum Zugriff auf die x-, y- und z-Koordinaten, welche als fair::StrCartesianPoint3D gespeichert werden. Optional können jeder Koordinate auch über das Objekt fair::StrPointInfo verschiedene Eigenschaften zugewiesen werden, wie z. B. Farbe, Punktgröße oder ein Textfeld. Damit kann die Segmentierung und Klassifizierung festgehalten werden. Für die Visualisierung der kartesischen Daten gibt es einen 3D-Viewer, welcher die kartesische Punktwolke und ihre Zusatzinformationen, z. B. die Farbe von einzelnen Punkten, anzeigen kann.

Fair::CImage: Dieses Objekt stellt einen Basis-Datencontainer für primitive Datentypen (char, int, float, double) dar. Für diese Arbeit werden damit hauptsächlich die Polar- und Remissions-Werte gespeichert. Polare Distanzdaten werden in Zentimeter als Double- und die Remissions-Werte relativ zwischen Null (schwarz) und Eins (weiß) als Float-Werte abgespeichert. Ein iterativer Zugriff erfolgt hier durch eine doppelte For-Schleife, bei der

über die Spalten und Zeilen iteriert wird. Für die Visualisierung der Daten können die Werte im CImage auf Grauwerte (0 bis 255) konvertiert (z. B. Remissionswert * 255) und danach als Grauwert-Bild angezeigt werden.

4.1.2 Andere verwendete Bibliotheken

Funktionalitäten, welche nicht mit der FairLib abgedeckt sind, können mithilfe von anderen Bibliotheken ausgeglichen werden. Folgende C++ Bibliotheken werden hierfür benötigt:

- Newmat: Für Matrixoperationen (Matrix-Multiplikation, Least-Squares, Eigenwerte, etc.)
- ANN (Approximate Nearest Neighbor): Für die Berechnung der Nächsten Nachbarn (KD-Tree)
- OpenCV (Open Computer Vision): Methoden zur Bildverarbeitung (Glättungsfilter, Kantendetektion, Konturendetektion, etc.)

4.2 Programmierung

Im Folgenden werden drei ausgewählte Segmentierungs-Verfahren vorgestellt. Die ersten beiden Verfahren, der RANSAC-Algorithmus und die Nachbarschaftsanalyse, sind rein mathematische Verfahren. Das dritte Verfahren, die Repräsentation der Polardaten über den *Bearing Angle*, stellt eine komplett andere Vorgehensweise dar. Hierbei werden zur Segmentierung der 3D-Daten hauptsächlich Methoden der Bildverarbeitung eingesetzt.

Die in dieser Arbeit vorgenommene RANSAC-Implementierung sucht Ebenen in einer Punktwolke. Um die Gesamtlaufzeit zu verringern, wird ein neuer Parameter *abort* als Abbruchkriterium eingeführt. Bei der Nachbarschaftsanalyse werden die Informationen eines Datenpunktes über die Eigenschaften der lokalen Nachbarschaft erweitert. Zusätzlich werden Schwellenwerte über den FCM-Algorithmus ermittelt. Für die Oberflächen-Varianz und den Normalenvektor ist die Kombination mit dem FCM-Algorithmus noch in keiner Literatur beschrieben. Beim dritten Verfahren, Segmentierung über den *Bearing Angle*, werden keine eigenen Ideen angewandt, sondern nur das von den Autoren beschriebene Vorgehen umgesetzt.

4.2.1 Random Sample Consensus

Der Random Sample Consensus Algorithmus (RANSAC), zu deutsch „Übereinstimmung mit einer zufälligen Stichprobe“, ist einer der ältesten und einfachsten Algorithmen, um geometrische Objekte in einer Punktwolke zu finden. Es ist ein iterativer Algorithmus,

welcher besonders effektiv auf verrauschten Daten, also Daten mit vielen Ausreißern, arbeitet. Die Ergebnisse sind nicht-deterministisch, da in jeder Iteration zufällige Stichproben ausgewählt werden. Durch Erhöhung der Iterationen, und somit der Laufzeit, erhöht sich jedoch die Wahrscheinlichkeit, gleiche und sehr genaue Ergebnisse zu erhalten [Fischler und Bolles, 1981].

Grundgedanke des Algorithmus ist, dass die Daten aus Einlieger- und Auslieger-Punkten (inlier und outlier) bestehen. Die Einlieger-Punkte sind eine Masse von Datenpunkten, welche durch eine geometrische Funktion ausgedrückt werden können. Die Auslieger-Punkte sind Ausreißer, also Daten-Punkte, welche außerhalb der gesuchten geometrischen Funktion liegen. Der Vorteil dieser Unterscheidung von Einlieger- und Auslieger-Punkten kann gut durch den Vergleich mit dem Least-Squares-Algorithmus verdeutlicht werden. In Abbildung 4.2 ist eine Punktwolke dargestellt, in der deutlich eine Linie, bestehend aus sehr vielen Punkten, zu sehen ist. Jedoch gibt es auch viele Datenpunkte um diese Linie herum. Für den Fall, dass eine gerade Linie in dieser Punktwolke gesucht wird, könnten diese Daten nun in Einlieger- und Auslieger-Punkte aufgeteilt werden. Wenn jedoch der Least-Squares-Algorithmus auf die Punktwolke angewandt wird, werden alle Punkte (auch die Ausreißer) gleich gewichtet, und es wird eine Geradengleichung gewählt, die nur die kleinsten Abstände zu allen Punkten hat. Der RANSAC-Algorithmus hingegen versucht, die Zahl der Punkte, welche durch die Geradengleichung eingeschlossen werden, zu maximieren.

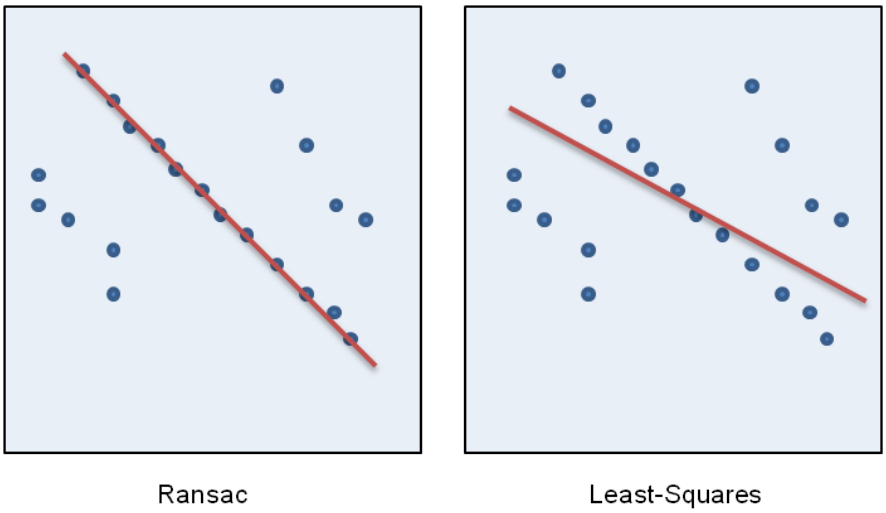

Abbildung 4.2: RANSAC vs. Least-Squares

Die einfachste geometrische Funktion im dreidimensionalen Raum ist die Ebenengleichung. Eine Ebene kann durch den Normaleneinheitsvektor oder über die Ebenengleichung bestimmt werden (s. Kapitel 2.3). Als Ergebnis würden alle ebenen großflächigen Bereiche gesucht werden, wie z.B. Wände, Decken und Böden. Doch der Unterschied zu dem vorherigen Beispiel im zweidimensionalen ist, dass hier nun mehrere Ebenen innerhalb ei-

ner Punktwolke vorhanden sind. Damit mehr als eine Ebene zuverlässig gefunden werden kann, muss der RANSAC-Algorithmus wiederholt aufgerufen und es müssen entsprechende Abbruchkriterien für einen Durchlauf definiert werden.

Eine Implementierung des originalen RANSAC-Algorithmus zum Finden einer Ebene innerhalb einer Punktwolke, wurde bereits für die FairLib in [Linder, 2010] realisiert. Dazu wurde die Datenstruktur fair::StrPlane3D eingeführt, welche es ermöglicht, eine Ebene im kartesischen Koordinatensystem zu definieren und danach für andere Koordinaten mit der Methode getDistance() den orthogonalen Abstand zu dieser zu erhalten. Auf diese Weise ist es möglich, den Abstand aller Datenpunkte zu einer beliebigen Ebene im Raum zu berechnen und zu minimieren. Die vorhandene Implementierung kann jedoch noch nicht auf echte Laserscanner-Daten angewandt werden, weil dort mehrere Ebenen (Wände, Decken, Böden) vorhanden sind. Damit diese in einer Szene gefunden werden können, muss die Zielfunktion verändert werden. Anstatt den Abstand der Ebene zu allen Datenpunkten zu minimieren, muss nach einer großen Menge von Punkten gesucht werden, die optimal in eine Ebenengleichung passen. Als Modifikation des RANSAC-Algorithmus wird eine neue Variable *abort* eingeführt. Diese soll dann zum Abbruch führen, wenn eine „sinnvolle Ebene“ gefunden wurde.

Die vorgenommene RANSAC-Implementierung ist als Pseudo-Code in Algorithmus 1 dargestellt.

data stellt die Datenpunkte vom Laserscanner dar, und $function$ ist eine beliebige geometrische Funktion. Diese Funktion entscheidet, welches geometrische Objekt in der Punktwolke gesucht werden soll. Der Parameter n ist abhängig von der gesuchten geometrischen Funktion, bei der Suche nach Ebenen ist $n = 3$. Die Anzahl der maximalen Iterationen *steps* entscheidet gleichermaßen über die Genauigkeit der Ergebnisse wie über die Laufzeit. Bei wenigen Iterationen können auch nur wenige Stichproben ausgewertet werden, und dafür wird folglich auch weniger Zeit benötigt. Die Laufzeit steigt linear mit der Anzahl der Iterationen, da für jede Stichprobe der Abstand aller Datenpunkte zum resultierenden Modell berechnet werden muss.

Die Variable *error* stellt die Fehlertoleranz dar, also den maximalen erlaubten Abstand von dem (aus der Funktion resultierendem) Modell. Der systematische Fehler beim SICK-Laserscanner beträgt einen Zentimeter, und daher entwickelt sich ein recht starkes Rauschen innerhalb der Punktwolke. Eine geometrische Form kann nicht über die Schnittmenge seiner Funktion in der Punktwolke gefunden werden und der systematische Fehler des Scanners muss in Form eines Schwellenwertes mitbeachtet werden. Da neben dem systematischen Fehler des Scanners auch noch eine leicht gewölbte Struktur bei großen Flächen durch die Neigevorrichtung entsteht, wurde *error=5cm* als optimaler Wert ermittelt.

Algorithmus 1 RANSAC-Algorithmus

Input:
data - Menge von Datenpunkten
$function$ - Geometrische Funktion (ohne Parameter)
n - Minimale Anzahl benötigter Parameter für die Funktion
steps - Anzahl Iterationen
error - Fehlertoleranz, maximaler Abstand von der Funktion
precision - Präzision, minimale Anzahl Datenpunkte (damit das Modell akzeptiert wird)
abort - Abbruchkriterium, es wurde ein gutes Modell gefunden

Output:
best_model - Funktions-Parameter, welche optimal zu den Daten passen
best_points - Datenpunkte, welche zum *best_model* passen
best_error - Kleinster Fehlerwert (mittlerer quadratischer Abstand)

```
best_error := ∞
while Iteration ≤ steps do
  maybe_points := {}
  maybe_model := Parametrisiere function mit n zufälligen Datenpunkten aus data
  for all d ∈ data do
    if maybe_model.getDistance(d) ≤ error then
      maybe_points.add(d)
      this_error += Berechne orthogonalen Abstand von maybe_model zu d
    end if
  end for
  if maybe_points.size() ≥ precision then
    if this_error < best_error then
      best_model := maybe_model
      best_points := maybe_points
      best_error := this_error
    end if
  end if
  if best_points.size() ≥ abort then
    break
  end if
end while

return best_model, best_points, best_error
```

Mit *precision* wird die Suchgenauigkeit eingestellt. Die Variable gibt an, wie viele Datenpunkte mindestens einem Modell zugewiesen werden müssen, damit dieses akzeptiert wird. Über einen großen *precision*-Wert kann sichergestellt werden, dass nur große signifikante Oberflächen gefunden werden. Es ist wichtig, den Wert groß genug zu wählen – entsprechend dem Rauschen in der Punktwolke. Falls er zu klein gewählt wird, resultiert dies in falschen Ergebnissen, da zufällige Konstellationen von Datenpunkten dann zu schnell akzeptiert werden. Für die vorhandene Punktwolke wurde *precision=5%* ausgewählt.

Die Variable *abort* ist Teil der Modifikation des Original RANSAC-Algorithmus. Hier wird wieder die Genauigkeit, also die Anzahl der gefundenen und zum Modell zugewiesenen Datenpunkte, ausgewertet. Diesmal wird darüber jedoch ein Abbruchkriterium definiert, nämlich für den Fall, dass so viele Punkte gefunden werden, dass diese mit großer Sicherheit zu der Menge der gesuchten Oberflächen gehört. Nützlich ist dies vor allem bei Wänden, Decken und Böden, also bei der Suche nach linearen Funktionen. Diese machen einen großen Teil der Punktwolke aus. Je schneller diese erkannt und aus der Punktwolke entfernt sind, desto schneller kann die ganze Szene untersucht werden. Mit *abort=20%* konnten in den vorhandenen Punktwolken die ersten zwei bis vier Ebenen in jeweils unter 200 Iterationen gefunden und damit bereits bis zu 50% der gesamten Datenpunkte zugeordnet werden.

Für die Implementierung wurden folgende Klassen eingesetzt:

- **fair::CCartesianCloud3D**: Datenstruktur für kartesische Datenpunkte
- **fair::StrPlane3D**: Datenstruktur für die Erzeugung von Ebenen
- **fair::DistanceEuklid**: Euklidische Distanzfunktion

Algorithmus 1 liefert genau ein Modell zurück in jedem Durchlauf. Um alle Flächen in einer Szene zu finden, muss er wiederholt aufgerufen werden. Jedesmal, wenn ein Modell zurückgeliefert wird, können die damit verbundenen Datenpunkte aus der Gesamtmenge *data* entfernt werden. Dadurch braucht der Algorithmus im nächsten Durchlauf weniger Zeit, da es weniger Abstände zu den Datenpunkten zu berechnen gibt. Die Komplexität des RANSAC-Algorithmus hat sich durch die Modifikation nicht verändert und ist $O(m * n)$, wobei m der Anzahl von Iterationen (Variable *steps*) und n der Anzahl von Datenpunkten (*data*.size()) entspricht. Die verwendeten Punktwolken (s. Abb. 4.3) haben eine Größe von ca. 200.000 Punkten. Bei $steps = 500$ Iterationen müssen also $500 * 200.000 = 100$ Millionen Operationen durchgeführt werden, um die erste Ebene in der Punktwolke zu finden.

4.2.2 Nachbarschaftsanalyse

Über die Nachbarschaftsanalyse ist es möglich, die Informationen eines Datenpunktes zu erweitern. Im Original besteht ein Datenpunkt nur aus einer x-, y- und z-Koordinate. Wird

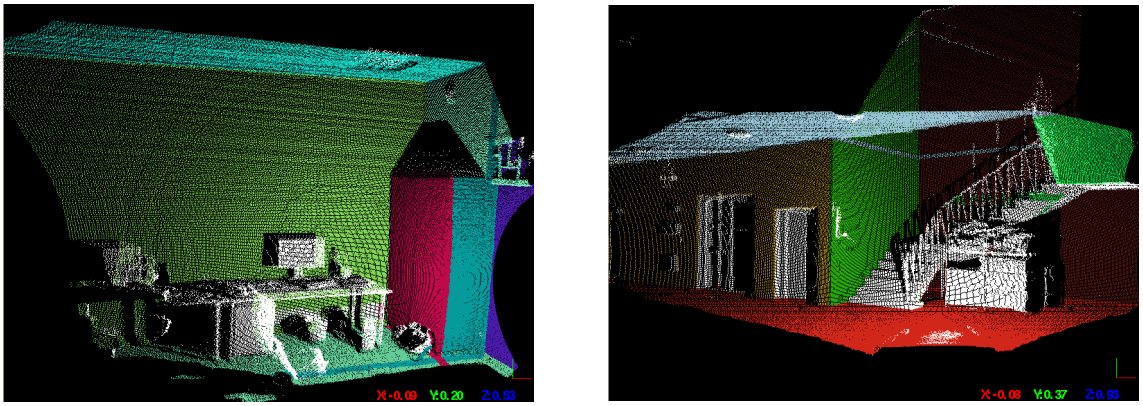

Abbildung 4.3: RANSAC-Ergebnisse: Büro und Flur

der Datenpunkt relativ zu seiner Nachbarschaft betrachtet, können Zusatzinformationen über diesen gesammelt werden. Je mehr Informationen über einen Datenpunkt gesammelt werden können, desto sicherer kann später eine Aussage über seine Funktion in der Gesamt-Punktwolke gemacht werden. In [Filin, 2002] und [Belton und Lichti, 2006] wurden einige Merkmale vorgestellt, über die eine grobe Segmentierung der Szene möglich ist. Die Genauigkeit der Ergebnisse ist bei der Nachbarschaftsanalyse besonders abhängig von der Genauigkeit der 3D-Daten. Bei stark verrauschten Daten müssen relativ viele Nachbarn einbezogen werden, damit eine zuverlässige Aussage über die Struktur der Nachbarschaft gemacht werden kann. Jedoch wird durch die Hinzunahme von mehr Nachbarn auch das Ergebnis ungenauer, da feine Konturen in der Menge verwischt werden. Dies ist vergleichbar mit den Faltungsfiltern aus der Bildverarbeitung: Ein 5 x 5-Filter verwischt wesentlich mehr Details als ein 3 x 3-Filter.

Da der verwendete SICK-Scanner eine relativ große Fehlertoleranz hat, ist davon auszugehen, dass hier eine große Anzahl von Nachbarn betrachtet werden muss. Der Fovea-Scanner hingegen, auf dem die Algorithmen später eingesetzt werden sollen, wird wesentlich genauere Ergebnisse liefern.

Um den Umgang und die Fehlersuche bei den verschiedenen Algortihmen zu erleichtern, wurde von Hand ein künstliches Modell erstellt. Dieses besteht aus mehreren Ebenen (in horizontaler und vertikaler Ausrichtung), einem Viertel-Kreis und verschiedenen 90°-Winkeln. Das Modell ist in Abbildung 4.4 dargestellt: Die Punktwolke besteht aus sieben Segmenten, wobei jedes Segment aus 10.000 Punkten besteht. Somit hat die Gesamtpunktwolke eine Größe von 70.000 Punkten. Dadurch, dass kein Rauschen in den Daten auftritt, sollte die Implementierung und das Testen der Nachbarschafts-Algorithmen erleichtert werden. Ziel ist es, die verschiedenen Abschnitte in der Punktwolke zu segmentieren. So sollen die Ebenen, Rundungen und Kanten mithilfe der Nachbarschaft als solche erkannt werden.

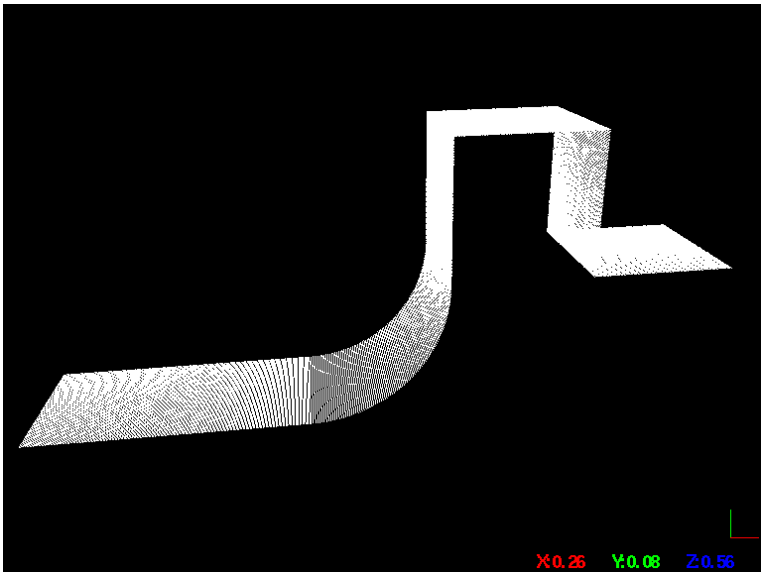

Abbildung 4.4: Mathematisch erzeugtes Modell ohne Ausreißer

Höhendifferenz zur Nachbarschaft

In Algorithmus 2 wird für jeden Datenpunkt der Abstand zur optimalen Ebene (die durch die Nachbarschaft aufgespannt wird) berechnet und als eindimensionaler Merkmalsvektor in *data_height* gespeichert. Die optimale Ebene wird über alle n ausgewählten Werte in der Nachbarschaft von d berechnet. Dies kann über den Least-Squares-Algorithmus oder über die Eigenwerte vorgenommen werden. Von dieser erzeugten Ebene kann nun der orthogonale Abstand zu dem Datenpunkt d berechnet und als neuer Merkmalsvektor in d abgespeichert werden. Auf diese Weise wird für jeden Datenpunkt ein neuer Merkmalsvektor berechnet. Benachbarte Punkte können ganz unterschiedliche Höhenwerte besitzen, da die Nachbarschaft für jeden Punkt anders ist. Ein niedriger Höhenwert lässt auf eine Ebene und ein großer Höhenwert auf eine Kante in der Nachbarschaft schließen.

Um das Ergebnis des Algorithmus auszuwerten, müssen die Daten in *data_height* mit den Datenpunkten in *data* verglichen werden. Für diesen Zweck wurde ein statischer Schwellenwert ausgewählt und die Datenpunkte dementsprechend in zwei Gruppen aufgeteilt. Dann wurde jedem Datenpunkt entsprechend seiner Gruppe eine Farbe zugewiesen. Der Schwellenwert kann für das Modell schlicht sehr niedrig gewählt werden, da in den ebenen Bereichen eine Höhendifferenz von Null zu erwarten ist. Bei realen Szenen hingegen, wo durch die Fehlertoleranz des Scanners viele Ausreißer entstehen, muss dieser Schwellenwert durch Experimentieren abgeschätzt und optimiert werden. In Algorithmus 3 ist das Vorgehen, für diese einfache und schnelle Möglichkeit die Ergebnisse zu visualisieren, beschrieben.

Algorithmus 2 Merkmalsvektor: Höhendifferenz - Abstand zur optimalen Ebene

Input:
data - Menge von Datenpunkten
n - Anzahl nächste Nachbarn

Output:
data_height - Merkmalsvektor für die Höhendifferenz in der Nachbarschaft (eindimensional)

data_height := {}
for all $d \in data$ **do**
  neighbors := Berechne die n nächsten Nachbarn für d
  plane := Berechne die optimale Ebene von *neighbors*
  data_height.add(Berechne orthogonalen Abstand von *plane* zu d)
end for

return *data_height*

Algorithmus 3 Visualisierung: Merkmalsvektoren über einen Statischen Schwellenwert

Input:
data - Menge von Datenpunkten
feature_vector - Eindimensionaler Merkmalsvektor
threshold - Schwellenwert bezogen auf *feature_vector*

Output:
data_colored - Farbzuweisung für jeden Datenpunkt

for all $d \in data$ & $v \in feature\_vector$ **do**
  if $v <= threshold$ **then**
    data_colored := Färbe Datenpunkt d Rot
  else
    data_colored := Färbe Datenpunkt d Grün
  end if
end for

return *data_colored*

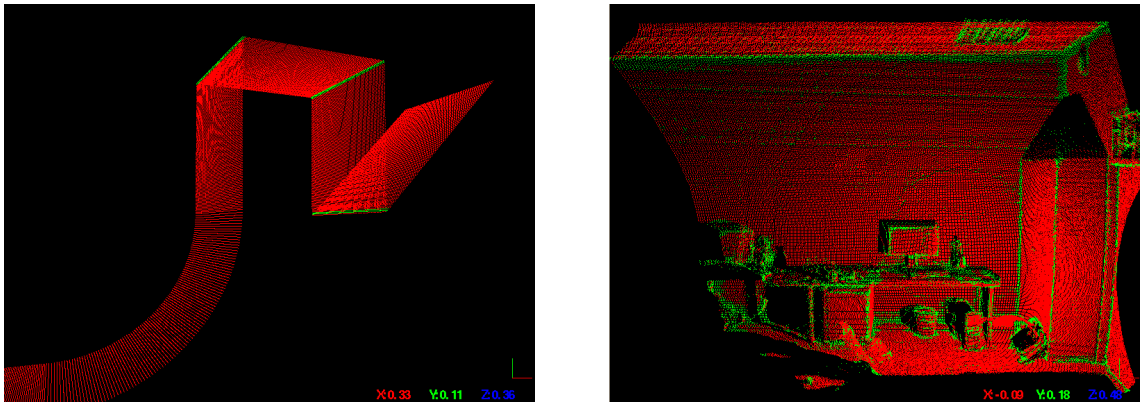

Abbildung 4.5: Nachbarschaftsanalyse: Abstand zur optimalen Ebene

Die Kombination aus Algorithmus 2 und 3 wurde auf das künstlich erzeugte Modell und eine reale Szene angewandt (s. Abb. 4.5). Bei der Berechnung der Höhendifferenz auf dem Modell wurden acht Nachbarn ausgewählt, sodass eine 3 x 3-Nachbarschaft untersucht wurde. Als Schwellenwert für die Visualisierung wurde 0,1 cm ausgewählt. Auch bei der Auswertung einer größeren Nachbarschaft und von kleineren Schwellenwerten wurden stets nur die Kanten markiert. Die stetige, runde Steigung kann damit noch nicht erkannt werden. Dies hat den Vorteil, dass über diesen speziellen Merkmalsvektor Kanten zuverlässig geprüft werden können. Ein Test auf der realen Szene bestätigt diese Vermutung: Trotz der vielen Ausreißer in den realen Daten können die Kanten relativ fein (auf etwa fünf bis acht Pixel genau) aufgelöst werden.

Oberflächen-Varianz

Ein weiterer Merkmalsvektor, der für jeden Datenpunkt über seine Nachbarschaft berechnet werden kann, ist die Oberflächen-Varianz. Diese wird oft in der Bildverarbeitung eingesetzt und beschreibt die Varianz in Richtung der Normalen in Relation zu der Varianz in der Nachbarschaft (s. Formel 2.6). In Algorithmus 4 wird diese für alle Datenpunkte berechnet und in einem eindimensionalen Vektor *data_variance* abgespeichert.

Algorithmus 4 Merkmalsvektor: Oberflächen-Varianz

Input:
data - Menge von Datenpunkten
n - Anzahl nächste Nachbarn

Output:
data_variance - Merkmalsvektor für die Oberflächen-Varianz in der Nachbarschaft

data_variance := {}
for all $d \in data$ **do**
  neighbors := Berechne die n nächsten Nachbarn für d
  eigenvalues := Berechne die Eigenwerte von *neighbors*
  $surface\_variance$:= Berechne die Oberflächenvarianz aus den Eigenwerten (s. Formel 2.6)
  data_variance.add($surface\_variance$)
end for

return *data_variance*

Der resultierende Merkmalsvektor wurde wieder mit Algorithmus 3 über einen statischen Schwellenwert mit den Datenpunkten kombiniert. Wie in Abbildung 4.6 zu sehen ist, werden diesmal ebene Bereiche und nicht-ebene Bereiche zuverlässig unterschieden. Kanten und stetige Steigungen sind in grün dargestellt. Diese auffälligen Bereiche wurden erfolgreich über die Oberflächen-Varianz erkannt. Im Gegensatz zu dem optimalen Modell muss in der realen Szene immer eine größere Nachbarschaft untersucht werden, um die signifikanten Bereiche zu finden. Für die Szene in Abbildung 4.6 wurden $n = 75$ nächste

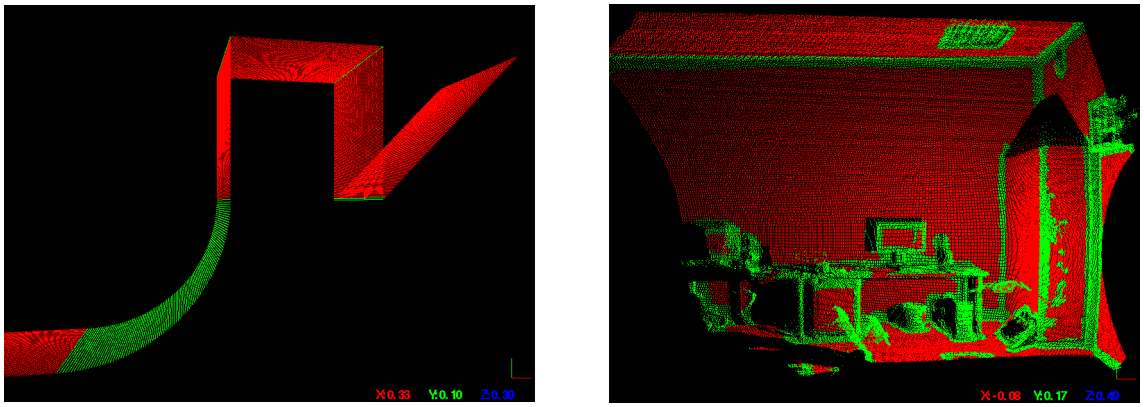

Abbildung 4.6: Nachbarschaftsanalyse: Oberflächen-Varianz

Nachbarn ausgewählt. Bei der Auswahl von weniger Nachbarn kommt es zu ungenauen Ergebnissen. Dies hängt jedoch stark von der Punktwolke und deren Dichte ab.

Algorithmus 5 Merkmalsvektor: Varianzanalyse auf der Oberflächen-Varianz

Input:
data - Menge von Datenpunkten
data_variance - Oberflächenvarianz
n - Anzahl nächste Nachbarn

Output:
data_varvar - Merkmalsvektor für die gefilterten Varianzwerte

data_varvar := {}
for all $d \in$ *data* **do**
  neighbors := Berechne die n nächsten Nachbarn für d
  varvar := Berechne die Varianz von *neighbors* (s. Formel 2.4)
  data_varvar.add(*varvar*)
end for

return *data_varvar*

Die Ergebnisse aus Algorithmus 3 können weiterhin genutzt werden, um die Genauigkeit um die markierten Bereiche herum zu verbessern [Belton und Lichti, 2006]. Der Merkmalsvektor *data_variance* hat bereits erfolgreich alle ebenen Bereiche markiert. Die Oberflächen-Varianz in diesen Bereichen ist sehr klein ($data\_variance < 0.001$). Die Werte in den Bereichen, wo Kanten und Unebenheiten auftreten, liegen relativ eng beieinander und sind deutlich größer ($data\_variance > 0.01$). Um nun aus diesen Daten feinere Kanten zu extrahieren, kann eine weitere Varianzanalyse auf dem eindimensionalen Merkmalsvektor *data_variance* vorgenommen werden. Diesmal soll jedoch die „Varianz in der Nachbarschaft“ (s. Formel. 2.4) berechnet werden. Das Ziel dabei ist, Bereiche mit stetiger Krümmung zu markieren. Solche Bereiche sollten nämlich einen auffällig hohen Varianz-

wert erzeugen. Der Mittelwert wird dabei jedesmal aus der Nachbarschaft heraus ermittelt und nicht über die Gesamtmenge der Werte. Dies ist wichtig, da die Dichte der Punktwolke nach hinten heraus abnimmt. Dadurch, dass der Mittelwert immer aus der Nachbarschaft ermittelt wird, wird die Empfindlichkeit auch adaptiv angepasst. Dies führt zu besseren Ergebnissen. Auch kann nun eine wesentlich kleinere Nachbarschaft von $n = 20$ Punkten ausgewählt werden, da die Ausreißer bereits im vorherigen Schritt eliminiert wurden. Algorithmus 5 zeigt das Vorgehen, um über *data_variance* einen neuen Merkmalsvektor *data_varvar* zu erzeugen.

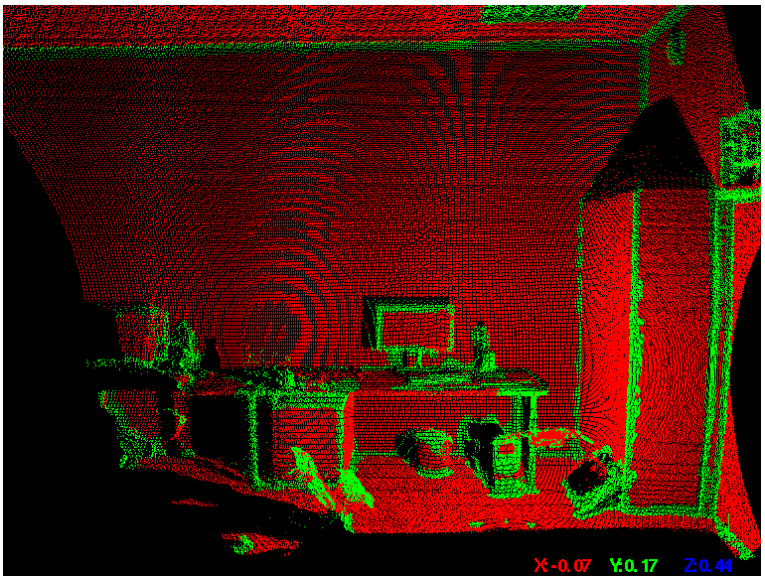

Abbildung 4.7: Varianzanalyse auf der Oberflächen-Varianz

In Abbildung 4.7 ist das Ergebnis der Varianzanalyse auf der Oberflächen-Varianz dargestellt. Dadurch, dass n so klein gewählt werden konnte, sind Kanten und unebene Bereiche wesentlich genauer erfasst worden als vorher. Allerdings werden Kanten zwar zuverlässig, aber sehr großzügig dargestellt. Dies kann auch nicht durch Verkleinern von n verbessert werden, da sonst Lücken bei den Kanten entstehen.

Aussenränder

Ein weiteres Informations-Tupel kann über die Eigenwerte berechnet werden, indem der größte Eigenwert vom zweitgrößten subtrahiert wird ($\lambda_3 - \lambda_2$). Die Differenz der beiden größten Eigenwerte beschreibt die Streuung, bzw. die Geometrie in der Nachbarschaft. Falls die Verteilung gleich ist, und in alle Richtungen die nächsten Nachbarn gesucht werden können, wird die Menge von Nachbarn einen Kreis um den untersuchten Datenpunkt bilden, da die nächsten Nachbarn über den kürzesten Abstand zu dem Datenpunkt ausgewählt wurden. Wird allerdings eine Rand-Koordinate ausgewählt, so gibt es in eine Richtung

Algorithmus 6 Merkmalsvektor: Aussenränder

Input:
data - Menge von Datenpunkten
n - Anzahl nächste Nachbarn

Output:
data_boundary - Merkmalsvektor für Außenpunkte

data_boundary := {}
for all $d \in data$ **do**
  neighbors := Berechne die n nächsten Nachbarn für d
  eigenvalues := Berechne die Eigenwerte von *neighbors*
  boundary := Berechne $\lambda_3 - \lambda_2$ (für die Eigenwerte $\lambda_1 < \lambda_2 < \lambda_3$)
  data_boundary.add(*boundary*)
end for

return *data_boundary*

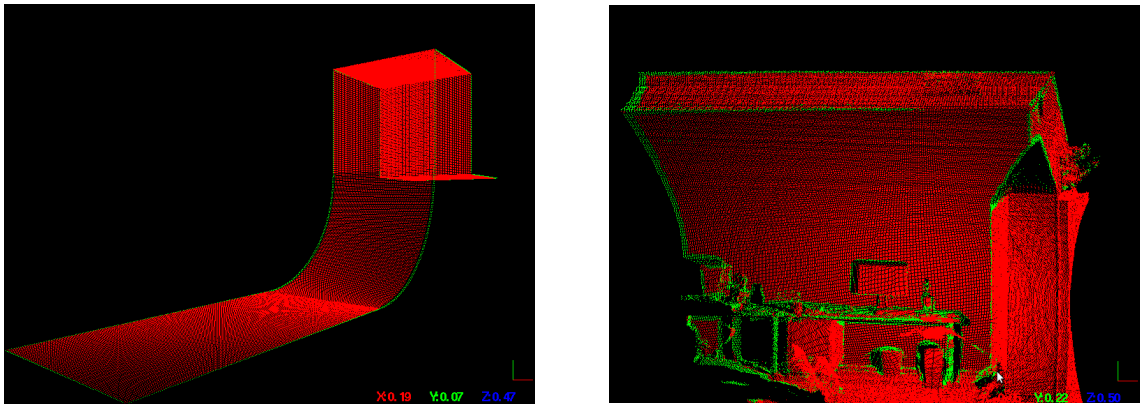

Abbildung 4.8: Erkennung der Aussenränder

keine nächsten Nachbarn mehr, und die Verteilung der ausgewählten Nachbarn bildet eine ovale Struktur. Über die beiden größten Eigenwerte kann leicht bestimmt werden, ob eine runde Struktur ($\lambda_3 - \lambda_2 = 0$) oder eine ovale Struktur ($\lambda_3 - \lambda_2 > 0$) vorliegt. In Algorithmus 6 ist die Berechnung der Werte beschrieben.

In Abbildung 4.8 wurde der Algorithmus auf das Modell und die reale Szene angewandt und visualisiert.

Dynamische Schwellenwerte über Fuzzy-Clustering

Ein wesentliches Problem bei der Visualisierung der Ergebnisse war bisher das Finden der Schwellenwerte. Diese wurden durch Experimentieren ermittelt. Dies ist aber äußerst zeitaufwendig und wird noch schwieriger für den Fall, dass mehrere Schwellenwerte sinnvoll gesetzt werden können. In [Biosca und Lerma, 2008] werden die Schwellenwerte in den Merkmalsvektoren über den Fuzzy-C-Means-Algorithmus gefunden. Dieser berechnet in

mehreren Iterationen den Abstand von den Zentroiden zu den Datenwerten und bewegt die Zentroide in Richtung der Datenwerte, die besonders häufig auftreten. Vom Benutzer muss nur die Anzahl an zu suchenden Clustern bzw. Schwellenwerten angegeben werden. Die Anzahl an Iterationen bestimmt die Genauigkeit. Für die 200.000 Datenpunkte in der realen Szene reichen bereits 100 Iterationen. Der Merkmalsvektor kann beliebige Dimensionen haben, hierfür muss nur die Distanzfunktion angepasst werden. Algorithmus 7 beschreibt den Aufbau von FCM.

In Abbildung 4.9 wurde der FCM-Algorithmus auf die Oberflächen-Varianz-Werte aus Algorithmus 4 angewandt. Da der Merkmalsvektor *data_variance* eindimensional ist und aus Zahlenwerten besteht, kann der euklidische Abstand als Distanzfunktion verwendet werden. Die Zuweisung der Farben kann über den Membership-Score erfolgen.

Algorithmus 7 Schwellenwerte: Automatisch über Fuzzy-C-Means

Input:
vector - Merkmalsvektor
steps - Anzahl an Iterationen
num_clusters - Anzahl zu unterscheidender Cluster

Output:
centroids - Zentroide der gefundenen Cluster

centroids := Erzeuge *num_clusters* zufällige Zentroide
while Iteration $\leq$ *steps* **do**

  for all $v \in$ *vector* **do**
    for all $c \in$ *centroids* **do**
      distance := Berechne Abstand von v zu c (Distanzfunktion abhängig von *vector*)
    end for
  end for

  for all $v \in$ *vector* **do**
    for all $c \in$ *centroids* **do**
      membership := Berechne den Membership-Score von v und c
    end for
  end for

  for all $c \in$ *centroids* **do**
    c := Berechne die Position von c neu mit Hilfe des Membership-Score
  end for

end while

return *centroids*, *membership*

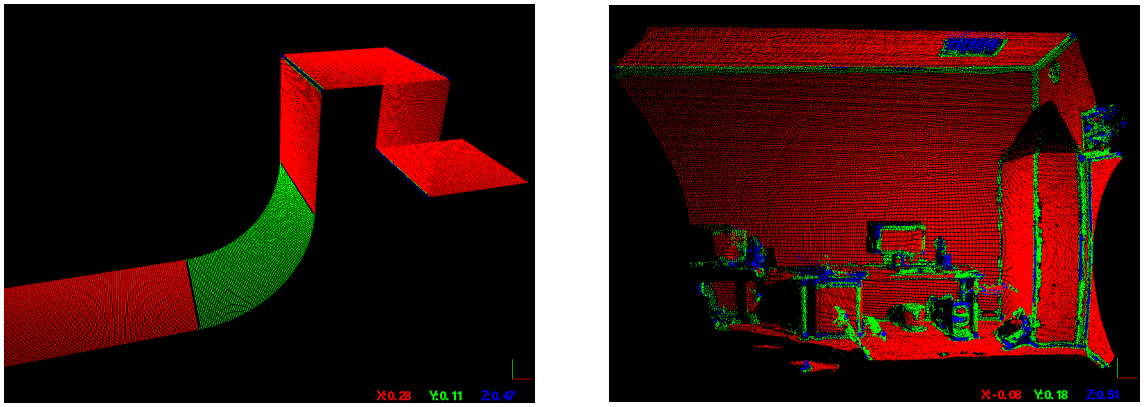

Abbildung 4.9: Automatische Unterscheidung verschiedener Oberflächen

Normalenvektor

Das letzte Informations-Tupel, welches schon oft genannt wurde, ist der Normalenvektor. Er kann verwendet werden, um Punkte, die annähernd in der gleichen Ebene liegen, zu bestimmen. Dafür wird der Normalenvektor aus lokalen Nachbarschaften berechnet. Bei vielen kleinen, im Raum verstreuten Objekten sind keine sinnvollen Rückschlüsse allein mithilfe des Normalenvektors zu erwarten. Wird allerdings der FCM-Algorithmus auf die Werte angesetzt, werden große Flächen und Grenzen (z. B. zwischen Decke und Wand) sehr fein erkannt. Der FCM-Algorithmus muss hierbei mit einem dreidimensionalen Merkmalsvektor umgehen. Dazu muss die Distanzfunktion angepasst werden. Das Ergebnis des FCM-Algorithmus, angewandt auf die Normalenvektoren des Modells, und eine reale Szene, sind in Abbildung 4.10 dargestellt.

Algorithmus 8 Merkmalsvektor: Normalenvektor

Input:
data - Menge von Datenpunkten
n - Anzahl nächste Nachbarn

Output:
data_normals - Merkmalsvektor für Normalenvektoren

data_normals := {}
for all $d \in$ *data* **do**
  neighbors := Berechne die n nächsten Nachbarn für d
  plane := Berechne die optimale Ebene aus *neighbors*
  normals := Speichere den Normalenvektor von *plane*
  data_normals.add(*normals*)
end for

return *data_normals*

Für die Implementierung wurden folgende Klassen eingesetzt:

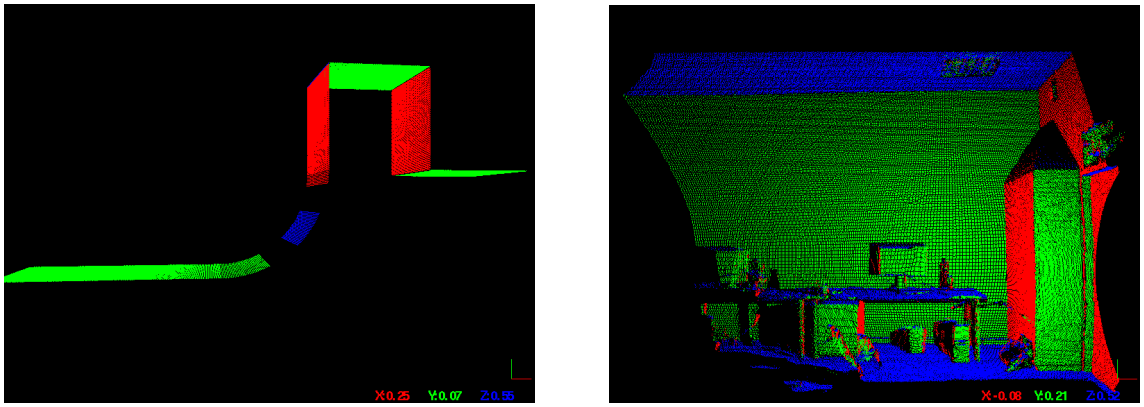

Abbildung 4.10: Clustering über den Normalenvektor

- **fair::CCartesianCloud3D**: Datenstruktur für kartesische Datenpunkte
- **fair::StrPlane3D**: Datenstruktur für die Erzeugung von Ebenen
- **fair::getDistanceEuklid**: Euklidische Distanzfunktion
- **ANN::ANNkd_tree**: Datenstruktur für die Erzeugung eines KD-Tree (Nächste Nachbarn)
- **Newmat::Matrix**: Mathematische Operationen auf Datenpunkten
- **Newmat::EigenValues**: Berechnung der Eigenwerte

4.2.3 Bearing Angle

Eine weitere Möglichkeit zur Segmentierung wurde in [Harati u. a., 2006] [Harati und Siegwart, 2007] und [Harati, 2008] beschrieben. Bei diesem Verfahren wird direkt auf den Polarkoordinaten gearbeitet und die Szene über ein kantenbasiertes Binärbild segmentiert. Die Grundlage dieses Verfahrens ist der sogenannte *Bearing Angle* (zu deutsch: Lagewinkel). Mit diesem können die Objekte in einer Szene wesentlich farbintensiver und kontrastreicher als auf den Distanz- und Remissionsbildern kenntlich gemacht werden. Als Vorraussetzung müssen die bei der Aufnahme eingestellten Scanner-Parameter bekannt sein. In Kapitel 2 wurde bereits beschrieben, wie der Scanner die Szene von links nach rechts in 0,25° Schritten aufnimmt und der Servomotor für jede Zeile um 0,25° nach oben inkrementiert. Mit diesen Informationen können die horizontalen und vertikalen *Bearing Angle* berechnet werden. Daraus ergeben sich dann zwei neue Repräsentationen der Szene. Hierzu werden jeweils zwei nebeneinander liegende Koordinaten miteinander verrechnet. Da die polaren Koordinaten die Distanzen enthalten und der Winkel zwischen diesen beiden Koordinaten bekannt ist, kann mit der Formel 4.1 der auftreffende Winkel auf der Oberfläche ermittelt werden. Die Formel nutzt schlicht den Satz des Pythagoras aus. Da die Länge von zwei

Kanten und weiterhin ein Winkel des Dreiecks bekannt sind, kann damit auch der zweite Winkel des Dreiecks berechnet werden. In Abbildung 4.11 ist das Vorgehen und der *Bearing Angle* skizziert: Dabei stellen p_i und p_{i-1} die beiden polaren Koordianten dar, ω ist der vom Scanner inkrementierte Winkel, und φ_h bzw. φ_v ist der resultierende *Bearing Angle*.

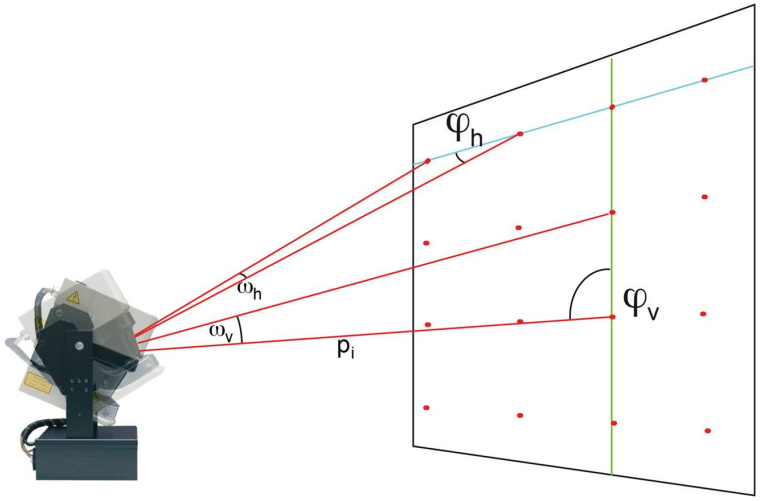

Abbildung 4.11: Bearing Angle φ_h bzw. φ_v [IAIS, 2011]

$$BA_i = arccos \frac{p_i - p_{i-1} cos\omega_i}{\sqrt{p_i^2 + p_{i-1}^2 - 2p_i p_{i-1} cos\omega_i}} \tag{4.1}$$

Nachdem diese Berechnung zweimal getrennt voneinander für die Zeilen und Spalten vorgenommen wurde, werden als Ergebnis zwei unabhängige Bilder mit der gleichen Auflösung wie die Originale (eine Spalte bzw. eine Zeile weniger, da immer Paare berechnet werden) ausgegeben. Als Werte enthalten diese Bilder nun Winkelangaben, genauer gesagt, die Lagewinkel auf der Oberfläche. Um diese zu visualisieren, kann der HSV-Farbraum genutzt werden. Dieser erlaubt es, den Farbton über das Winkelmaß zu definieren. Auf diese Weise können die zuvor berechneten Winkelangaben direkt auf einen Farbraum abgebildet werden. Die anderen beiden Farbkanäle neben dem Farbton sind die Sättigung und der Helligkeitswert: Diese werden dabei auf 100% gesetzt. In Abbildung 4.12 wurde die Büro-Szene nun in die *Bearing Angle*-Repräsentation überführt und auf dem HSV-Farbraum abgebildet. Es gibt einen starken Qualitätsunterschied zwischen den beiden Bildern. Das liegt daran, dass bei dem linken Bild die Zeilen und im rechten Bild die Spalten für die Berechnung des *Bearing Angle* genommen wurden. Eine Zeile wurde jeweils in einem Durchgang von links nach rechts vom Scanner aufgenommen. Für die Spalten jedoch wurde jedesmal der Servomotor bewegt. Dieser ist offensichtlich nicht so präsize. Denn es fällt in dieser Repräsentation verstärkt auf, dass die Spalten nicht an einem Stück aufgenommen wurden, sondern erst über die Summe der Zeilen entstanden. Die Repräsentation zeigt jedoch

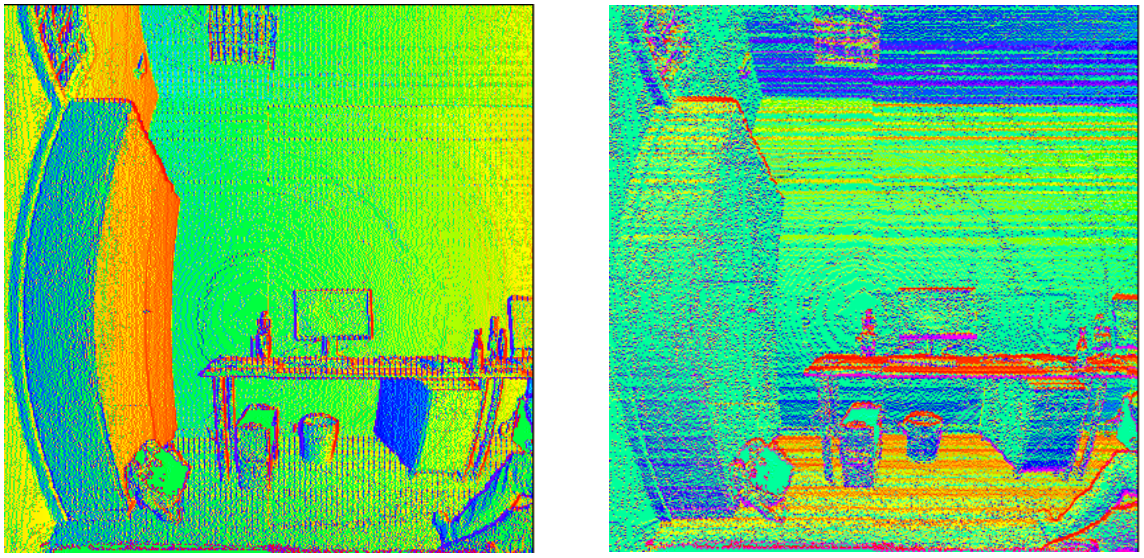

Abbildung 4.12: Bearing Angle auf dem HSV-Farbraum

alle erwarteten Eigenschaften. Das linke Bild, welches zeilenweise berechnet wurde, zeigt sehr starke Farbunterschiede ín horizontaler Richtung. Das rechte Bild, welches spaltenweise berechnet wurde, zeigt diese Farbunterschiede in der vertikalen Richtung. Aus der Kombination beider Bilder lassen sich mit bloßem Auge bereits sehr genau Wände, Decken und Böden unterscheiden. Dies gelingt wesentlich besser als über das Distanz- oder Remissionsbild.

Es ist gut zu erkennen, dass stets in die Richtung, in der der *Bearing Angle* berechnet wurde, horizontal oder vertikal ein Farbverlauf entsteht. Gerade Ebenen verändern ihre Farben von links nach rechts. Das ist in der Büro-Szene zu sehen, in der die gegenüberliegende Wand von grün ins gelbliche wechselt. Der Farbwechsel entsteht durch die Bewegungsrichtung des Scanners. Je weiter dieser sich von links nach rechts bewegt, desto kleiner wird der aufgespannte Winkel, welcher hier als Farbwert dargestellt wird. Die Bewegungsrichtung des Scanners ist allerdings berechenbar, und somit kann der entstehende Farbverlauf ausgeglichen werden. Dazu wurde der sogenannte *Running Angle* eingeführt. Dieser setzt auf dem *Bearing Angle* auf und normiert die berechneten Winkel. Dies geschieht mithilfe des Vorwissens. Da der Winkel bekannt ist, der in jedem Durchlauf inkrementiert wird, kann die immer größer werdende Summe dieser Winkel auf den *Bearing Angle* addiert werden, um den mit der Bewegungsrichtung des Scanners kleiner werdenden *Bearing Angle* auszugleichen. In Abbildung 4.13 ist eine Scanner-Zeile dargestellt. Für φ_1 wurde der *Running Angle* RA_1 als $BA_1 + 0 * \omega$, für φ_2 als $BA_2 + 1 * \omega$, usw. berechnet.

$$RA_i = BA_i + \Phi_i \tag{4.2}$$

$$\text{mit } \Phi_i = (i-1) * \omega$$

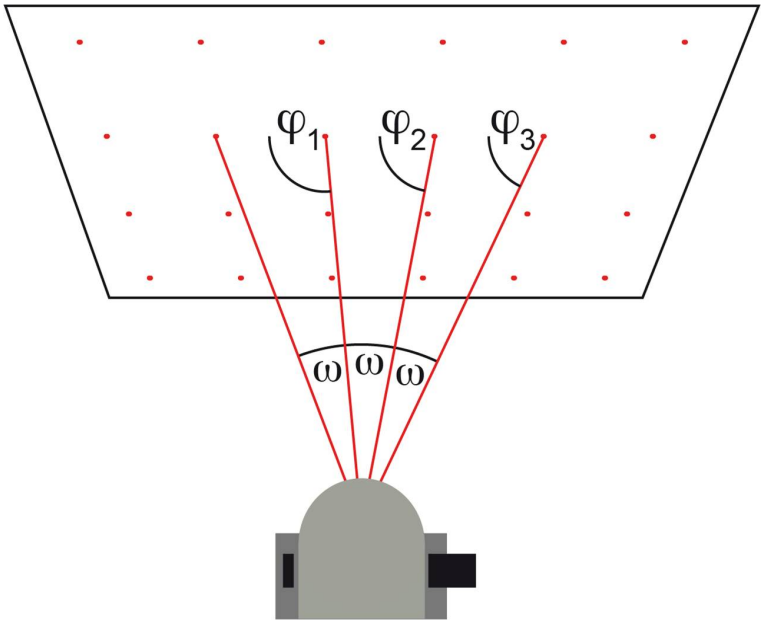

Abbildung 4.13: Running Angle φ_i [IAIS, 2011]

In Abbildung 4.14 wurde die zuvor erzeugte Repräsentation mithilfe des *Running Angle* neu berechnet. Das Ergebnis ist ein Bild, in dem die Farbübergänge eliminiert wurden und ebene Flächen mit der gleichen Ausrichtung auch im gleichen Farbton dargestellt werden. Die Kontraste zwischen den verschiedenen Oberflächen sind jedoch ähnlich kräftig wie zuvor. Es ist zu sehen, dass der Schrank und die anliegende Wand nun den gleichen Orange-Ton haben. Dies liegt daran, dass der Schrank parallel zur Wand steht und somit genau den gleichen auftreffenden Winkel wie die Wand erzeugt.

Über die *Running Angle*-Bilder ist es möglich, alle Kanten in der Szene zu finden. Die Kontraste sind so verstärkt worden, dass aus der Kombination der beiden Bilder alle signifikanten Flächen unterschieden werden können. Jedoch sind die Bilder noch so stark verrauscht, dass eine Kantendetektion viele falsche Stellen markieren würde. Die Bilder können nicht beliebig oft gefiltert werden, um das Rauschen zu entfernen. Würde der Gauss-Filter verwendet werden, könnten wichtige Pixel so verwischt werden, dass sich Übergänge verschieben. Um dies zu vermeiden, sollte eine Filtertechnik eingesetzt werden, die die Daten nicht verändert. Wie in Kapitel 2.5 beschrieben, sollten für die Glättung von Laserscanner-Daten anisotrope Filter eingesetzt werden. Daher wurde die Haar-Transformation ausgewählt. Indem das Bild mit dieser herunterskaliert wurde, wurden alle verrauschten Stellen eliminiert und es blieben nur die wirklich wichtigen Übergänge erhalten. Dies ist vor allem für die vertikale Repräsentation des *Running Angle* wichtig, da hier das Rauschen übermäßig stark auftritt. Mit der folgenden Formel kann das Bild beliebig oft herunterskaliert werden. Die Skalierung kann jedoch immer nur in jeweils eine Richtung vorgenommen werden – entweder horizontal oder vertikal.

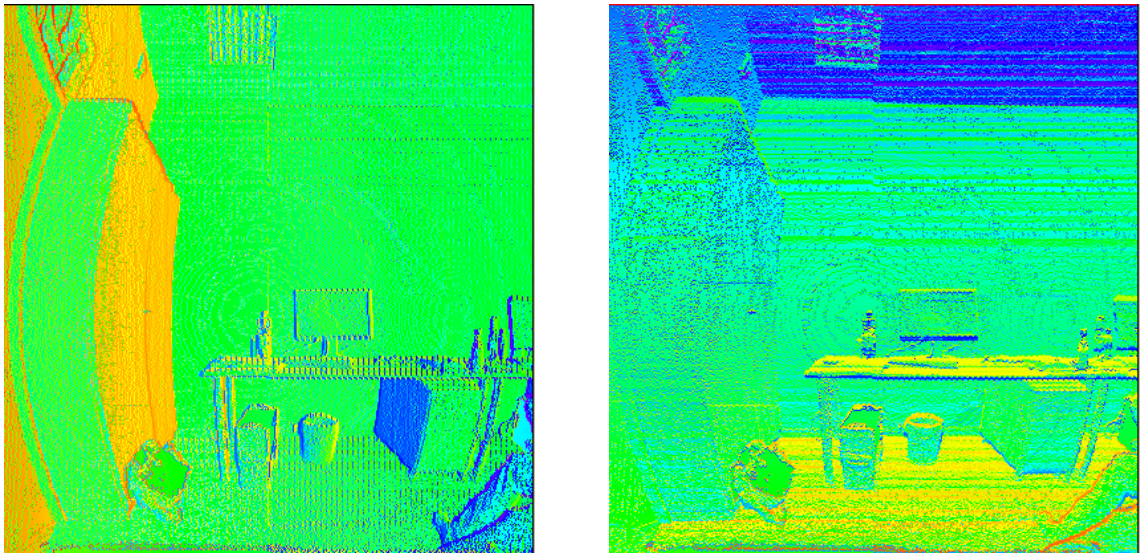

Abbildung 4.14: Running Angle auf dem HSV-Farbraum

$$s_l = s_{2l} + d_l/2 \tag{4.3}$$

$$\text{mit } d_l = s_{2l+1} - s_{2l}$$

In Abbildung 4.15 wurde die Haar-Transformation spaltenweise auf das vertikale *Running Angle*-Bild angewandt. Zur besseren Visualisierung wurde das Bild diesmal als Grauwertbild ausgegeben. Der Farbraum ist allerdings unabhängig von den Pixelwerten zu betrachten, da nur der Farbraum verändert wurde, nicht jedoch die Werte.

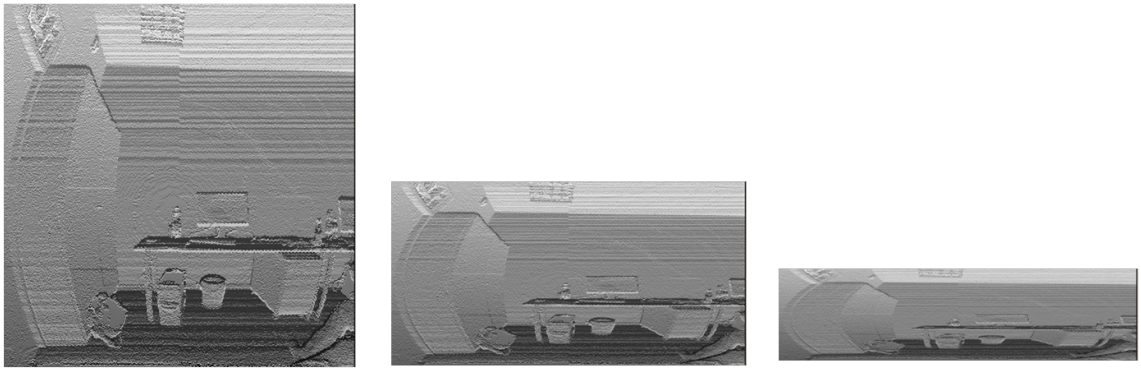

Abbildung 4.15: Spaltenweise Haar-Transformation auf vertikalem Running-Angle-Bild

Analog dazu wurde auf dem horizontalen *Running Angle*-Bild die zeilenweise Haar-Transformation angewandt.

Über die verschiedenen Skalierungen des Bildes ist es nun möglich, die gesuchten Kanten zuverlässig mit dem Sobel-Filter zu bestimmen. Dazu wird dieser in einer 3 x 3-Matrix eingesetzt und die resultierenden Werte über einen Schwellenwert binarisiert. Auf diese Weise

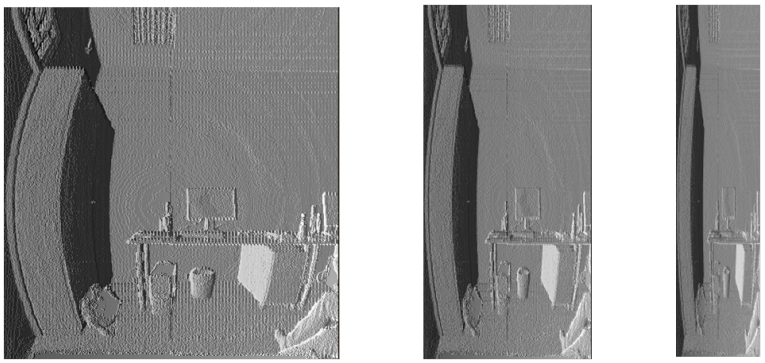

Abbildung 4.16: Zeilenweise Haar-Transformation auf horizontalem Running-Angle-Bild

werden alle Bilder verarbeitet. Es ist besonders wichtig, dass die Kanten später geschlossen werden können. Daher wird ein großer Schwellenwert auf das gefilterte Bild angewendet. Dicke geschlossene Kanten sind nämlich sinnvoller als feine unvollständige Kanten. Es wird ein Schwellenwert von 60° (die Pixelwerte können immer noch mit Gradwerten verglichen werden) auf das Bild gelegt. Hierdurch werden die Kanten eindeutig kenntlich gemacht. Die Binärbilder mit den verschiedenen Skalierungen müssen nun wieder vereint werden. Da in den echten Daten das Rauschen mit der Größe des Bildes abnimmt, sollte die Schnittmenge der Kanten ausgewählt werden. Die Binärbilder, welche aus dem vertikalen *Running Angle* entstanden sind, und das Ergebnis sind in Abbildung 4.17 dargestellt. Es ist gut zu erkennen, wie das Rauschen durch die mehrfache Skalierung verschwand. Bei der Berechnung der Schnittmenge muss natürlich der jeweilige Skalierungsfaktor berücksichtigt werden. So werden beim Vergleich für die Schnittmenge jeweils zwei, vier und acht Pixel (in der höchsten Skalierungsstufe) des Originals mit einem Pixel im herunterskalierten Bild verglichen.

Die so gefundenen Kanten werden in der Literatur *Curve Edges* bzw. *Turn Edges* genannt. Sie beschreiben einen Richtungswechsel auf einer fortlaufenden Oberfläche. Eine zweite Gruppe von Kanten entsteht durch Objekte, die mitten im Raum positioniert sind. Diese werden einvernehmlich als *Jump Edges* bezeichnet. Sie können gut detektiert werden, da sie isoliert im Raum stehen. Für die Polarkoordinaten heißt das, dass ein signifikanter Distanzunterschied plötzlich auftritt und genauso plötzlich wieder verschwindet. Über den *Bearing Angle* können *Jump Edges* besonders gut detektiert werden, da ein signifikanter Distanzunterschied in den Polarkoordinaten durch einen sehr steilen Winkel von zwei benachbarten Werten ausgedrückt wird.

$$Jump = \{BA_i \approx \Phi_{Jump} \ \& \ |p_i - p_{i-1}| > d_{min}\} \tag{4.4}$$

Dabei werden für Φ_{Jump} 175° - 185° und 355° - 5° als Grenzwerte gewählt. Der Abstand d_{min} zwischen zwei Koordinaten p_i und p_{i-1} sollte mindestens einen Zentimeter betragen.

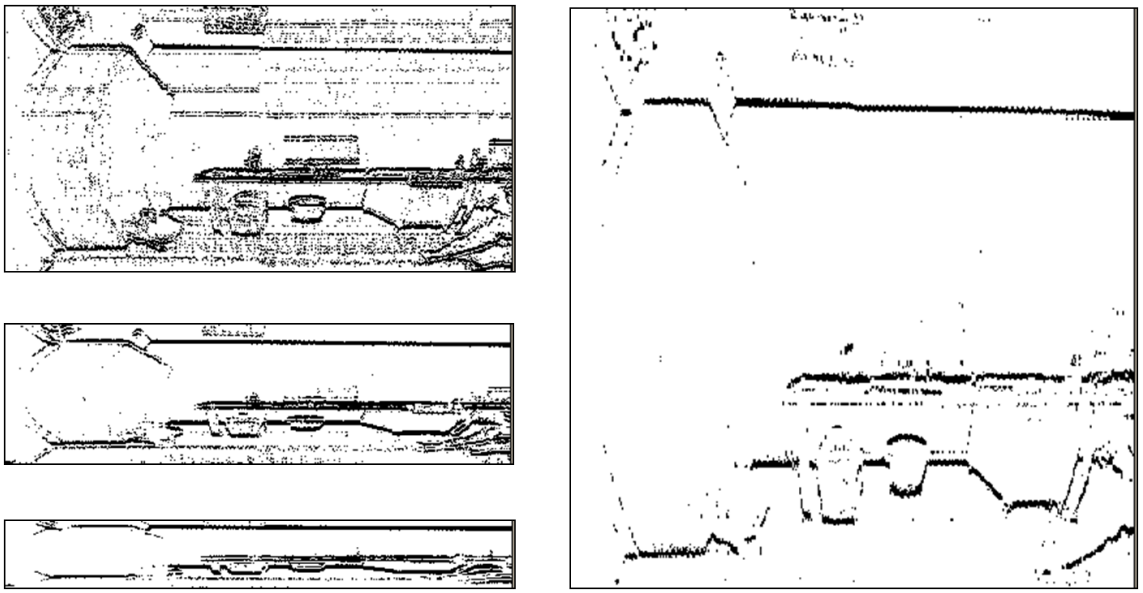

Abbildung 4.17: Turn Edges: Links die drei Skalierten Binärbilder und rechts die Schnittmenge dieser

Nach dieser Formel können wieder beide *Bearing Angle* Bilder in horizontaler und vertikaler Richtung berechnet werden. Das Ergebnis der Büro-Szene ist in Abbildung 4.18 dargestellt.

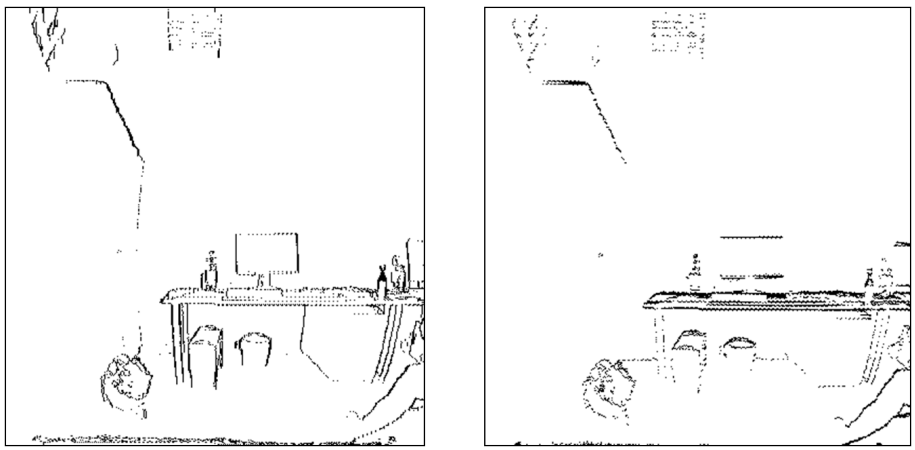

Abbildung 4.18: Jump Edges: Abgeleitet vom horizontalen (links) und vertikalen (rechts) Bearing-Angle-Bild

Schließlich wird aus den vier entstandenen Kantenbildern die Vereinigung gebildet. Dabei werden die Bilder übereinander gelegt und sämtliche Werte von allen Bildern in ein einziges Bild übertragen. Dieses Bild enthält nun alle in der Szene gefundenen Kanten und ist somit die Basis für die Segmentierung. In Abbildung 4.19 sind alle vier Kantenbilder und das Endresultat dargestellt.

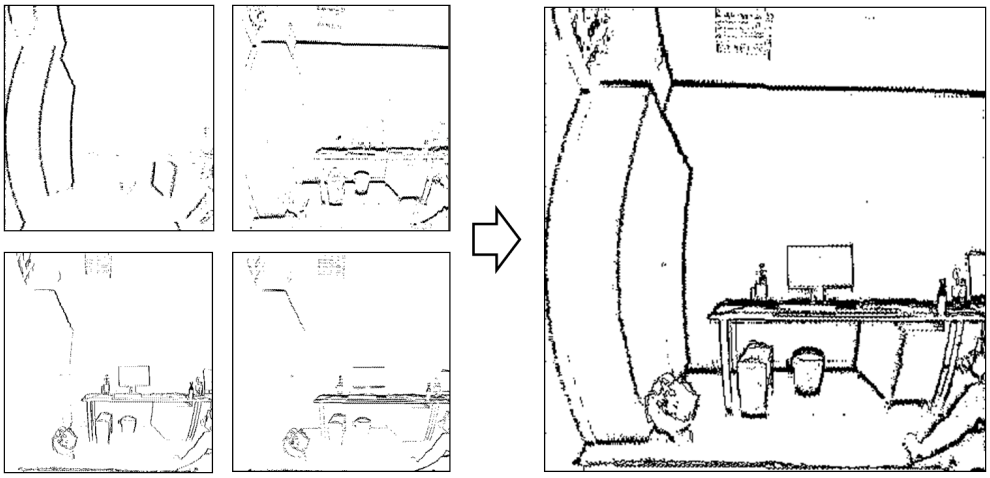

Abbildung 4.19: Zwischenergebnisse (links) und Endergebnis (rechts) der Kantendetektion

Bevor die einzelnen Segmente endgültig voneinander getrennt sind und farbig markiert werden können, muss nun noch das morphologische Schließen (Dilatation und Erosion) der Kanten vorgenommen werden. Das sorgt dafür, dass kleine Öffnungen bei unvollständigen Kanten geschlossen werden. Gleichzeitig werden die Kanten etwas aufgebläht durch diese Operation. Für die Segmentierung ist es aber wichtig, dass die Kanten zu 100% geschlossen sind, damit ein Segment auch isoliert dargestellt werden kann. Bei kleinsten Öffnungen zu anderen Segmenten würden dadurch falsche Segmente als zusammengehörig klassifiziert werden. Die Genauigkeit des Endergebnis würde darunter leiden. Es wurden daher die Kanten verstärkt aufgebläht und dadurch die Zuverlässigkeit erhöht. Abschließend wurde das Kantenbild mit einem Konturen-Such-Algorithmus in die verschiedenen Segmente unterteilt und farbig markiert. In Abbildung 4.20 ist die segmentierte Szene dargestellt. Um die Optik zu verbessern und die vielen kleinen entstandenen Flächen zu entfernen, wurde noch zusätzlich der Medianfilter auf dem Endergebnis ausgeführt.

Für die Implementierung wurden folgende Klassen eingesetzt:

- **fair::CCartesianCloud3D**: Datenstruktur für kartesische Datenpunkte
- **fair::CImage**: Datenstruktur für polare Datenpunkte
- **OpenCV::cvFindContours**: Detektion von Konturen in Binärbildern
- **OpenCV::cvMorphologyEx**: Morphologische Operationen zum Schließen von unvollständigen Konturen (Dilatation und Erosion)
- **OpenCV::cvSmooth**: Glättungsfilter (hier als Median-Filter eingesetzt)

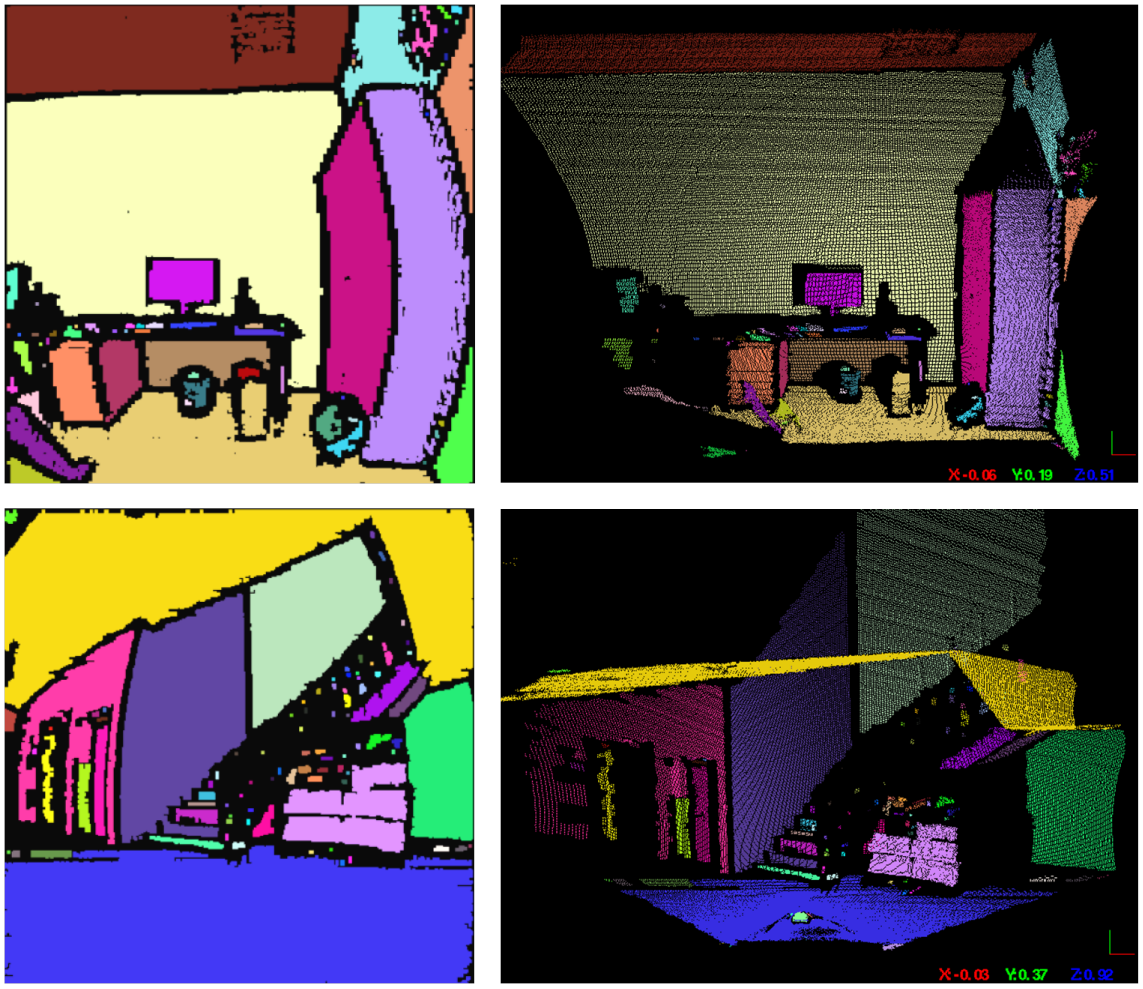

Abbildung 4.20: Polare (links) und kartesische Repräsentationen (rechts) der segmentierten Szenen

Algorithmus 9 Segmentierung über den Bearing Angle

Input:
data - Polare Koordinaten einer Szene

Output:
segmented_data - Segmentierte Szene

Phase 1: Bearing- und Running-Angle
for all *rows* ∈ *data* **do**
  BA_horizontal := Berechne den Bearing Angle zeilenweise
  RA_horizontal := Berechne den Running Angle aus *BA_horizontal*
end for

for all *columns* ∈ *data* **do**
  BA_vertical := Berechne den Bearing Angle spaltenweise
  RA_vertical := Berechne den Running Angle aus *BA_vertical*
end for

Phase 2: Haar-Transformation
HAAR_horizontal := Berechne die ersten drei Haar-Transformationen aus *RA_horizontal*
HAAR_vertical := Berechne die ersten drei Haar-Transformationen aus *RA_vertical*

Phase 3: Kantendetektion
edge_horizontal_1 := Berechne die Kanten aus *RA_horizontal* mit dem Sobel-Filter
edge_horizontal_2 := Berechne die Kanten der drei Bilder in *HAAR_horizontal*
edge_horizontal_binary := Vereinige Kantenbilder und berechne Binärbild über Schwellenwert

turn_edge_horizontal := Berechne die Schnittmenge aus den Binärbildern
jump_edge_horizontal := Markiere Stellen aus *RA_horizontal* mit Winkeln um 0 oder 180 Grad
edge_horizontal := Berechne die Vereinigung aus *turn_edge_horizontal* und *jump_edge_horizontal*

Berechne *edge_vertical* analog

edges := Berechne die Vereinigung aus *edge_horizontal* und *edge_vertical*

Phase 4: Konturendetektion
edges_closed := Führe morphologisches Schließen auf *edges* aus (Erosion und Dilatation)
conturs := Finde alle geschlossenen Konturen in *edges_closed*
segmented_data := Markiere alle Konturen aus *conturs* unterschiedlich

return *segmented_data*

5 Auswertung

In diesem Kapitel werden einige einfache Experimente beschrieben, um eine zuverlässige Aussage über Qualität und Laufzeit der einzelnen Segmentierungs-Verfahren machen zu können. Die Algorithmen wurden alle auf dem gleichen Computer ausgeführt (Intel Core2Duo 8400 mit 3.0 GHz, 4 GB DDR2 Speicher und SATA Festplatte).

5.1 Ergebnisse

Für die Evaluation der verschiedenen Verfahren und Ergebnisse mussten unterschiedliche Methoden und Modelle eingesetzt werden. Im Folgenden werden daher alle Ergebnisse differenziert betrachtet und ausgewertet.

5.1.1 RANSAC

Der RANSAC-Algorithmus wurde in dieser Arbeit leicht modifiziert. Die Komplexität des Algorithmus hat sich dadurch nicht verändert. Sie ist weiterhin $O(m * n)$, wobei m der Anzahl an Iterationen und n der Anzahl an Datenpunkten entspricht.

Der RANSAC-Algorithmus arbeitet mit Zufallszahlen und daher sind die Ergebnisse nicht deterministisch. Mit steigender Anzahl der Iterationen sollte die Genauigkeit ebenfalls steigen. Die Qualität der Ergebnisse des implementierten Algorithmus wird hauptsächlich durch die Anzahl der Iterationen bestimmt. Die anderen Parameter (*error=5cm*, *precision=5%*, *abort=25%*) sind stark von dem Rauschfaktor und der Auflösung der vorhandenen Punktwolken abhängig. Für die vom SICK-Scanner erzeugten Punktwolken, wurden diese in Kapitel 4 bestmöglich bestimmt.

Für die Evaluation wurde die Büro-Szene ausgewählt. Es gibt elf signifikante Ebenen in dieser Szene: Drei Außenwände, eine Decke, ein Boden, zwei Schrankflächen, zwei Unterschrankflächen, ein Monitor und eine Tischoberseite. Für die Auswertung in Tabelle 5.1 wurde der RANSAC-Algorithmus mehrfach aufgerufen und das Ergebnis manuell ausgewertet. Es wurden die richtig gefundenen (richtig positiv) und die falsch gefundenen Flächen (falsch positiv) gezählt. Zusätzlich wurde noch der Mittelwert und die Standardabweichung aus fünf unabhängigen Durchläufen ermittelt.

Mit diesem Ergebnis lassen sich mehrere Aussagen über den Algorithmus machen. Zum einen zeigt der Verlauf der richtig positiv erkannten Flächen, dass die Genauigkeit nicht mit der Anzahl der Iterationen steigt – wie es erwartet worden wäre. Im Bereich zwischen 500

Tabelle 5.1: RANSAC: Auswertung der Büro-Szene

Iterationen	Laufzeit	Mittelwert / Standardabw.	richtig positiv	falsch positiv
2500	67,49 Sek	63,60 Sek / 12,76 Sek	8 von 11	0 von 11
1500	49,34 Sek	35,34 Sek / 11,67 Sek	9 von 11	0 von 11
1000	27,78 Sek	26,04 Sek / 4,42 Sek	8 von 11	1 von 11
800	20,21 Sek	21,10 Sek / 3,94 Sek	8 von 11	0 von 11
500	14,80 Sek	14,03 Sek / 2,78 Sek	9 von 11	0 von 11
350	8,86 Sek	9,85 Sek / 1,47 Sek	7 von 11	2 von 11
200	7,15 Sek	5,94 Sek / 1,30 Sek	6 von 11	1 von 11
100	4,32 Sek	3,67 Sek / 0,54 Sek	6 von 11	1 von 11
50	2,84 Sek	2,61 Sek / 0,60 Sek	5 von 11	3 von 11

und 2500 Iterationen kann kein signifikanter Qualitätsunterschied festgestellt werden. Die Qualität kann in diesem Bereich nicht weiter über die Anzahl der Iterationen bzw. über die Laufzeit verbessert werden. Unterhalb von 500 Iterationen sinkt die Qualität zwar spürbar, allerdings sind die Ergebnisse im Bereich von 50 bis 100 Iterationen nicht so schlecht, wie sie in Relation zur Laufzeit zu erwarten wären.

Der subjektive Eindruck der Ergebnisse ist gut bis befriedigend. Einerseits können Flächen zuverlässig in kurzer Zeit gefunden werden, andererseits sind die Einflüsse durch die Zufallsvariable deutlich zu erkennen. Bei sich schneidenden Ebenen fällt besonders negativ auf, dass diese bei jedem neuen Durchlauf unterschiedlich zugewiesen werden. Je nachdem, welche Ebene zuerst gefunden wird, bekommt diese einen Teil der anderen Ebene zugewiesen. Dies ist ein grundsätzliches Problem des Algorithmus und kann nur schwer eliminiert werden.

5.1.2 Nachbarschaftsanalyse

Über die Nachbarschaftsanalyse sollen Informationen über die Lage eines Datenpunktes gesammelt werden. Je nach Einsatzzweck kann es nützlich sein, kleinere Regionen in der Punktwolke zu klassifizieren und diese als Ausgangspunkt für weitere Untersuchungen zu verwenden. In dieser Arbeit wurden grundsätzlich mathematische Verfahren eingesetzt, um auf diese Weise zwischen verschiedenen Merkmalen unterscheiden zu können. Die mit diesen mathematischen Verfahren erzeugten Merkmalsvektoren sollen in erster Linie als Grundlage und Inspiration für weitere Segmentierungs-Verfahren dienen. Es ist auch denkbar, die mathematischen Verfahren als Post-Processing-Methoden einzusetzen, um Ergebnisse die z. B. über die Bildverarbeitung entstanden sind, zu überprüfen. Der größte Nachteil bei der konsequenten Berechnung der Nachbarschaft ist die resultierende Laufzeit. Die Komplexität ist $O(m * n)$, wobei m der Anzahl der Nachbarn und n der Anzahl der Datenpunkte

Tabelle 5.2: Nachbarschaftsanalyse: Laufzeiten bei 200.000 Datenpunkten (Büro-Szene)

Nachbarn	Laufzeit
200	20,64 Sek
175	17,32 Sek
150	15,75 Sek
125	11,23 Sek
100	9,30 Sek
75	7,42 Sek
50	5,14 Sek
25	4,47 Sek
15	3,13 Sek
5	2,04 Sek

entspricht. In Tabelle 5.2 ist die Laufzeit für verschiedene Nachbarschaftsdichten aufgelistet. Daraus lässt sich erkennen, dass eine Echtzeit-Verarbeitung der Daten mit diesen Methoden nicht möglich ist.

Zu der Qualität und dem Nutzen der einzelnen Verfahren kann noch Folgendes ausgesagt werden:

- Kantenerkennung durch Höhendifferenz in der Nachbarschaft: Dieses Verfahren ist positiv durch die Genauigkeit aufgefallen (fünf bis acht Pixel genau, bei 75 Nachbarn). Bei den anderen vorgestellten Nachbarschafts-Verfahren ist die Genauigkeit etwa gleich zu der Anzahl der Nachbarn. Eng beieinanderliegende Kanten, wie z. B. an Türrahmen, konnten hiermit gut detektiert werden. In dem künstlich erzeugten Modell wurden die Kanten zu 100% erkannt.

- Oberflächen-Varianz: Über die Oberflächen-Varianz können Flächen mit verschiedenen Eigenschaften gut unterschieden und detektiert werden. In der Theorie können beliebig viele Strukturen damit unterschieden werden. Bei dem künstlich erzeugten Modell konnte dies mit höchster Genauigkeit gezeigt werden. In der Praxis hängt es allerdings von der Genauigkeit des Scanners ab, wie viele Strukturen unterschieden werden können. Mit dem SICK-Laserscanner lassen sich nur die drei vorgeführten Typen (ebene Fläche, unebene Fläche und Kante) unterscheiden.

- Außenkanten: Die Berechnung der Außenkanten über die Eigenwerte hat weder im künstlich erzeugten Modell noch in der realen Szene die erhofften Ergebnisse gebracht. Im Modell wurden Ungenauigkeiten an der Viertelkreis-Krümmung festgestellt, die nur über die Auswertung einer größeren Nachbarschaft ausgeglichen werden konnten. Als einziges Verfahren mussten hier im künstlich erzeugten Modell mehr als acht Nachbarn hinzugezogen werden, um die gesuchten Bereiche zu markieren. In der

Tabelle 5.3: Bearing Angle: Laufzeiten

Anzahl Datenpunkte	Laufzeit
25.000	29 ms
50.000	65 ms
100.000	120 ms
200.000	227 ms
300.000	339 ms
400.000	443 ms

realen Szene haben sich die Probleme, welche bereits im Modell bestanden, noch weiter verstärkt.

- Normalenvektor: Die Detektion von Flächen mit der gleichen Ausrichtung im Raum ist außerordentlich positiv zu bewerten. In Kombination mit dem FCM-Algorithmus, welcher die resultierenden Normalenvektoren in verschiedene Gruppen unterteilt, konnten besonders in den realen Szenen positive Ergebnisse erzielt werden. Dies hängt auch damit zusammen, dass nur 25 Nachbarn für die reale Szene berechnet werden mußten und damit die Genauigkeit sehr hoch ist. Alle anderen vorgestellten Methoden benötigen mindestens 50 bis 75 Nachbarn, um vergleichbar gute Ergebnisse auf der realen Szene zu erzielen.

5.1.3 Bearing Angle

Die Repräsentation der Polardaten über den *Bearing Angle* und die anschließende Weiterverarbeitung mit Methoden der Bildverarbeitung ist ein effektives Verfahren. Die Parameter müssen einmalig bestimmt werden und sind abhängig von der Genauigkeit des verwendeten Laserscanners. Hauptsächlich werden die Auflösung und der Öffnungswinkel in horizontaler und vertikaler Richtung benötigt. Um die Genauigkeit des Ergebnisses zu beeinflussen, kann der verwendete Kantenfilter modifiziert oder ausgetauscht werden. Auch die Anzahl an Haar-Transformationen könnte je nach Auflösung des Laserscanner erhöht oder verringert werden. Die Komplexität dieses Verfahrens ist $O(n)$ bei n Datenpunkten. Dies ist Dank der 2D-Repräsentation möglich. In Tabelle 5.3 ist zu sehen, dass auch die Laufzeit nahezu linear mit der Anzahl der Datenpunkte steigt.

Die Laufzeit für 200.000 Datenpunkte beträgt nur 227 ms. Dies ist besonders erfreulich, da keine Optimierungen bezüglich der Laufzeit vorgenommen wurden. Während der Implementierung stand nur die Qualität des Ergebnisses im Vordergrund.

Die Qualität hängt von der Genauigkeit der erfassten Konturen ab. Um diese festzustellen, wurden zwei künstlich erzeugte Modelle verwendet. Im ersten Modell wurde eine

90°-Kante in Polarkoordinaten erstellt. Die Kante ist genau einen Datenpunkt dick. Damit soll die Genauigkeit bei der Detektion von *Turn Edges* bestimmt werden. Diese treten bei fortlaufenden Oberflächen auf. In Abbildung 5.1 sind das Distanzbild des Modells und daneben die detektierten Flächen dargestellt. Die detektierte Kante zwischen den beiden Flächen im rechten Ergebnisbild ist acht Pixel breit. Die Flächen links und rechts wurden dadurch jeweils vier Pixel kleiner. Somit lässt sich aussagen, dass bei der Detektion von *Turn Edges* eine Ungenauigkeit von mindestens acht Pixeln an den Kanten entlang entsteht.

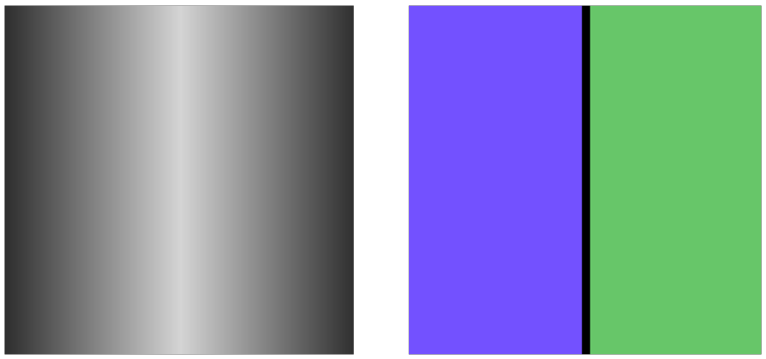

Abbildung 5.1: Turn Edge: Distanzbild einer 90°-Kante (links) und deren Detektion (rechts)

Mit dem zweiten Modell sollte die Genauigkeit bei der Detektion von *Jump Edges* bestimmt werden. Dazu wurde ein Rechteck mitten im Raum positioniert. In Abbildung 5.2 sind wieder das Distanzbild und die gefundenen Segmente dargestellt. Dieses Mal ist die gefundene Kante nur zwei Pixel dick. Dabei wurde dem Rechteck und dem Hintergrund jeweils ein Pixel durch die entstandene schwarze Kante entzogen. Die Detektion von *Jump Edges* ist somit wesentlich genauer als die von *Turn Edges*. Dies liegt daran, dass *Jump Edges* bereits mathematisch wesentlich einfacher zu erfassen sind als *Turn Edges*.

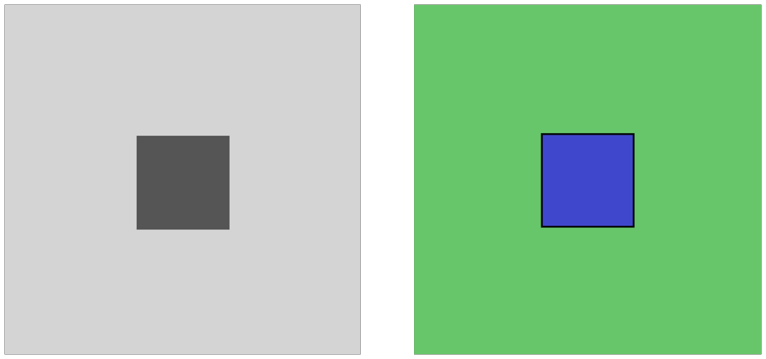

Abbildung 5.2: Jump Edge: Distanzbild eines Objektes im Raum und dessen Detektion

Tabelle 5.4: Bearing Angle: Auswertung

Szene	richtig positiv „Große Flächen“	richtig positiv „Kleine Flächen“
Büro 1	11 von 11	3 von 5
Büro 2	9 von 9	2 von 3
Büro 3	6 von 6	3 von 7
Flur	7 von 7	4 von 7
Person 1	11 von 11	1 von 1
Person 2	9 von 9	4 von 5

Abschließend wurden sechs reale Szenen manuell ausgewertet. Dabei wurde zwischen großen und kleinen Flächen unterschieden. „Kleine Flächen“ werden dabei definiert als Flächen, die in einer Richtung weniger als vierzig Pixel umfassen.

Die Auswertung in Tabelle 5.4 zeigt, dass „große Flächen“ mit 100%iger Genauigkeit gefunden werden können. Kleinere Flächen und Objekte hingegen können nicht zuverlässig erkannt werden. Dies liegt an der relativ groben Kantenerkennung, welche zuvor beschrieben wurde. Verbesserungsvorschläge dazu werden im nächsten Abschnitt beschrieben.

5.2 Diskussion

Der RANSAC-Algorithmus stellte einen guten Ausgangspunkt für die Segmentierung von 3D-Daten dar. Es wurden grundsätzliche Methoden entwickelt, um die Punktwolke zu manipulieren und gezielt Informationen daraus zu extrahieren. Darüber konnte der Autor ein Verständnis von den Eigenschaften der vorhandenen, durch den SICK-Laserscanner erzeugten Punktwolken entwickeln. Dies war sehr lehrreich und hat die Auswahl der weiteren Verfahren positiv beeinflusst. Negativ ist das nicht-deterministische Verhalten des RANSAC-Algorithmus aufgefallen. Es entstand durch die verwendete Zufallsvariable und wurde besonders bei der Auswertung deutlich. Dieses nicht-deterministische Verhalten konnte in dieser Arbeit auch nicht durch eine erhöhte Anzahl an Iterationen ausgeglichen werden. Dennoch sind die gesammelten Erfahrungen und die entstandenen Methoden nützlich für die weitere Entwicklung und Verbesserung von Segmentierungs-Verfahren.

Die Auswertung zeigt, dass es – unabhängig von der Laufzeit – schwierig ist, alle Objekte in der verrauschten Punktwolke zu finden. Dies ist besonders bei der Nachbarschaftsanalyse aufgefallen: Hier mussten sehr großzügige Nachbarschaften von bis zu 100 Punkten berechnet werden, um brauchbare Werte zu erhalten. Positiv ist der Normalenvektor aufgefallen: Dieser brauchte nur 25 Nachbarn, um großflächige Bereiche zuverlässig zu unterscheiden.

Die Detektion von ebenen Flächen ist jedoch nur ein Teilproblem der Segmentierung von 3D-Daten. Um alle Flächen und Objekte in einer Szene zu detektieren, musste eine vollkommen andere Taktik ausgewählt werden. Anstatt die kartesischen Koordinaten mit mathematischen Methoden zu verarbeiten, wurden die polaren Daten über die Bildverarbeitung ausgewertet. Dies brachte gleich mehrere Vorteile mit sich. Zum einen konnten die polaren Daten als 2D-Bild repräsentiert und mit Methoden aus der Bildverarbeitung bearbeitet werden. Zum anderen ist die Laufzeit gesunken, da die Verarbeitung von 2D-Daten deutlich schneller als die von 3D-Daten ist. Über den *Bearing Angle* konnte eine detaillierte und differenzierte Repräsentation der polaren Daten erstellt werden. Über diese wurden alle wichtigen Kanten in der jeweiligen Szene zuverlässig und schnell detektiert. Auffallend war, dass die Kanten dabei stark aufgebläht wurden. Dies hängt mit der eingesetzten Kombination aus Kantenfilter und Haar-Transformation zusammen und kann noch verbessert werden. Dazu gibt es zwei Möglichkeiten: Entweder wird die Qualität des Ausgangsbildes verbessert oder der Kantenfilter ausgetauscht. Es ist denkbar, die Qualität des Ausgangsbildes über die Modifikation der Detail-Koeffizienten, welche während der Haar-Transformation berechnet wurden, zu verbessern. Die Rücktransformation dieses Bildes sollte weniger Rauschen enthalten und eine bessere Kantenerkennung ermöglichen. Die Kantendetektion könnte weiterhin über die Modifikation der Schwellenwerte oder über den Einsatz eines anderen Kantenfilters, wie z. B. den Canny-Filter, verbessert werden.

6 Zusammenfassung und Ausblick

Diese Arbeit wurde für ein Projekt des Fraunhofer Instituts IAIS erstellt. Es geht um die Entwicklung eines neuen 3D-Laserscanners. Basierend auf diesem 3D-Laserscanner soll eine Sicherheits-Anwendung realisiert werden. Für eine Softwarekomponente – die Segmentierung von 3D-Daten – wurde der Stand der Forschung untersucht und es wurden drei Segmentierungs-Verfahren ausgewählt und implementiert. Die ersten beiden Verfahren, der *RANSAC-Algorithmus* und die *Nachbarschaftsanalyse*, sind rein mathematische Verfahren. Das dritte Verfahren, die Repräsentation der Polardaten über den *Bearing Angle*, stellt eine komplett andere Vorgehensweise dar. Hierbei werden zur Segmentierung der 3D-Daten hauptsächlich Methoden der Bildverarbeitung eingesetzt.

Der RANSAC-Algorithmus wurde zur Detektion von Ebenen eingesetzt. Um die Gesamtlaufzeit zu verringern, wurde ein neuer Parameter als Abbruchkriterium eingeführt. Bei der Nachbarschaftsanalyse wurde der Merkmalsvektor eines 3D-Datenpunktes durch verschiedene Berechnungen aus der lokalen Nachbarschaft erweitert. Mit diesem erweiterten Merkmalsvektor konnten bestimmte Merkmale aus den Gesamtdaten – wie Kanten, ebene und unebene Bereiche – extrahiert werden. Die Schwellenwerte wurden über den *Fuzzy-C-Means*-Algorithmus ohne weitere Parameter-Angabe aus den Merkmalsvektoren bestimmt. Für die Oberflächen-Varianz und den Normalenvektor ist die Kombination mit dem Fuzzy-C-Means-Algorithmus noch in keiner Literatur beschrieben. Beim dritten Verfahren, Segmentierung über den Bearing Angle, wurden keine eigenen Ideen angewandt, sondern lediglich das von den Autoren beschriebene Vorgehen umgesetzt.

Abschließend wurden alle Verfahren hinsichtlich der Laufzeit und Güte der Segmentierung ausgewertet. Es konnte gezeigt werden, dass die Laufzeit der Segmentierung bei dem Bearing-Angle-Verfahren im Mikrosekunden-Bereich liegt und sich daher für Echtzeitanwendungen eignet. Auch die Genauigkeit der richtig positiv erkannten Flächen ist bei diesem Segmentierungs-Verfahren, im Vergleich zu den anderen beiden, am höchsten.

Der Autor wird auch nach dieser Abschlussarbeit weiter an dem Thema Segmentierung von 3D-Daten im Rahmen des Projektes Fovea-3D arbeiten. Es gibt zwei verschiedene Ansatzpunkte zur Optimierung der Kantendetektion bei der Segmentierung über den Bearing Angle. Entweder wird dazu die Qualität des Ausgangsbildes verbessert oder der Kantenfilter modifiziert bzw. ausgetauscht. Es ist denkbar, die Qualität des Ausgangsbildes über die Modifikation der Detail-Koeffizienten, welche während der Haar-Transformation entstehen, zu verbessern. Die Rücktransformation dieses Bildes sollte weniger Rauschen enthalten und

eine bessere Kantendetektion ermöglichen. Die Kantendetektion könnte weiterhin über die Modifikation der Schwellenwerte oder über den Einsatz eines anderen Kantenfilters, wie z. B. dem Canny-Filter, verbessert werden.

Ein wichtiger Arbeitsschritt wird die Anpassung der Algorithmen an den Fovea-Scanner sein. Aufgrund der besseren Datenqualität ist zu erwarten, dass die Nachbarschaftsanalyse auch mit einer geringen Anzahl von Nachbarn eingesetzt werden kann. Dadurch werden die Ergebnisse genauer, und die Laufzeit sinkt. Für den Bearing Angle muss die neue Scan-Trajektorie berücksichtigt und der Filterprozess angepasst werden. Auch hier wird erwartet, dass sich die bessere Datenqualität positiv auf die Qualität der Ergebnisse der Segmentierung auswirkt.

Die Segmentierung ist als Vorstufe zur Detektion von 3D-Objekten im Raum zu sehen. Die in dieser Arbeit vorgestellten Segmentierungs-Verfahren bilden eine Basis für das langfristige Ziel des IAIS Institutes im Projekt Fovea-3D: die Objekterkennung.

Literaturverzeichnis

[Belton und Lichti 2006] Belton, David ; Lichti, Derek D.: Classification and segmentation of terrestrial laserscanner point clouds using local variance Information. In: *International Archives of Photogrammetry and Remote Sensing (IAPRS)* 36 (2006), S. 44–49

[Bhanu u. a. 1986] Bhanu, B. ; Lee, C.C. ; Henderson, T.: Range data processing: Representation of surfaces by edges. In: *International Conference on Pattern Recognition (ICPR)* (1986), S. 236–238

[Biosca und Lerma 2008] Biosca, Josep M. ; Lerma, José L.: Unsupervised robust planar segmentation of terrestrial laser scanner point clouds based on fuzzy clustering methods. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 63 (2008), Nr. 1, S. 84–98

[Filin 2002] Filin, Sagi: Surface clustering from airborne laser scanning data. In: *International Archives of Photogrammetry and Remote Sensing (IAPRS)* 34 (2002), S. 117–124

[Fischler und Bolles 1981] Fischler, Martin A. ; Bolles, Robert C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In: *Readings in computer vision: issues, problems, principles, and paradigms* 24 (1981), S. 381–395

[Gächter u. a. 2006] Gächter, Stefan ; Nguyen, Viet ; Siegwart, Roland: Results on range image segmentation for service robots. In: *Proceedings of the Fourth IEEE International Conference on Computer Vision Systems* (2006), S. 53–69

[Harati 2008] Harati, Ahad: *Simultaneous Localization and Mapping for Structured Indoor Environments*, Eidgenössische Technische Hochschule Zürich (ETHZ), Dissertation, 2008

[Harati u. a. 2006] Harati, Ahad ; Gächter, Stefan ; Siegwart, Roland: Fast range image segmentation for indoor 3D-SLAM / Swiss Federal Institute of Technology (ETH. Zürich). 2006. – Forschungsbericht

[Harati und Siegwart 2007] Harati, Ahad ; Siegwart, Roland: A new approach to segmentation of 2D range scans into linear regions. In: *Proceedings of The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2007), S. 283–288

[Hoover u. a. 1996] Hoover, Adam ; Jean-Baptiste, Gillian ; Jiang, Xiaoyi ; Flynn, Patrick J. ; Bunke, Horst ; Goldgof, Dmitry B. ; Bowyer, Kevin ; Eggert, David W. ; Fitzgibbon, Andrew ; Fisher, Robert B.: An experimental comparison of range image segmentation algorithms. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18 (1996), S. 673–689

[IAIS 2011] IAIS, Fraunhofer: *Fraunhofer IAIS Sick Scanner*. Januar 2011. – URL `http://www.iais.fraunhofer.de`

[Jähne 2005] Jähne, Bernd: *Digitale Bildverarbeitung*. 6. Auflage. Springer Verlag, 2005

[Jiang und Bunke 1994] Jiang, X.Y. ; Bunke, H.: Fast segmentation of range images into planar regions by scan line grouping. In: *Machine Vision and Applications* 7 (1994), S. 115–122

[Khalifa u. a. 2003] Khalifa, I. ; Moussa, M. ; Kamel, K.: Range image segmentation using local approximation of scan lines with application to CAD model acquisition. In: *Machine Vision Applications* 13 (2003), S. 263–274

[Linder 2010] Linder, Thorsten: *3D features extraction from 3D point clouds*, University of Applied Sciences Bonn-Rhein-Sieg, Diplomarbeit, 2010

[Microsoft 2011] Microsoft: *Microsoft Kinect Kamera*. Januar 2011. – URL `http://www.xbox.com/kinect`

[Nguyen u. a. 2007] Nguyen, Viet ; Gächter, Stefan ; Martinelli, Agostino ; Tomatis, Nicola ; Siegwart, Roland: A comparison of line extraction algorithms using 2D range data for indoor mobile robotics. In: *Autonomous Robots* 23 (2007), S. 97–111

[Nüchter und Hertzberg 2008] Nüchter, Andreas ; Hertzberg, Joachim: Towards semantic maps for mobile robots. In: *Robotics and Autonomous Systems* 56 (2008), S. 915–926

[Pauly u. a. 2002] Pauly, Mark ; Gross, Markus ; Kobbelt, Leif P.: Efficient simplification of point-sampled surfaces. In: *Proceedings of the Conference on Visualization '02* (2002), S. 163–170

[Rabbani u. a. 2006] Rabbani, T. ; Vosselman, G. ; Heuvel, F. A. van den: Segmentation of point clouds using smoothness constraint. In: *International Archives of Photogrammetry and Remote Sensing (IAPRS)* 36 (2006), S. 248–253

[Sappa und Devy 2001] Sappa, A. ; Devy, M.: Fast range image segmentation by an edge detection strategy. In: *3D Digital Imaging and Modeling, International Conference on* (2001), S. 292–299

[Segaran 2008] Segaran, Toby: *Kollektive Intelligenz - Analysieren, Programmieren & Nutzen.* O'REILLY, 2008

[Sithole und Vosselman 2003] Sithole, G. ; Vosselman, G.: Automatic Structure Detection in a Point Cloud of an Urban Landscape. In: *Proceedings 2nd GRSS/ISPRS Joint Workshop on Remote Sensing and Data Fusion over Urban Areas* (2003), S. 67–71

[Stollnitz u. a. 1996a] Stollnitz, Eric J. ; DeRose, Tony D. ; Salesin, David H.: Wavelets for Computer Graphics: A Primer, Part 1. In: *IEEE Computer Graphics Applications* 15 (1996), S. 76–84

[Stollnitz u. a. 1996b] Stollnitz, Eric J. ; Derose, Tony D. ; Salesin, David H.: *Wavelets for Computer Graphics: Theory and Applications.* Morgan Kaufmann Publishers, 1996

[Strang 2003] Strang, Gilbert: *Introduction to Linear Algebra.* 3rd. Wellesley-Cambridge Press, 2003

[Surmann u. a. 2003] Surmann, H. ; Nüchter, A. ; Hertzberg, J.: An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. In: *Robotics and autonomous systems* 45 (2003), Nr. 1–3, S. 181–198

[Tong u. a. 2004] Tong, Wai-Shun ; Tang, Chi-Keung ; Mordohai, Philippos ; Medioni, Gérard: First order augmentation to tensor voting for boundary inference and multiscale analysis in 3D. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26 (2004), S. 594–611

[Vosselman u. a. 2004] Vosselman, G. ; Gorte, B. G. H. ; Sithole, G. ; Rabbani, T.: Recognising structure in laser scanner point clouds. In: *International Archives of Photogrammetry and Remote Sensing (IAPRS)* 36 (2004), S. 33–38

[Xu und Ii 2005] Xu, Rui ; Ii, Donald W.: Survey of clustering algorithms. In: *IEEE Transactions on Neural Networks* 16 (2005), Nr. 3, S. 645–678